HP-UX Reference

Vol 1: Section 1

HP 9000 Series 300/800 Computers
HP-UX Release 7.0

HP Part Number 09000-90013

Hewlett-Packard Company
3404 East Harmony Road, Fort Collins, Colorado 80525

Legal Notices

The information contained in this document is subject to change without notice.

Hewlett-Packard Company makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard Company shall not be liable for errors contained herein or direct, indirect, special, incidental, or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty: A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. However, minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

To ensure that you receive new editions of this manual when changes occur, you may subscribe to the appropriate product support service, available through your HP sales representative.

September 1989. First Edition. This manual replaces manual part number 09000-90009, and is valid for HP-UX Release 7.0 on both Series 300 and Series 800 systems.

Notes

Table of Contents
for
Volume 1

Table of Contents
Volume 1

Section 1: User Commands

Table of Contents
Volume 1

Entry Name(Section) *name* **Description**

Table of Contents
Volume 1

Entry Name(Section) *name* **Description**

Table of Contents
Volume 1

Entry Name(Section) *name* **Description**

Entry Name(Section) *name* **Description**

Entry Name(Section) *name* **Description**

Entry Name(Section) *name* **Description**

Introduction to HP-UX Reference

INTRODUCTION

HP-UX is Hewlett-Packard Company's implementation of an operating system that is compatible with various industry standards. It is based on the AT&T UNIX* System V Release 2 operating system, but also includes important features from Berkeley Software Distribution 4.2. Improvements include literally thousands of bug fixes, plus enhanced capabilities and other features developed by HP. This combination makes HP-UX a very powerful, useful, and reliable operating system capable of supporting a wide range of applications ranging from simple text processing to sophisticated engineering graphics and design. It can also be readily used to control instruments and other peripheral devices through the HP-IB (IEEE-488) and HP-HIL interfaces as well as through GPIO interfacing. Real-time capabilities further expand HP-UX's flexibility as a powerful tool for solving tough problems in design, manufacturing, business, and other areas where responsiveness and performance are important. Extensive international language support enables HP-UX to interact with users in any of dozens of human languages. HP-UX interfaces easily with local area networks and resource-sharing facilities. By using industry-standard protocols, it also provides flexible interaction with other computers and operating systems. Optional software products extend HP-UX capabilities into a broad range of specialized needs.

This manual is not intended for use as a learning tool for beginners. It is a reference guide that is most useful to experienced users of UNIX or UNIX-like systems. If you are not already familiar with UNIX and HP-UX, refer to the series of Beginner's Guides, tutorial manuals, and other learning documents supplied with your system or available separately. System implementation and maintenance details are explained in the *HP-UX System Administrator Concepts* and *System Administrator Tasks* manuals normally furnished with each system.

MANUAL ORGANIZATION

Due to the size and complexity of the HP-UX operating system, the *HP-UX Reference* is divided into several sections contained in three volumes. Volume 1 contains User commands in Section 1. Sections 2 (System Calls) and 3 (Subroutine Libraries) are in Volume 2. These topics are of interest mainly to programmers. Volume 3 contains a potpourri of subjects not contained in the first two: Section 1M (System Administration Commands) is related to system installation and maintenance. Most commands in this section require super-user privilege before they can be used. Section 4 (File Formats) is of interest mostly to administrators and programmers. Section 5 (Miscellaneous Topics) contains a haphazard collection of miscellany of interest to widely various users. Section 7 (Device Files), like Section 4, contains information commonly needed by administrators and programmers. The Glossary in Section 9 is also of interest to a variety of users, but is not a simple beginner's definition of terms. Rather, it contains often very precise definitions of terms as used in the HP-UX environment. A separate Table of Contents and Index is provided for each respective volume to assist you in finding needed information. The index to Volume 1 also contains references to built-in features in the various command interpreters ("shells").

---

* UNIX and System V are trademarks of AT&T Bell Laboratories.

Section 1 (*User Commands*) describes programs that are usually invoked directly by users or from command language procedures, as opposed to system calls (Section 2) or subroutines (Section 3) that are called by user and application programs. Most heavily used commands reside in the directory **/bin** (for **bin**ary programs). Other less frequently used programs reside in **/usr/bin** to save space in **/bin** and to reduce search time for commonly-used commands. These directories are normally searched automatically by the command interpreter called a shell (*sh*(1), *csh*(1)), *ksh*(1)), or other command interpreter facilities. Most other Section 1 commands are located in **/lib** and **/usr/lib**. Refer to *hier*(5) and tutorial manuals supplied with your system for more information about file system structure.

Section 1M (*System Administration Commands*) describes commands used for system maintenance including boot processes, crash recovery, system integrity testing, and other needs. This section contains topics that pertain primarily to system administrator and super-user tasks.

Section 2 (*System Calls*) describes entries into the HP-UX kernel, including the C-language interface.

Section 3 (*Subroutines and Subroutine Libraries*) describes available subroutines that reside (in binary form) in various system libraries stored in directories **/lib** and **/usr/lib**. Refer to *intro*(3C) for descriptions of these libraries and the files where they are stored.

Section 4 (*File Formats*) documents the structure of various types of files. For example, the link editor output-file format is described in *a.out*(4). Files that are used only by a single command (such as intermediate files used by assemblers) are not described. C-language **struct** declarations corresponding to the formats in Section 4 can be found in directories **/usr/include** and **/usr/include/sys**.

Section 5 (*Miscellaneous*) contains a variety of information such as descriptions of header files, character sets, macro packages, and other topics.

Section 6 (*Games*) is absent because no games are currently supported on HP-UX.

Section 7 (*Device Special Files*) discusses the characteristics of special (device) files that provide the link between HP-UX and system I/O devices. The names for each topic usually refer to the type of I/O device rather than to the names of individual special files.

Section 9 (*Glossary*) is located in Volume 1 after Section 1M. It defines selected terms used in this manual.

Index An alphabetical listing of keywords and topics based on the NAME line on the first page of each manual page entry as well as other information. The right-hand column refers to the manual entry name (Section number is in parentheses).

Each section (except 9) contains a number of independent entries usually referred to as **manual entries**, but also called **manpages** or **manual pages**. Each manual entry consists of one or more printed pages, with the entry name and section number printed in the upper corners of each page. Entries are arranged alphabetically within each section of the reference, except for the introductory entry at the beginning of each section. Textual cross-references to other manual entries are of the form *pagename*(nL) where n is the section number and L is a letter representing a subsection where applicable. For example, *io_burst*(3I) refers to an entry in the I/O subroutine library (Section 3, subsection I) named *io_burst*.

Each printed page has two page numbers printed at the bottom of the page. The center page number is numbered starting over with page 1 at the beginning of each entry, and is placed between two dashes in normal typeface. Another number in bold is printed at the outside corner on each page. This page number is part of a continuous sequence as in a normal book, and is primarily for the convenience of support and manufacturing personnel. Normal users most typically locate entries by the alphabetical headings at the top of the page as when using a dictionary.

Some manual entries describe two or more commands or routines in a single entry. In such cases, the entry is not duplicated for each topic, but appears only once, usually arranged under the first keyword appearing in the NAME section of the entry. Occasionally, an entry name does not appear on the NAME line. In such instances, the name describes the keywords in more general terms such as the entry for *acct* or *acctsh* in Section 1M or *string* in Section 3.

SYSTEM STANDARDIZATION

This reference is based on extensive system-design control documents that have been used to ensure software compatiblity across HP-UX computer model lines. HP-UX is compatible with the AT&T UNIX System V Interface Definition (SVID), but also includes many popular features from the Berkeley Software Distributions (BSD), plus HP enhancements for international language support, real-time, graphics, and instrument control. HP-UX is one of the most reliable UNIX-like operating systems available from any supplier, due in part to thousands of bug fixes in software obtained from AT&T, Berkeley, and other originators, and extensive testing of the result in real-use environments.

As of this printing, HP-UX has been implemented on HP 9000 Series 200, 300, 500, and 800 computers. This document is valid for HP-UX Release 7.0 on Series 300 and 800 systems. The Series 300 AXE (Applications Execution Environment) supports a subset of the standard HP-UX operating system, and is documented in manuals provided with the AXE system.

Page Headers

Great effort has been expended to make the HP-UX operating system compatible between Series 300 and Series 800. However, there are significant differences between the two hardware series, making some features that may be highly desirable on one series inappropriate or useless on the other (such as *isl*(1M) or *vmstat*(1) commands which are supported on Series 800 but have no meaning for Series 300 systems). If an entry pertains to only one series, it is clearly indicated at the top of each page of that entry.

Some entries in this manual pertain to optional software subsystems (such as BASIC/UX, for example) that are not HP-UX. Such manual entries are clearly marked as such. They are provided here for the convenience of users whose systems have the optional software installed on them.

MANUAL ENTRY FORMATS

All manual entries follow an established topic format, but not all topics are included in each entry.

NAME — Gives the name(s) of the entry and briefly states its purpose.

SYNOPSIS — Summarizes the use of the entry or program entity being described. A few conventions are used:

Boldface strings are literals, and are to be typed exactly as they appear in the manual.

Italic strings represent substitutable argument names and names of manual entries found elsewhere in the manual.

Square brackets [] around an argument name indicate that the argument is optional.

Ellipses (...) are used to show that the previous argument can be repeated.

A final convention is used by the commands themselves. An argument beginning with a dash (−), a plus sign (+), or an equal sign (=) is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore it is unwise to have files names that begin with −, +, or =.

DESCRIPTION — Discusses the function and behavior of each entry.

NETWORKING FEATURES — Information under this heading is applicable only if you are using the network feature described there (such as NFS).

RETURN VALUE — Discusses various values returned upon completion of program calls.

DIAGNOSTICS — Discusses diagnostic indications that may be produced. Self-explanatory messages are not listed.

ERRORS — Lists error conditions and their corresponding error message or return value.

EXAMPLES — Provides examples of typical usage, where appropriate.

WARNINGS — Points out potential pitfalls.

DEPENDENCIES — Points out variations in HP-UX operation that are related to the use of specific hardware or combinations of hardware.

AUTHOR — Indicates the origin of the software documented by the manual entry. Unless noted otherwise, the author of an entry is AT&T. If the entry indicates Berkeley authorship, it refers to the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES Lists file names that are built into the program or command.

SEE ALSO Provides pointers to related topics.

EXTERNAL INFLUENCES Information under this heading pertains to programming for various spoken languages. Typical entries indicate support for single- and/or multi-byte characters, the effect of language-related environment variables on system behavior, and other related information.

BUGS Discusses known bugs and deficiencies, occasionally suggesting fixes.

STANDARDS CONFORMANCE For each command or subroutine entry point addressed by one or more of the following industry standards, this section lists the standard specifications to which that HP-UX component conforms.

The meanings of the notations used for the various standards are:

SVID2	System V Interface Definition Issue 2 (AT&T)
XPG2	X/Open Portability Guide Issue 2 (X/Open, Ltd.)
XPG3	X/Open Portability Guide Issue 3 (X/Open, Ltd.)
POSIX.1	IEEE Standard 1003.1-1988 (IEEE Computer Society) (Portable Operating System Interface for Computer Environments)
FIPS	151-1 Federal Information Processing Standard 151-1 (National Institute of Standards and Technology)

The table of contents included at the beginning of each volume contains a complete listing of all manual entries in the order they appear in each section, as well as alphabetically intermixed lists of all keywords that appear in manual entries covering multiple keywords. This combination provides an easy path for locating commands and features whose keyword names are not the same as the title heading on the corresponding manual entry.

HOW TO GET STARTED

This discussion provides a very brief overview of how to use the HP-UX system: how to log in and log out, how to communicate through your machine, how to run a program, and how to access an electronic copy of this manual on your system. If you are a beginning user, refer to other tutorial manuals for a more complete introduction to the system.

Logging In

To log in you must have a valid user name, which can be obtained from your system administrator. Press the **BREAK** key to get a **login:** message if it is not present.

When a connection has been established, the system displays **login:** on your terminal. Type your user name then press the **RETURN** key. If a password is required (strongly recommended!), the system asks for it, but does not print it on the terminal.

It is important that you type in your login name in lowercase if possible. If you type uppercase letters, HP-UX assumes that your terminal cannot generate lowercase letters, and treats subsequent uppercase input as lowercase. When you have logged in successfully, the shell displays a **$** prompt unless programmed for a different prompt (the shell is described below under the heading: "How to Run a Program").

For more information, consult *login*(1) and *getty*(1M), which discuss the login sequence in more detail, and *stty*(1), which tells you how to describe the characteristics of your terminal to the system (*profile*(4) explains how to accomplish this last task automatically every time you log in).

Logging Out

You can log out by typing an end-of-file indication (ASCII **EOT** character, usually typed as control-d) to the shell (see *csh*(1) and *ksh*(1) for information about **ignoreeof** if you are using C shell or Korn shell). The shell will terminate and the **login:** message will appear again.

How to Communicate Through Your Terminal

HP-UX gathers keyboard input characters and saves them in a buffer. The accumulated characters are not passed to the shell or other program until a **RETURN** is typed.

HP-UX terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is printing on your display or terminal. Of course, if you type during output, the output will have the input characters interspersed in it. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is severely overloaded or operating abnormally. When the read-ahead limit is exceeded, the system throws away **all** the saved characters.

Erase, Kill, and Output Stop/Resume Characters

By default, the character @ "kills" all characters typed before it on an input line from the terminal. The character # erases the last character typed. Successive uses of # will erase characters back to, but not beyond, the beginning of the input line; @ and # can be used as normal text characters by preceding them with \ (thus to erase a \, you need two #s). These default erase and kill characters can be changed, and usually are (see *stty*(1)).

The ASCII **DC3** (control-S) character can be used to temporarily stop output. It is commonly used on video terminals to suspend output to the display while you read what is already being displayed. You can then resume output to the display by typing a **DC1** (control-Q). When **DC1** (control-Q) or **DC3** (control-S) are used to suspend or restart output, they are not sent to the keyboard command-line buffer for passing to the program. However, any other characters typed on the keyboard are saved and used as input later in the program.

Interrupt and Quit Characters

The ASCII **DEL** character (sometimes labelled "rubout" or "rub") is not passed to programs, but instead generates an *interrupt signal*. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed*(1), for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editing operation without losing the file being edited.

The *quit* signal is generated by typing the ASCII **FS** (control-\) character. It causes a running program to terminate.

End-of-Line and Tab Characters

Besides adapting to the speed of the terminal, HP-UX tries to be intelligent as to whether you have a terminal with a new-line (line-feed) key, or whether it must be simulated with a carriage-return and line-feed pair. In the latter case, all incoming carriage-return characters are changed to line-feed characters (the standard line delimiter), and a carriage-return/line-feed pair is echoed to the terminal. If you get into the wrong mode, see *stty*(1).

Tab characters are used freely in HP-UX source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. The *stty*(1) command sets or resets this mode. The system assumes that tabs are set every eight character positions. The *tabs*(1) command can set tab stops on your terminal, if the terminal supports tabs.

How to Run a Program

When you have successfully logged into HP-UX, a program called a shell is monitoring input from your terminal. The shell accepts typed lines from the terminal, splits them into command names and arguments, then executes the command which is nothing more than an executable program. Usually, the shell looks first in your current directory (discussed below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space and/or tab characters.

When a program terminates, the shell will ordinarily regain control and prompt you with a **$** (unless redefined to some other prompt), to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh*(1).

The Current Directory

HP-UX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is assumed to be in that directory by default. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. The permissions you have in other directories and files will have been granted or denied to you by their respective owners, or by the system administrator. To change the current working directory use *cd*(1).

On-Line Reference Manual

The *HP-UX Reference* is also available on-line by using the *man*(1) command if the manual pages are present on the system. Refer to the *man*(1) manual page entry in Volume 1 for more information.

Path Names

To refer to files not in the current directory, you must use a path name. Full (absolute) path names begin with /, which is the name of the *root* directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a /), until finally the file name is reached (e.g., **/usr/ae/filex** refers to file **filex** in directory **ae,** while **ae** is itself a subdirectory of **usr**; **usr** is a subdirectory of the root directory). See the glossary (Section 9) for a formal definition of *path name.*

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (without a prefixed /). Generally, a path name can be used anywhere a file name is required.

Important commands that modify the contents of directories are *cp*(1), *mv*(1), and *rm*(1), which respectively copy, move (i.e., rename, relocate, or both), and remove files. To determine the status of files or the contents of directories, use *ls*(1). Use *mkdir*(1) for making directories, *rmdir*(1) for destroying them, and *mv*(1) for renameing them.

For a more complete discussion of the file system, see the references cited at the beginning of the *Introduction* above. It may also be useful to glance through Section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Writing a Program

To enter the text of a source program into an HP-UX file, use *vi*(1)(*preferred by most* users), *ex*(1), or *ed*(1). The three principal languages available under HP-UX are C (see *cc*(1)), FORTRAN (see *f77*(1)), and Pascal (see *pc*(1)). After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file named **a.out** in the current directory. Since the results of a subsequent compilation may also be placed in **a.out**, thus overwriting the current output, you may want to use *mv*(1) to give the output a unique name. If the program is written in assembly language, you will probably need to link library subroutines with it (see *ld*(1)). FORTRAN, C, and Pascal call the linker automatically.

When you have gone through this entire process without encountering any diagnostics, the resulting program can be run by giving its name to the shell in response to the prompt.

Your programs can receive arguments from the command line just as system programs do by using the *argc* and *argv* parameters. See the supplied C tutorial for details.

Text Processing

Almost all text is entered through a text editor. The editor preferred above all others provided with HP-UX is the *vi* editor. For batch-processing text files, the *sed* editor is very efficient. Other editors are used much less frequently. The *ex* editor is useful for handling certain situations while using *vi* but most other editors are rarely used except in various scripts.

The following editors are the same program masquerading under various names: *vi*, *view*, and *vedit* (see *vi*(1)), *ex* (see *ex*(1)), and *edit* (see *edit*(1)). For information about the streaming editor *sed*, see *sed*(1). The *ed* line editor is described in the *ed*(1) manual entry.

The commands most often used to display text on a terminal are *more*(1), *cat*(1), and *pr*(1). The *cat*(1) command simply dumps ASCII text on the terminal, with no processing at all. The *more*(1) command displays text on the terminal a screenful at a time, pausing for an acknowledgement from the user before continuing. The *pr*(1) command paginates text, supplies headings, and has a facility for multi-column output. It is most commonly used in conjunction with *lp*(1) to pipe formatted text to a line printer.

Inter-User Communication

Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be beneficial to learn about them, because someone else may direct them toward you. To communicate with another user currently logged in, *write*(1) can be used to transfer text directly to that user's terminal display (if permission to do so has been granted by the user). Otherwise, *mailx*(1) or *mail*(1) sends a message to that user's mailbox. The user is then informed by HP-UX that mail has arrived (if currently logged in) or mail is present (when he or she next logs in). Refer to the *mailx*, *mail*, and *write* manual entries in Section 1 for explanations of how each of these commands is used.

When you log in, a message-of-the-day may greet you before the first prompt.

HP-UX FILE SYSTEMS

HP-UX supports two basic file systems, depending on which series you are using. A third file system is described for historical purposes.

HFS High-performance File System. This file system format is implemented on all Series 300 and 800 systems, and on Series 200 beginning at Release 5.0.

SDF Structured Directory Format. This file system format is implemented on all Series 500 releases. Use the SDF utilities such as *sdfcp*(1), *sdfinit*(1), *sdfrm*(1), *sdffind*(1), etc. to access SDF files from other systems.

BFS Bell File System. This obsolete file system was implemented on the Integral PC and Series 200 systems prior to HP-UX Release 5.0. BFS files are no longer supported on HP-UX systems (starting at Release 7.0).

File system formats are transparent to most users, and are of little importance in most applications. Most of the time, formats only prevent direct reading of disks of a particular format on a machine that supports a different format. Thus, SDF cannot be read on an HFS system without using SDF utilities. However, an SDF-based system can readily transfer files to an HFS-based system over UUCP, LAN, or other supported data communication facilities.

When transportable data is needed, a tape cartridge, flexible disk, or optical read/write disk can be used. Flexible disks can be readily formatted and read or written in LIF (Logical Interchange Format) by using the *lifinit*, *lifcp*, *lifls*, *lifrename*, and *lifrm* commands in Section 1. LIF media is readily usable on other non-HP-UX systems that support the HP LIF format.

Introduction

Notes

Introduction

Notes

Section 1:
User Commands

NAME

intro – introduction to command utilities and application programs

DESCRIPTION

This section describes commands accessible by users, as opposed to system calls in Section (2), or subroutines in Section (3), which are accessible by user programs.

Command Syntax

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(s)] [*cmdarg*(s)]

where:

name	The name of an executable file.
option	– *noargletter*(s) or, – *argletter*<>*optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Path name (or other command argument) *not* beginning with – or, – by itself indicating the standard input.

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (for example, see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

SEE ALSO

getopt(1), exit(2), wait(2), getopt(3C), hier(5).

The introduction to this manual.

NAME

ada – Ada compiler

SYNOPSIS

ada [*options*] [*files*] *libraryname*

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada is the HP-UX Ada compiler. It accepts several types of file arguments:

(1) Arguments whose names end with **.ad***?*, where *?* is any single alphanumeric character, are taken to be Ada source files.

(2) The *libraryname* argument names an Ada library that must have been previously created using the *ada.mklib*(1) command. An Ada library is an HP-UX directory containing files that are used by the various components of the Ada compilation system. There is no required or standard suffix on the name of an Ada library.

The named source files are compiled, and each successfully compiled unit is placed in the specified Ada library by the compiler. When binding, or binding and linking, information is extracted from the Ada library to perform the bind and/or link operation. The Ada library must always be specified.

To ensure the integrity of the internal data structures of Ada libraries, libraries are locked for the duration of any operations which are performed on them. During compilation, *libraryname* is normally locked for updating and other Ada libraries can be locked for reading. During binding/linking, Ada libraries are only locked for reading.

If *ada* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Users are strongly discouraged from placing any additional files in Ada library directories. User files in Ada libraries are subject to damage by or might interfere with proper operation of *ada* and related tools.

(3) All other file arguments, including those whose names end with **.o** or **.a** are passed on to the linker *ld*(1) to be linked into the final program. It is not possible to link-only with the *ada*(1) command.

(4) Although shown in the preferred order above, *options*, *files*, and *libraryname* arguments can appear in any order.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada*) can be used (see *ada*(1), Environment Variables).

Normally this variable is defined and set in the systemwide shell startup files **/etc/profile** (for *sh*(1) and *ksh*(1)) and **/etc/csh.login** (for *csh*(1)). However, it can be set by a user either interactively or in a personal shell startup file, **.profile** (for *sh*(1) and *ksh*(1)) or **.cshrc** (for *csh*(1)).

ADA_PATH must contain the path name of the directory in which the Ada compiler components have been installed.

The value of this variable must be a rooted directory (i.e. it must begin with a /) and the directory specification must not end with a /.

ADAOPTS

The environment variable **ADAOPTS** can be used to supply commonly used (or default) arguments to *ada*. **ADAOPTS** is associated directly with *ada* (it is not used by any other component of the Ada compilation system).

Arguments can be passed to *ada* through the **ADAOPTS** environment variable, as well as on the command line. *Ada*(1) picks up the value of **ADAOPTS** and places its contents before any arguments on the command line. For example (in *sh*(1) notation),

```
$ ADAOPTS="-v -e 10"
$ export ADAOPTS
$ ada -L source.ada test.lib
```

is equivalent to

```
$ ada -v -e 10 -L source.ada test.lib
```

Compiler Options

The following options are recognized:

−a Store the supplied annotation string in the library with the compilation unit. This string can later be displayed by the unit manager. The maximum length of this string is 80 characters. The default is no string.

−b Display abbreviated compiler error messages (default is to display the long forms).

−c Suppress link phase and, if binding occurred, preserve the object file produced by the binder. This option only takes effect if linking would normally occur. Linking normally occurs when binding has been requested.

Use of this option causes an informational message to be displayed on standard error indicating the format of the *ld*(1) command that should be used to link the program. It is recommended that the user supply additional object (**.o**) and archive (**.a**) files and additional library search paths (**-l***x*) only in the places specified by the informational message.

When *ld* is later used to actually link the program, the following conditions must be met:

1. The Ada library specified when the bind was performed must be respecified.
2. The **.o** file generated by the binder must be specified before any HP-UX archive is specified (either explicitly or with **−l**).
3. If **−lc** is specified when linking, any **.o** file containing code that uses *stdio*(3S) routines must be specified before **−lc** is specified.

−d Cause the compiler to store additional information in the Ada library for the units being compiled for use by the Ada debugger (see *ada.probe*(1)). Only information required for debugging is saved; the source is not saved. (see **−D**.) By default, debut information is not stored.

Cause the binder to produce a debug information file for the program being bound so that the resulting program can be manipulated by the Ada debugger. The debug information file name will be the executable program file name with **.cui** appended. If the debug information file name would be truncated by the file system on which it would be created, an error will be reported.

Only sources compiled with the **−d** or **−D** option contribute information to the debug information file produced by the binder.

−e *<nnn>* Stop compilation after *<nnn>* errors (legal range 0..32767, default 50).

−i Cause any pending or existing instantiations of generic bodies in this Ada library, whose actual generic bodies have been compiled or recompiled in another Ada library, to be compiled (or recompiled) in this Ada library.

This option is treated as a special "source" file and the compilation is performed when the option is encountered among the names of any actual source files.

Any pending or existing instantiations in the same Ada library into which the actual generic body is compiled (or recompiled), do not need this option. Such pending or existing instantiations are automatically compiled (or recompiled) when the actual generic body is compiled into the same Ada library.

Warning: Compilation (or recompilation) of instantiations either automatically or by using this option only affects instantiations stored as separate units in the Ada library (see **−u**). Existing instantiations which are "inline" in another unit are not automatically compiled or recompiled by using this option. Units containing such instantiations must be explicitly recompiled by the user if the actual generic body is recompiled.

−k Cause the compiler to save an internal representation of the source in the Ada library for use by the Ada cross referencer *ada.xref*(1). By default, the internal representation is not saved.

−l*x* Cause the linker to search the HP-UX library named either **/lib/lib***x***.a** (tried first) or **/usr/lib/lib***x***.a** (see *ld*(1)).

-m *<nnn>* The supplied number is the size in Kbytes to be allocated at compile time to manipulate library information. The range is 500 to 32767. The default is 500. The default size should work in almost all cases. In some extreme cases involving very large programs, increasing this value will improve compilation time. Also, if the value is to small, STORAGE_ERROR can be raised.

−n Cause the output file from the linker to be marked as shareable (see **−N**). For details refer to *chatr*(1) and *ld*(1).

−o *outfile* Name the output file from the linker *outfile* instead of **a.out**. In addition, if used with the **−c** option, name the object file output by the binder *outfile***.o** instead of **a.out.o**. If debugging is enabled (with **−d** or **−D**), name the debug information file output by the binder *outfile***.cui** instead of **a.out.cui**.

The object file output by the binder is deleted if **−c** is not specified.

−q Cause the output file from the linker to be marked as demand loadable (see **−Q**). For details refer to *chatr*(1) and *ld*(1).

−r *<nnn>* Set listing line length to *<nnn>* (legal range 60..255, default 79). This option applies to the listing produced by both the compiler and the binder (see **−B**, **−L** and **−W b,−L**).

−s Cause the output of the linker to be stripped of symbol table information (see *ld*(1) and *strip*(1)).

−t *c,name* Substitute or insert subprocess *c* with *name* where *c* is one or more of a set of identifiers indicating the subprocess(es). This option works in two modes: 1) if *c* is a single identifier, *name* represents the full path name of the new subprocess; 2) if *c* is a set of (more than one) identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path name of the new subprocesses.

The possible values of *c* are the following:

b binder body (standard suffix is *adabind*)
c compiler body (standard suffix is *adacomp*)
0 same as *c*
l linker (standard suffix is *ld*)

−u Cause instantiations of generic program unit bodies to be stored as separate units in the Ada library (see **−i**).

If **−u** is not specified, and the actual generic body has already been compiled when an instantiation of the body is compiled, the body generated by the instantiation is stored "inline" in the same unit as its declaration.

If **−u** is specified, or the actual generic body has not already been compiled when an instantiation of the body is compiled, the body generated by the instantiation is stored as a separate unit in the Ada library.

The **−u** option may be needed if a large number of generic instantiations within a given unit result in the overflow of a compiler internal table.

Specifying **−u** reduces the amount of table space needed, permitting the compiler to complete. However it also increases the number of units used within the Ada library, as well as introduces a small amount of overhead at execution time, in units which instantiate generics.

−v Enable verbose mode, producing a step-by-step description of the compilation, binding, and linking process on standard error.

−w Suppress warning messages.

−x Perform syntactic checking only. The *libraryname* argument must be supplied, although the Ada library is not modified.

−B Causes the compiler to produce a compilation listing, suppressing page headers and the error summary at the end of the compilation listing. This is useful when comparing a compilation listing with a previous compilation listing of the same program, without the page headers causing mismatches. This option can not be specified in conjunction the **−L** option.

−C Only generate checks for stack overflow. Use of this option may result in erroneous (in the Ada sense) program behavior. In addition, some checks (such as those automatically provided by hardware) might not be suppressed. See the Users Guide for more information.

−D Cause the compiler to store additional information in the Ada library for the units being compiled, for use by the Ada debugger (see *ada.probe*(1)). In addition to saving information required for debugging, an internal representation of the actual source is saved. This permits accurate source level debugging at the expense of a larger Ada library if the actual source file changes after it is compiled. (see **−d**.) By default, neither debug information nor source information is stored.

Cause the binder to produce a debug information file for the program being bound so that the resulting program can be manipulated by the Ada debugger. The debug information file name is the executable program file name with **.cui** appended. If the debug information file name would be truncated by the file system on which it would be created, an error will be reported.

Only sources compiled with the **−d** or **−D** option contribute information to the debug information file produced by the binder

−G Generate code but do not update the library. This is primarily intended to allow one to get an assembly listing (with -S) without changing the library. The *libraryname* argument must be supplied, although the Ada library is not modified.

−I Suppress all inlining. No procedures or functions are expanded inline and **pragma inline** is ignored. This also prevents units compiled in the future (without this option in effect) from inlining any units compiled with this option in effect.

−L Write a program listing with error diagnostics to standard output. This option can not be specified in conjunction with the **−B** option.

−M <*main*> Invoke the binder after all source files named in the command line (if any) have been successfully compiled. The argument <*main*> specifies the entry point of the Ada program; <*main*> must be the name of a parameterless Ada library level procedure.

The library level procedure <*main*> must have been successfully compiled into (or linked into) the named Ada library, either by this invocation of *ada* or by a previous invocation of *ada* (or *ada.umgr*(UTIL)).

The binder produces an object file named **a.out.o** (unless **−o** is used to specify an alternate name), only if the option **−c** is also specified. The object file produced by the binder is deleted unless the option **−c** is specified. Note that the alternate name is truncated, if necessary, prior to appending **.o**.

−N Cause the output file from the linker to be marked as unshareable (see **−n**). For details refer to *chatr*(1) and *ld*(1).

−O Invoke the optimizer. This is equivalent to **+O eioE**.

−P <*nnn*> Set listing page length to <*nnn*> lines (legal range 10..32767 or 0 to indicate no page breaks, default 66). This length is the total number of lines listed per listing page. It includes the heading, header and trailer blank lines, listed program lines, and error message lines. This option applies to the listing produced by both the compiler and the binder (see **−L** and **−W b,−L**).

−Q Cause the output file from the linker to be marked as not demand loadable (see **−q**). For details refer to *chatr*(1) and *ld*(1).

−R Suppress all runtime checks. However, some checks (such as those automatically provided by hardware) might not be suppressed. Use of this option may result in erroneous (in the Ada sense) program behavior.

−S Write an assembly listing of the code generated to standard output. This output is not in a form suitable for processing with *as*(1).

−W *c,arg1[,arg2,...,argN]*

Cause *arg1* through *argN* to be handed off to subprocess *c*. The *argi* are of the form *−argoption[,argvalue]*, where *argoption* is the name of an option recognized by the subprocess and *argvalue* is a separate argument to *argoption* where necessary. The values that *c* can assume are those listed under the **−t** option as well as **d** (driver program).

For example, the specification to pass the **-r** (preserve relocation information) option to the linker would be:

-W l,-r

For example, the following:

-W b,-m,10,-s,2

sends the options **−m 10 −s 2** to the binder. Note that all the binder options can be supplied with one **-W**, (more than one **−W** can also be used) and that any embedded spaces must be replaced with commas. Note that **−W b** is the only way to specify binder options.

The **−W d** option specification allows additional implementation−specific options to be recognized and passed through the compiler driver to the appropriate subprocess. For example,

-W d,-O,eo

sends the option **−O eo** to the driver, which sends it to the compiler so that the **e** and **o** optimizations are performed. Furthermore, a shorthand notation for this mechanism can be used by prepending the option with +; as follows:

+O eo

This is equivalent to **−W d,−O,eo**. Note that for simplicity this shorthand is applied to each implementation−specific option individually, and that the *argvalue* (if any) is separated from the shorthand *argoption* with white space instead of a comma.

−X Perform syntactic and semantic checking. The *libraryname* argument must be supplied, although the Ada library is not modified.

Binder Options

The following options can be passed to the binder using **-W b,...**:

−W b,−b At execution time, interactive input blocks if data is not available. All tasks are suspended if input data is not available. This option is the default if the program contains no tasks (see **−W b,−B**).

−W b,−k Keep uncalled subprograms when binding. The default is to remove them.

−W b,−m,<*nnn*>
Set the initial program stack size to <*nnn*> units of 1024 bytes (legal range 1..32767, default 10 units = 10 * 1024 bytes = 10240 bytes). The value is rounded up to the next multiple of 2.

−W b,−s,<*nnn*>
Cause round-robin scheduling to be used for tasking programs. Set the time slice to <*nnn*> tens of milliseconds (legal range 1..32767 or 0 to turn off time slicing). By default, round-robin scheduling is enabled with a time slice of 1 second (<*nnn*> = 100).

The time slice granularity is 20 milliseconds (<*nnn*> = 2).

−W b,−t,<*nnn*>
Set task stack size of created tasks to <*nnn*> units of 1024 bytes.

Set the initial (and maximum) task stack size (legal range 1..32767, default 8 units = 8 * 1024 bytes = 8192 bytes).

−W b,−w Suppress warning messages.

−W b,−x Perform consistency checks without producing an object file and suppress linking. The **−W b,−L** option can be used to obtain binder listing information when this option is specified (see **−W b,−L** below).

−W b,−B At execution time, interactive input does not block if data is not available. Only the task(s) doing interactive input are suspended if input data is not

available. This option is the default if the program contains tasks (see **−W b,−b**).

−W b,−L Write a binder listing with warning/error diagnostics to standard error.

−W b,−T Suppress procedure traceback in response to runtime errors and unhandled exceptions.

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked for only a short time when libraries within them are manipulated. However, multiple Ada libraries might need to be locked for longer periods during a single operation. If more than one library is locked, *ada* places an exclusive lock on one library, so it can be updated, and a shared lock on the other(s), so that they can remain open for read-only purposes.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part that holds the lock. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system desiring to read from the Ada family or library.

If *ada* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you may remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it will permit access by other tools that might find the contents invalid or that might damage the Ada family or Ada library.

DIAGNOSTICS

The diagnostics produced by *ada* are intended to be self-explanatory. Occasional messages might be produced by the linker.

If a program listing (**−B** or **-L**) and/or generated code listing (**-S**) is requested from the compiler, this listing as well as compiler error messages are written to standard output.

If neither a program listing nor a generated code listing is requested from the compiler, erroneous source lines and compiler error messages are written to standard error.

If a binder listing is requested from the binder (with **-W b,-L**), the binder listing as well as binder error messages are written to standard error.

If a binder listing is not requested from the binder, binder error messages are written to standard error.

Errors detected during command line processing or during scheduling of the compiler, binder, or linker, are written to standard error. If any compiler, binder, or linker errors occur, *ada* writes a one-line summary to standard error immediately before terminating.

WARNINGS

Options not recognized by *ada* are not passed on to the linker. The option **−W l,***arg* can be used to pass such options to the linker.

Ada does not generate an error or warning if both optimization and debugging are requested. However, *ada.probe* is only capable of limited debugging of optimized code. Certain *ada.probe* commands may give misleading or unexpected results. For example, object values may be stored in registers; therefore the value displayed from memory may be incorrect. For this

reason, the ability to examine or modify objects and expressions may be impaired. Dead code elimination or code motion may affect single step execution or prevent breakpoints from being set on specific source lines.

DEPENDENCIES

Series 300

The binder option **−W b,−T** also causes traceback tables to be excluded from final executable file.

The following options are specific to the Series 300:

+O *what* Selectively invoke optimizations. The *what* argument must be specified, and indicates which optimizations should be performed.

The *what* argument can be a combination of the letters **e**, **i**, **o**, and **p**. Either **e** or **p**, but not both, can be specified. All other combinations are permitted; however, at most one of each letter can be specified.

e Same as *p* (below).

i Permit procedures and functions not declared with **pragma inline** to be expanded inline at the compiler's discretion. Only procedures and functions in the current source file are considered.

Procedures and functions declared with **pragma inline** are always considered candidates for inline expansion unless **−I** is specified; this optimization only causes additional procedures and functions to be considered.

o Peephole optimizations are performed on the final object code.

p Optimizations are performed to remove unnecessary checks, optimize loops, and remove dead code.

E Same as *P* (below).

P Optimizations are performed on common subexpressions and register allocation.

+h <*type*> Bind/link the program to use the specified <*type*> of hardware floating point assist for user code floating point operations (see **+H**). The two <*type*>'s currently supported are **68881** (the 68881 math coprocessor) and **68882** (the 68882 math coprocessor). The code generated is the same for either <*type*>. This is the default if the host system provides a 68881 or a 68882 coprocessor.

+i <*type*> Compile the program to inline the specified <*type*> of hardware floating point assist for user code floating point operations (see **+I**). The two <*type*>'s currently supported are **68881** (the 68881 math coprocessor) and **68882** (the 68882 math coprocessor). The code generated is the same for either <*type*>. This is the default if the host system provides a 68881 or a 68882 coprocessor.

+H Bind/link the program to use software floating point routines for user code floating point operations (see **+h**). This is the default if the host system does not provide a 68881 or 68882 coprocessor.

+I Compile the program to make calls to a math library for user code floating point operations (see **+i**). The **+h** option is then used at bind/link time to specify whether hardware or software is used for floating point operations. This is the default if the host system does not provide a

68881 or 68882 coprocessor.

Unlike other Series 300 compilers, it is not possible to link-only using the *ada*(1) command. If separate linking is desired, use the *ld*(1) command.

A successful bind produces a (non-executable) **.o** file. The **.o** file is normally deleted unless the compiler option **−c** is specified.

AUTHOR

Ada was developed by HP and Alsys.

FILES

file.ad?	input file (Ada source file).
libraryname	user Ada library (created using *ada.mklib*(1)) in which compiled units are placed by a successful compilation and from which the binder extracts the units necessary to build a relocatable file for *ld*(1). Temporary files generated by the compiler are also created in this directory and are automatically deleted on successful completion. Users are strongly discouraged from placing any additional files in Ada library directories. User files in Ada libraries are subject to damage by or may interfere with proper operation of *ada* and related tools.
file.o	binder-generated object file or user-specified object file relocated at link time.
a.out	linked executable output file.
file.cui	binder-generated debug information file.
$ADA_PATH/ada	Ada compilation control program.
$ADA_PATH/adacomp	Ada compiler.
$ADA_PATH/adabind	Ada binder.
$ADA_PATH/ada_environ	Ada environment description file.
$ADA_PATH/adaargu	Ada argument formatter.
$ADA_PATH/alternate	Ada predefined library (sequential version).
$ADA_PATH/installation	Ada installation family.
$ADA_PATH/public	Ada public family.
$ADA_PATH/err_tpl	Ada compiler/binder error message files.
$ADA_PATH/predeflib	Ada predefined library, tasking version.
$ADA_PATH/libada020.a	Ada run-time HP-UX library (MC68020).
$ADA_PATH/libada881.a	Ada run-time HP-UX library (MC68881).
/lib/crt0.o	C run-time startup.
/lib/libc.a	HP−UX C library.
/lib/libm.a	HP−UX math library

SEE ALSO

ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300),

Ada Tools Manual,

Reference Manual for the Ada Programming Language (ANSI/MIL-STD-1815A),

Reference Manual for the Ada Programming Language, Appendix F (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.fmgr – create, examine, and modify an Ada family

SYNOPSIS

ada.fmgr

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.fmgr, the Ada family manager, can be used to create, examine, and modify the contents of an Ada family. An Ada family is a group of Ada libraries that are all visible to one another.

Units in a library within a particular family can be dependent on units in another library in the same family, and a program can be composed (bound) using units from any number of libraries in a given family.

Libraries that are members of different families are normally invisible to each other and cannot access each other's units. There is one exception to this rule: libraries belonging to the INSTALLATION family (a family provided with the Ada compilation system) are accessible to libraries in any other family.

Ada.fmgr is an interactive tool that, when executed, prompts with "Family_Manager." Commands are then entered to examine or modify the contents of a family.

There are two families that are special to the the Ada compilation system. They are the **PUBLIC** and **INSTALLATION** families. These two families can only be created, moved, and removed using *ada.fmgr*, **and can not be manipulated with the** *ada.mkfam*, *ada.mvfam*, and *ada.rmfam* commands. However, other operations may be performed on the INSTALLATION and PUBLIC families using the appropriate commands and options.

For the syntax and semantics of the commands, refer to the "*Ada User's Guide*," or use the available online help facility in the family manager.

The following commands can be typed in response to the "Family_Manager" prompt:

Default	Set default values for commands.
Erase	Remove an Ada family.
Global	Set global characteristics for commands.
Help	Obtain help on the various *ada.fmgr* commands.
Invoke	Read *ada.fmgr* commands from a file.
Mend	Rebuild an Ada family from a list of libraries.
New	Create a new Ada family.
Quit	Leave *ada.fmgr*.
Rename	Rename an Ada family.
System	Execute a shell command.
Unlock	Unlock a locked Ada family.

The *help* command provides online help for all commands.

If the file **.ada.fmgrrc** exists in the user's home directory, its commands are executed before the first prompt is issued.

If the file **.ada.fmgrdf** exists in the user's home directory, its commands are executed after those in **.ada.fmgrrc** (if present) and before the first prompt is issued. The commands in this file are

DEFAULT commands saved automatically by the use of the DEFAULTS => KEEP parameter of the QUIT command.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.fmgr*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. However, an Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.fmgr* cannot obtain a lock after a suitable number of retries,, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library may be locked, but the locking program(s) may have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

WARNINGS

Care should be taken when specifying the names PUBLIC, INSTALLATION, and OLD. These keywords are recognized by the family manager as having special meanings, and their use as local family names is strongly discouraged.

AUTHOR

Ada.fmgr was developed by HP and Alsys.

FILES

$HOME/.ada.fmgrrc	User start up commands.
$HOME/.ada.fmgrdf	User start up defaults.
$ADA_PATH/ada.fmgr	Ada interactive family manager command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.fmgr.hlp	Ada family manager help command data.
$ADA_PATH/ada.fmgr.btl	Ada family manager command tables.

SEE ALSO

ada(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.format – Ada source program formatter

SYNOPSIS

ada.format [**-c** | **-i** | **-s**] [**-k**] [**-m**] [**-p**] [**-t** *n*] *file*

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.format takes an Ada source *file* and produces a listing of the formatted source on the standard output. *Ada.format* requires the source *file* suffix to be **.ad***?*, where *?* is any single alphanumeric character. If the source contains any syntax errors, *ada.format* terminates with a diagnostic error message and the line number indicating where the first error occurred. No listing of the formatted source is generated.

By default, *ada.format* formats the Ada source *file* according to the syntactic structure used in the *Reference Manual for the Ada Programming Language*. The extent and nature of the transformations performed can be optionally controlled by specifying *ada.format* options and by using pragma INDENT within the source file to protect regions of source from being formatted.

The following options are recognized:

–c Capitalize the first letter of each word component (separated by underscore) of an identifier. By default, identifiers are converted to uppercase. Only one of the options **-c**, **–i** or **–s** may be selected.

–i All identifiers are converted to lowercase. By default, identifiers are converted to uppercase. Only one of the options **-c**, **–i** or **–s** may be selected.

–k All Ada keywords are converted to uppercase. By default, keywords are converted to lowercase.

–m Parameters of unspecified mode (by implication, mode **in**) are converted to explicit **in** parameters. Note that if either the specification or body of a unit is formatted with this option while the other is not, the compiler indicates that the two parameter specifications do not conform, according to the *Reference Manual for the Ada Programming Language* [6.3.1 (8)].

–p The source structure is preserved so that each source line is treated as an indivisible entity. The source structure is indented according to its lexical elements, but individual source lines are not restructured.

–s All of the identifiers are preserved as in the source file. By default, identifiers are converted to uppercase. Only one of the options **-c**, **–i** or **–s** may be selected.

–t *n* Use *n* spaces for indentation; legal range 2 .. 6, default 3.

Use of the pragma INDENT in the source file has the following effects:

pragma INDENT (OFF);	Disable *ada.format* so that subsequent source lines are not formatted.
pragma INDENT (ON);	Enable *ada.format* so that subsequent source lines are formatted.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.format*) can be used (see *ada*(1), Environment Variables).

AUTHOR

Ada.format was developed by HP and Alsys.

FILES

$ADA_PATH/ada.format Ada source program formatter command.
$ADA_PATH/ada.format.bin
Ada source program formatter subprocess.

SEE ALSO

ada(1), ada.fmgr(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300),

Ada Tools Manual (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.funlock – unlock an Ada family

SYNOPSIS

ada.funlock -I | -P | *familyname*

Remarks:

This command requires installation of optional ADA software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.funlock removes all locks on an Ada family.

It is intended that *ada.funlock* be used to remove inappropriate locks. Such locks might be left by abnormally terminated components of the Ada compilation system or by system crashes.

Unlocking an Ada family requires write permission the family directory.

Extreme care must be used when unlocking an Ada family. If any component of the Ada compilation system is reading the Ada family, removing locks may cause that component to encounter internal inconsistencies and fail. In addition, if any component of the Ada compilation system is updating the Ada family, removing locks may cause that component to corrupt the Ada family and fail.

Only one of the options **-I**, **-P**, and *familyname* can be specified.

The following options are recognized:

–I	Unlock the installation family.
–P	Unlock the public family.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.funlock*) can be used (see *ada*(1), Environment Variables).

DIAGNOSTICS

Ada.funlock returns exit status 0 if the Ada family was successfully unlocked. Otherwise a diagnostic is issued and a non-zero exit status is returned.

AUTHOR

Ada.funlock was developed by HP and Alsys.

FILES

$ADA_PATH/ada.funlock	Ada family unlock command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.fmgr.hlp	Ada family manager help command data.
$ADA_PATH/ada.fmgr.btl	Ada family manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

ada.lmgr – create, examine, and modify an Ada library

SYNOPSIS

ada.lmgr [**-I** | **-P** | *familyname*]

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.lmgr, the Ada library manager, can be used to create, examine, and modify the contents of an Ada library.

Units in a library within a particular family can be dependent on units in another library in the same family and a program can be composed (bound) using units from any number of libraries in a given family.

Libraries that are members of different families are normally invisible to each other and cannot access each other's units. There is one exception to this rule: libraries belonging to the INSTALLATION family (a family provided with the Ada compilation system) are accessible to libraries in any other family.

Only one of the options **-I**, **-P**, or *familyname* may be specified. If no family is specified, the PUBLIC family is assumed.

The following options are recognized:

–I	Manipulate libraries in the installation family.
–P	Manipulate libraries in the public family.

Ada.lmgr is an interactive tool that, when executed, prompts with "Library_Manager." Commands are then entered to examine or modify the contents of a library.

For the syntax and semantics of the commands, refer to the "*Ada User's Guide,*" or use the available online help facility in the library manager.

The following commands can be typed in response to the "Library_Manager" prompt:

Default	Set default values for commands.
Erase	Remove an Ada library.
Global	Set global characteristics for commands.
Help	Obtain help on the various *ada.lmgr* commands.
Invoke	Read *ada.lmgr* commands from a file.
List	List the Ada libraries in an Ada family.
New	Create a new Ada library.
Other	Change the current family.
Protect	Mark the current library as either read-only or update-allowed.
Quit	Leave *ada.lmgr*.
Rename	Rename an Ada library.
System	Execute a shell command.
Unlock	Unlock a locked Ada library.
View	Display information about the current family.

The *help* command provides online help for all commands.

If the file **.ada.lmgrrc** exists in the user's home directory, its commands are executed before the first prompt is issued.

If the file **.ada.lmgrdf** exists in the user's home directory, its commands are executed after those in **.ada.lmgrrc** (if present) and before the first prompt is issued. The commands in this file are DEFAULT commands saved automatically by the use of the DEFAULTS => KEEP parameter of the QUIT command.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.lmgr*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. However, an Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.lmgr* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library may be locked, but the locking program(s) may have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

AUTHOR

Ada.lmgr was developed by HP and Alsys.

FILES

$HOME/.ada.lmgrrc	User start up commands.
$HOME/.ada.lmgrdf	User start up defaults.
$ADA_PATH/ada.lmgr	Ada interactive library manager command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

INTERNATIONAL SUPPORT

8-bit file names.

NAME

ada.lsfam – list information about an Ada family

SYNOPSIS

ada.lsfam [**-I** | **-P** | *familyname*]

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.lsfam lists pertinent information about the family *familyname*

The information *Ada.lsfam* prints includes the full path name of the family being listed, whether the family is a user or installation family, the associated installation family if it is a user family, the family creation date, the last modification date of the family and the number of libraries in the family.

Only one of the options **-I**, **-P**, and *familyname* can be specified. If no family is specified, the PUBLIC family is assumed.

The following options are recognized:

–I	List information about the INSTALLATION family.
–P	List information about the PUBLIC family.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.lsfam*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.lsfam* cannot obtain a lock after suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you may remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.lsfam returns exit status **0** if the Ada family was successfully listed; otherwise, a diagnostic is issued and a non-zero exit status is returned.

AUTHOR

Ada.lsfam was developed by HP and Alsys.

FILES

$ADA_PATH/ada.lsfam	Ada family list command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lslib(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1), ada.make(1),

Ada User's Guide (Series 300).

INTERNATIONAL SUPPORT

8-bit file names.

NAME

ada.lslib – list the libraries in an Ada family

SYNOPSIS

ada.lslib [**-f** | **-b**] [**-I** | **-P** | *familyname*]

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.lslib lists the Ada libraries in the family specified.

The user can request either a full or a brief listing of the libraries in the specified family. The default is a brief listing.

Only one of the options **-f** and **-b**, and one of the options **-I**, **-P** and *familyname* can be specified. If no family is specified, the PUBLIC family is assumed.

The following options are recognized:

-b	Generate a brief listing.
-f	Generate a full listing.
-I	List libraries in the INSTALLATION family.
-P	List libraries in the PUBLIC family.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.lslib*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.lslib* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you may remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.lslib returns exit status **0** if Ada libraries were successfully listed; otherwise, a diagnostic is issued and a non-zero exit status is returned.

AUTHOR

Ada.lslib was developed by HP and Alsys.

FILES

$ADA_PATH/ada.lslib	Ada library list command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

INTERNATIONAL SUPPORT

8-bit file names.

NAME

ada.make – determine compilation order, update Ada libraries

SYNOPSIS

ada.make [*options*] [*library*] [*files*]

Remarks:

This command requires installation of optional ADA software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.make reads Ada source files and libraries in order to produce a minimal, correctly ordered sequence of commands necessary to compile or recompile a set of Ada compilation units.

The *library* argument names an Ada library that must have already been created using the *ada.mklib*(1) command.

Files specifies a set of Ada source files. For an application that consists of a large number of source files, the system-imposed limit on argument list length may be exceeded. For this reason, patterns may be given as file arguments. A pattern may contain the character *, which matches any sequence of zero or more characters. The character * must be escaped from the shell.

If neither *library* nor *files* is specified, *ada.make* operates on the files in the current directory whose names end with **.ad***?*, where *?* is any single alphanumeric character (see *Source vs. Library Mode*, below).

The command sequence generated by *ada.make* is written to file **ada.make.out** (or to the file named by the **-o** option). The commands in this file can be executed by *sh*(1) using the command "sh ada.make.out".

Options

The following options are recognized:

–a Produce commands to compile units regardless of whether they are out-of-date.

–b Emit a command to bind the unit specified by the option **–r** *unitpattern* (only allowed if *unitpattern* specifies a single unit).

–f *file* Read *library* and *files* arguments from *file*. A file name of - denotes the standard input.

–i Check the validity of all required units, regardless of where they reside in the family. Output may contain commands to acquire links or compile source files into any library in the family (see *Multi-Library Operation*, below).

–o *outfile* Name the output file *outfile* instead of **ada.make.out**.

–p Assume that all links in *library* and all units outside of *library* are valid (see *Multi-Library Operation*, below).

–r *unitpattern* Specify which units are to be brought up to date. *Unitpattern* may be the name of a single unit, or may contain one or more occurrences of the character *, which matches any sequence of zero or more characters in a unit name.

The character * must be escaped from the shell.

The default *unitpattern* is *, which selects all units in *library* and *files*.

–s *searchlibrary* Specify a library to be searched when a required unit is missing (no link to the unit exists in *library*, and the unit is not contained in *files*).

This option may be repeated to specify multiple libraries. *Ada.make* searches the libraries in the order they are given; the first correctly named unit is

selected.

−v Enable verbose mode, which writes the *ada.make* command being processed and the command sent to the subprocess *ada.make.bin* to standard error.

Source vs. Library Mode

Ada.make can work in source mode, to produce a correct compilation order for a set of source files, in library mode, to identify files that need to be recompiled, or in a combination of these two modes. When operating on an application whose units are contained in multiple libraries, *ada.make* produces commands to create the required links between libraries.

When *library* is not specified, *ada.make* operates in source mode on *files*. If *files* are not given, it uses files in the current directory whose names end with **.ad**?, where ? is any single character. When operating in source mode, the resulting compile commands do not specify a library. The commands can be executed, however, if the environment variable **ADAOPTS** contains a library name (see *ada*(1), Environment Variables).

If *library* is specified, it is read to determine which units are obsolete. For each unit *ada.make* examines, it uses the source file name that was recorded in the library when the unit was compiled.

When both *library* and *files* are specified, units contained in *files* have precedence over corresponding previously compiled units.

Multi-Library Operation

Ada.make provides three modes of operation for applications that are distributed among multiple libraries. These modes determine whether the validity of units outside *library* is checked, and whether commands will be emitted that would alter libraries other than *library*.

The default mode is to check all required units, regardless of what library they reside in. Any obsolete unit, however, is recompiled into *library*, rather than into the library in which the obsolete unit was found.

The **-p** ("partial") option directs *ada.make* to assume that all links in *library* and all units outside of *library* are valid. No commands to acquire links are emitted. Compile commands are only emitted for units contained in *files* and units that were previously compiled in *library*.

In the preceding two modes of operation, only *library* is changed -- no other libraries in the family are affected.

When the **-i** ("in situ") option is used, *ada.make* checks the validity of all required units and links regardless of what library they reside in. Any compilation or command to acquire a link is performed in the library that contains the obsolete unit or link. In this mode, modifications are not limited to *library*, but can occur in any library in the family.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.make*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part which holds the lock. An Ada family or library

locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.make* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

AUTHOR

Ada.make was developed by HP and Alsys.

FILES

$ADA_PATH/ada.make	*ada.make* command
$ADA_PATH/ada.make.bin	*ada.make* subprocess
ada.make.out	output file

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300),

Ada Tools Manual (Series 300).

NAME

ada.mkfam – make (or reinitialize) an Ada family

SYNOPSIS

ada.mkfam [**–f**] *familyname*

Remarks:

This command requires installation of optional ADA software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.mkfam creates the Ada family *familyname* in mode 777 (possibly altered by *umask*(1)). The Ada family created is used by the compiler to maintain groups of libraries which have visibility of each other.

Ada.mkfam requires write permission in the parent directory. *Ada.mkfam* only creates the last component of the specified path as the Ada family; all parent directories must already exist.

An Ada family is a group of Ada libraries that are visible to one another. Units in a library in a particular family can be dependent on units in another library in the same family and a program can be composed (bound) using units from any number of libraries in a given family.

Libraries that are members of different families are normally invisible to each other and cannot access each other's units. There is one exception to this rule; libraries belonging to the INSTALLATION family (a family provided with the Ada compilation system) are accessible to libraries in any other family.

If *familyname* exists, *ada.mkfam* can be used to reinitialize it (thus making it an empty family).

If the standard input is a terminal, *ada.mkfam* requests confirmation before reinitializing an existing Ada family. Confirmation is not requested if the **–f** option is given or if the standard input is not a terminal.

The following options are recognized:

–f Do not ask for confirmation if the family being created already exists; reinitialize the existing Ada family to the empty state.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.mkfam*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.mkfam* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.mkfam returns exit status **0** if the Ada family directory was successfully created. Otherwise a diagnostic is issued and a non-zero exit status is returned.

WARNINGS

An Ada family is a directory containing files for the use of the Ada compilation system. Users are strongly discouraged from placing any user files or directories in Ada family directories. User files/directories in Ada families are subject to damage by or might interfere with proper operation of the Ada compilation system. In particular, *ada.mkfam* does not preserve such user files when reinitializing an existing Ada family, and is unable to reinitialize an existing Ada family (effectively destroying it) if the existing Ada family directory contains any subdirectories.

AUTHOR

Ada.mkfam was developed by HP and Alsys.

FILES

$ADA_PATH/ada.mkfam	Ada make family command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.fmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.fmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

ada.mklib – make (or reinitialize) an Ada library

SYNOPSIS

ada.mklib [**–f**] [**–I** | **–P** | *familyname*] *libraryname*

DESCRIPTION

Ada.mklib creates the Ada library *libraryname* in mode 777 (possibly altered by *umask*(1)). The Ada library created is specified by name during subsequent compilations of Ada source, and is used by the compiler to maintain intermediate forms of Ada compilation units.

Ada.mklib requires write permission in the parent directory. *Ada.mklib* only creates the last component of the specified path as the Ada library; all parent directories must already exist.

The created library is made a member of an Ada family of libraries. By default, the created library is made a member of the PUBLIC family. Optionally, the created library can be made a member of any existing Ada family.

An Ada family is a group of Ada libraries that are visible to one another. Units in a library in a particular family can be dependent on units in another library in the same family and a program can be composed (bound) using units from any number of libraries in a given family.

Libraries that are members of different families are normally invisible to each other and cannot access each other's units. There is one exception to this rule; libraries belonging to the INSTALLATION family (a family provided with the Ada compilation system) are accessible to libraries in any other family.

If *libraryname* exists, *ada.mklib* can be used to reinitialize it (thus making it an empty library). If the library being reinitialized is a member of any family other than PUBLIC, the family must be specified when reinitializing the library.

If the standard input is a terminal, *ada.mklib* requests confirmation before reinitializing an existing Ada library. Confirmation is not requested if the **–f** option is given or if the standard input is not a terminal.

Only one of the options **-I**, **-P** and *familyname* can be specified. If none of these options are specified, the PUBLIC family is assumed.

The *familyname* argument may optionally be preceded by **-F**, e.g. **-F** *myfam*. This option is included only for backward compatibility with older versions of the ADS and should not be used when developing new applications.

The following options are recognized:

–f Do not ask for confirmation if the library being created already exists; reinitialize the existing Ada library to the empty state.

–I Make the newly created Ada library a member of the INSTALLATION family or identify the library to be reinitialized as a member of the INSTALLATION family. Ada libraries in the INSTALLATION family are visible to Ada libraries in all other families. Units in INSTALLATION family libraries can be accessed by units in any Ada library and an Ada program can be composed (bound) using units from libraries in any given family and units from libraries in the INSTALLATION family.

–P Make the newly created library a member of the PUBLIC family or identify the library to be reinitialized as a member of the PUBLIC family. This is the default family if no family is specified.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada

compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.mklib*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.mklib* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.mklib returns exit status **0** if the Ada library directory was successfully created. Otherwise a diagnostic is issued and a non-zero exit status is returned.

WARNINGS

An Ada library is a directory containing files for the use of the Ada compilation system. Users are strongly discouraged from placing any user files or directories in Ada library directories. User files/directories in Ada libraries are subject to damage by or might interfere with proper operation of the Ada compilation system. In particular, *ada.mklib* does not preserve such user files when reinitializing an existing Ada library, and is unable to reinitialize an existing Ada library (effectively destroying it) if the existing Ada library directory contains any subdirectories.

AUTHOR

Ada.mklib was developed by HP and Alsys.

FILES

$ADA_PATH/ada.mklib	Ada make library command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.mvfam – move (rename) an Ada family

SYNOPSIS

ada.mvfam [**-f**] *fromfam tofam*

Remarks:

This command requires installation of optional ADA software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

The Ada family *fromfam* is moved (renamed) to *tofam*. The contents of *tofam* are completely replaced with the contents of *fromfam*.

If the standard input is a terminal, *ada.mvfam* requests confirmation before replacing an existing *tofam*. Confirmation is not requested if the **–f** option is given or if the standard input is not a terminal.

Write permission is required in the parent directory of both *fromfam* and *tofam*.

The following option is recognized:

–f

Do not ask for confirmation if *tofam* already exists; remove *tofam* and rename *fromfam*.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.mvfam*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part which holds the lock. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.mvfam* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools which might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.mvfam returns exit status **0** if the operation was successful; otherwise, a diagnostic is issued and a non-zero exit status is returned.

If *fromfam* is not an Ada family or *tofam* exists and is not an Ada family, a diagnostic is issued and no action is taken.

WARNINGS

An Ada family is a directory containing files for the use of the Ada compilation system. Users are strongly discouraged from placing any user files or directories in Ada family directories. User files/directories in Ada families are subject to damage by or might interfere with proper operation of the Ada compilation system. In particular, *ada.mvfam* does not preserve such user files when renaming (moving) an Ada family, and is unable to overwrite an existing Ada family (effectively destroying it) if the existing Ada family directory contains any subdirectories.

AUTHOR

Ada.mvfam was developed by HP and Alsys.

FILES

$ADA_PATH/ada.mvfam	Ada move family command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.fmgr.hlp	Ada family manager help command data.
$ADA_PATH/ada.fmgr.btl	Ada family manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

ada.mvlib – move (rename) an Ada library

SYNOPSIS

ada.mvlib [**-f**] *fromlib tolib*

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

The Ada library *fromlib* is moved (renamed) to *tolib*. The contents of *tolib* are completely replaced with the contents of *fromlib*.

If *tolib* exists, it must be a member of the same family as *fromlib*.

If the standard input is a terminal, *ada.mvlib* requests confirmation before replacing an existing *tolib*. Confirmation is not requested if the **–f** option is given or if the standard input is not a terminal.

Write permission is required in the parent directory of both *fromlib* and *tolib*.

The following option is recognized:

–f Do not ask for confirmation if *tolib* already exists; remove *tolib* and rename *fromlib*.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.mvlib*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part which holds the lock. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.mvlib* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools which might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.mvlib returns exit status **0** if the operation was successful; otherwise, a diagnostic is issued and a non-zero exit status is returned.

If *fromlib* is not an Ada library or *tolib* exists and is not an Ada library, a diagnostic is issued and no action is taken.

WARNINGS

An Ada library is a directory containing files for the use of the Ada compilation system. Users are strongly discouraged from placing any user files or directories in Ada library directories. User files/directories in Ada libraries are subject to damage by or might interfere with proper operation of the Ada compilation system. In particular, *ada.mvlib* does not preserve such user files when renaming (moving) an Ada library, and is unable to overwrite an existing Ada library (effectively destroying it) if the existing Ada library directory contains any subdirectories.

AUTHOR

Ada.mvlib was developed by HP and Alsys.

FILES

$ADA_PATH/ada.mvlib	Ada move library command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.probe – Ada source level debugger and viewer

SYNOPSIS

ada.probe [*options*] *executable* [*args*]

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.probe is a source level debugger for Ada programs. It provides a controlled environment for their execution. Additionally, *ada.probe* provides powerful source viewing facilities for analyzing the static structure of large Ada programs.

Executable is an Ada executable program file having one or more units compiled with either the **-d** or **-D** option of the *ada*(1) command. In addition, the binding of the program must have included either of these options to create the associated *executable*.**cui** file. *Ada.probe* information is stored solely in the Ada program library and in the *executable*.**cui** file. The *executable* file is not modified in any way.

The following options are recognized:

-c	Dialog between the *executable* and the user starts at the current cursor position within the *ada.probe* screen. Only one of the options **-c** or **–m** may be selected.
-e *file*	Redirect standard error of the *executable* to the specified *file* name.
-i *file*	Redirect standard input of the *executable* from the specified *file* name.
-l *file*	Record any output written to the Message Window in the specified *file* name.
-m	Dialog between the *executable* and user is held in the Message Window of the *ada.probe* screen. Only one of the options **-c** or **–m** may be selected.
-o *file*	Redirect standard output of the *executable* to the specified *file* name.

Ada.probe invokes the *executable* with the specified *args* (if any). The initial screen layout consists of two windows:

- The Message Window displays output from *ada.probe*. In addition, dialog between the *executable* and user can be held in this window. (See Dialog.)
- The Menu Window displays the list of available commands. Each choice is either a command or a submenu that groups related commands.

Dialog

Dialog which is not redirected by the **-i** **-o** and **-e** options is controlled by either the **-c** or **-m** option. If neither option is specified, the default depends on whether or not *ada.probe* is running within the X *Window System*. Within the X *Window System*, dialog is held in the X window from which *ada.probe* was invoked; otherwise dialog is held in the Message Window of the *ada.probe* screen.

Commands

Commands are selected from the Menu Window or by entering user programmable key bindings. Command parameters, if required, are entered in pop-up windows. Within the permanent Menu Window the following commands are available:

Break	Submenu of breakpoint management commands.

Exec	Submenu of execution control commands.
Object	Submenu of object manipulation commands.
Progress	Submenu of progression facility commands.
Quit	Exit *ada.probe*.
Regs	Display register values.
Units	Submenu of unit manipulation commands.
Help	Obtain help on the various *ada.probe* commands.
Xtra	Submenu of advanced and miscellaneous facilities.

For the syntax and semantics of these commands and associated submenu commands, refer to the *Ada Tools Manual* or use the available on-line help facility.

If the file **.ada.proberc** exists in the user's home directory, its commands are executed before the first prompt is issued.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.probe*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part which holds the lock. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.probe* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

WARNINGS

Ada.probe is capable of limited debugging of optimized code. Certain commands may give misleading or unexpected results. For example, object values may be stored in registers; therefore the value displayed from memory may be incorrect. For this reason, the ability to examine or modify objects and expressions may be impaired. Dead code elimination, code motion, or inlining may affect single step execution or prevent breakpoints from being set on specific source lines.

A user cannot debug a program while the *executable* is already being debugged or executed. Attempting to start a debugging session on an *executable* that is already in use will cause a

WRITE_MEMORY error. If another user starts running an executable that is currently being debugged, there may be some interesting interactions.

AUTHOR

Ada.probe was developed by HP and Alsys.

FILES

$HOME/.ada.proberc	User start up commands.
$ADA_PATH/adahostcap	Ada host system capability checker.
$ADA_PATH/ada.probe	Ada debugger command.
$ADA_PATH/ada.probe.bin	Ada debugger subprocess.
$ADA_PATH/upi/ada.probe.kb	Ada debugger key binding data.
$ADA_PATH/upi/tcf	Ada debugger terminal configuration data.
$ADA_PATH/upi/*.btl	Ada debugger command tables.
$ADA_PATH/upi/*.hix	Ada debugger help command index.
$ADA_PATH/upi/*.hlp	Ada debugger help command data.
$ADA_PATH/upi/*.snf	Ada debugger font files.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300),

Ada Tools Manual (Series 300),

Using the X Window System.

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.protect – protect (unprotect) an Ada library

SYNOPSIS

ada.protect [**-r** | **-u**] *libraryname*

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.protect marks an Ada library as read-only or as able to be updated.

The following options are recognized:

–r Mark the Ada library read-only. This is also the default action if no option is specified.

–u Mark the Ada library as able to be updated.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.protect*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when a library is added, removed, or renamed; however, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part which holds the lock. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.protect* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.protect returns exit status **0** if the Ada library was successfully protected (unprotected); otherwise a diagnostic is issued and a non-zero exit status is returned.

AUTHOR

Ada.protect was developed by HP and Alsys.

FILES

$ADA_PATH/ada.protect	Ada protect library command.
$ADA_PATH/adacli	Ada command language interpreter.

$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.rmfam – remove an Ada family

SYNOPSIS

ada.rmfam [**-f**] *familyname*

Remarks:

This command requires installation of optional ADA software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.rmfam removes the Ada family (directory) *familyname*.

If the standard input is a terminal, *ada.rmfam* requests confirmation before removing an Ada family. Confirmation is not requested if the **–f** option is given or if the standard input is not a terminal.

Removal of an Ada family requires write permission in its parent directory and recursively in all its subdirectories, but neither read nor write permission on the files within it.

The following option is recognized:

–f Do not ask for confirmation before removing the Ada family.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.rmfam*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.rmfam* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you may remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.rmfam returns exit status **0** if the Ada family was successfully removed; otherwise, a diagnostic is issued and a non-zero exit status is returned.

WARNINGS

An Ada family is a directory containing files used by the Ada compilation system. Users are strongly discouraged from placing any user files or directories in Ada family directories. User files/directories in Ada families are subject to damage by or might interfere with proper

operation of the Ada compilation system. In particular, *ada.rmfam* attempts to remove all such files and subdirectories.

Lack of write permission to the parent directory or subdirectories causes *ada.rmfam* to fail to correctly remove the Ada family and leads to partial destruction of the structure under the Ada family.

Removing an Ada family when your current working directory is the Ada family directory or a subdirectory of the Ada family directory might lead to an incompletely removed Ada family.

AUTHOR

Ada.rmfam was developed by HP and Alsys.

FILES

$ADA_PATH/ada.rmfam	Ada remove family command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.fmgr.hlp	Ada family manager help command data.
$ADA_PATH/ada.fmgr.btl	Ada family manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1), ada.xref(1).

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

ada.rmlib – remove an Ada library

SYNOPSIS

ada.rmlib [**-f**] *libraryname*

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.rmlib removes the Ada library (directory) *libraryname*.

If the standard input is a terminal, *ada.rmlib* requests confirmation before removing an Ada library. Confirmation is not requested if the **–f** option is given or if the standard input is not a terminal.

Removal of an Ada library requires write permission in its parent directory and recursively in all its subdirectories, but neither read nor write permission on the files within it.

The following option is recognized:

–f Do not ask for confirmation before removing the Ada library.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.rmlib*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the locking agent. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only purposes.

If *ada.rmlib* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but is should not be locked, you may remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

DIAGNOSTICS

Ada.rmlib returns exit status **0** if the Ada library was successfully removed; otherwise, a diagnostic is issued and a non-zero exit status is returned.

WARNINGS

An Ada library is a directory containing files used by the Ada compilation system. Users are strongly discouraged from placing any user files or directories in Ada library directories. User files/directories in Ada libraries are subject to damage by or might interfere with proper

operation of the Ada compilation system. In particular, *ada.rmlib* attempts to remove all such files and subdirectories.

Lack of write permission to the parent directory or subdirectories causes *ada.rmlib* to fail to correctly remove the Ada library and leads to partial destruction of the structure under the Ada library.

Removing an Ada library when your current working directory is the Ada library directory or a subdirectory of the Ada library directory might lead to an incompletely removed Ada library.

AUTHOR

Ada.rmlib was developed by HP and Alsys.

FILES

$ADA_PATH/ada.rmlib	Ada remove library command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada component manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.umgr(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.umgr – examine and modify an Ada library

SYNOPSIS

ada.umgr [**-r** | **-u**] *libraryname*

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.umgr, the Ada unit manager, can be used to examine and modify the contents of an Ada library.

The following options are recognized:

–r Open the Ada library in read-only mode. The Ada library can only be examined, but not modified. This is the default mode.

–u Open the Ada library in update mode. The ERASE and ACQUIRE commands and the RESET option of the CHECK command require that the Ada library be opened in update mode.

Ada.umgr is an interactive tool which, when executed, prompts with "Unit_Manager". Commands are then entered to examine or modify the contents of the Ada library.

For the syntax and semantics of the commands, refer to the *Ada User's Guide*, or use the available online help facility in the unit manager.

The following commands can be typed in response to the "Unit_Manager" prompt:

Acquire	Make links in the current Ada library to units in another Ada library (in the same family or INSTALLATION family).
Check	Verify that links in the current Ada library to units in other Ada libraries are valid (that is, up to date).
Default	Set default values for commands.
Erase	Erase units in the current Ada library.
Global	Set global characteristics for commands.
Help	Obtain help on the various *ada.umgr* commands.
Invoke	Read *ada.umgr* commands from a file.
Join	Compute a program closure (that is, determine whether a bind can be performed).
List	List units in the current Ada library.
Other	Change the current Ada library.
Quit	Leave *ada.umgr*.
System	Execute a shell command.
Users	List dependencies of a unit in the current Ada library.
View	Obtain general information about the current Ada library.

The *help* command provides online help for all the commands.

If the file **.ada.umgrrc** exists in the user's home directory, its commands are executed before the first prompt is issued.

If the file **.ada.umgrdf** exists in the user's home directory, its commands are executed after those in **.ada.umgrrc** (if present) and before the first prompt is issued. The commands in this file are DEFAULT commands saved automatically by using the DEFAULTS => KEEP parameter of the QUIT command.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.umgr*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part which holds the lock. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system which only desires to read from the Ada family or library.

If *ada.umgr* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you may remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools which might find the contents invalid or might damage the Ada family or Ada library.

AUTHOR

Ada.umgr was developed by HP and Alsys.

FILES

$HOME/.ada.umgrrc	User start up commands.
$HOME/.ada.umgrdf	User start up defaults.
$ADA_PATH/ada.umgr	Ada interactive unit manager command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.umgr.hlp	Ada unit manager help command data.
$ADA_PATH/ada.umgr.btl	Ada unit manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.unlock(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.unlock – unlock an Ada library

SYNOPSIS

ada.unlock libraryname

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.unlock removes all locks on an Ada library.

It is intended that *ada.unlock* be used to remove inappropriate locks. Such locks might be left by abnormally terminated components of the Ada compilation system or by system crashes.

Unlocking an Ada library requires write permission in its parent family's directory.

Extreme care must be used when unlocking an Ada library. If any component of the Ada compilation system is reading the Ada library, removing locks may cause that component to encounter internal inconsistencies and fail. In addition, if any component of the Ada compilation system is updating the Ada library, removing locks may cause that component to corrupt the Ada library and fail.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.unlock*) can be used (see *ada*(1), Environment Variables).

DIAGNOSTICS

Ada.unlock returns exit status 0 if the Ada library was successfully unlocked. Otherwise a diagnostic is issued and a non-zero exit status is returned.

AUTHOR

Ada.unlock was developed by HP and Alsys.

FILES

$ADA_PATH/ada.unlock	Ada unlock library command.
$ADA_PATH/adacli	Ada command language interpreter.
$ADA_PATH/ada.mngrs.bin	Ada components manager subprocess.
$ADA_PATH/ada.lmgr.hlp	Ada library manager help command data.
$ADA_PATH/ada.lmgr.btl	Ada library manager command tables.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.xref(1),

Ada User's Guide (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

ada.xref – Ada cross-referencer

SYNOPSIS

ada.xref [**-l**] *file*

Remarks:

This command requires installation of optional Ada software (not included with the standard HP-UX operating system) before it can be used.

DESCRIPTION

Ada.xref produces a list of cross references for each name encountered in the compilation unit(s) submitted to it. *Ada.xref* operates in either source or library mode, as defined below:

source mode — *Ada.xref* gathers information directly from the specified Ada source *file*. In this mode, *ada.xref* requires the source *file* suffix to be **.ad***?*, where *?* is any single alphanumeric character. The advantage of this mode is that the compilation unit(s) need not be compiled.

library mode — *Ada.xref* retrieves the information which has been stored in one or more Ada program libraries at compile time. In this mode, *file* contains a list of compilation units along with the library name(s) containing the actual (as opposed to linked) compilation units. The compilation units specified must have been compiled with the **-k** option of the *ada(1)* command. The advantage of this mode is that *ada.xref* can identify Ada objects precisely, resolve overloading, understand deferred constants, and extend the cross-references to parent and "withed" units not directly submitted.

The following option is recognized:

–l Indicate library mode cross-referencing to be performed (default is source mode).

In library mode, any number of libraries can be specified, but all must belong to the same family. The library names are specified in the *file* using the following syntax:

use *library_name*

and the unit names are specified in the *file* using the following syntax:

unit_name [**specification** | **body** | **all**]

where the optional unit parameter defines whether to cross-reference the specification, body or both the specification and body of the specified unit. By default, only the body of the specified *unit_name* is cross-referenced.

Environment Variables

The environment variable **ADA_PATH** is associated with all components of the Ada compilation system. It must be set properly and exported before any component of the Ada compilation system (including *ada.xref*) can be used (see *ada*(1), Environment Variables).

Locks

To ensure the integrity of their internal data structures, Ada libraries and families are locked for the duration of operations that are performed on them. Normally Ada families are locked only briefly when libraries within them are manipulated. However, sometimes multiple Ada libraries must be locked for longer periods during a single operation. If more than one library must be locked for a single operation, at most one library is locked for updating and all other locked libraries are locked for read-only.

An Ada family or library locked for updating cannot be accessed in any way by any part of the Ada compilation system except by the part which holds the lock. An Ada family or library locked for reading can be accessed by any part of the Ada compilation system for read-only

purposes.

If *ada.xref* cannot obtain a lock after a suitable number of retries, it displays an informational message and terminates.

Under some circumstances, an Ada family or Ada library might be locked, but the locking program(s) might have terminated (for example, due to system crash or network failure). If you determine that the Ada family or Ada library is locked but should not be locked, you can remove the lock.

Use *ada.unlock*(1) to unlock an Ada library and *ada.funlock*(1) to unlock an Ada family. However, unlocking should be done with care. If an Ada family or Ada library is actually locked by a tool, unlocking it permits access by other tools that might find the contents invalid or might damage the Ada family or Ada library.

AUTHOR

Ada.xref was developed by HP and Alsys.

FILES

$ADA_PATH/ada.xref	Ada cross-referencer command.
$ADA_PATH/ada.xref.bin	Ada cross-referencer subprocess.

SEE ALSO

ada(1), ada.fmgr(1), ada.format(1), ada.funlock(1), ada.lmgr(1), ada.lsfam(1), ada.lslib(1), ada.make(1), ada.mkfam(1), ada.mklib(1), ada.mvfam(1), ada.mvlib(1), ada.probe(1), ada.protect(1), ada.rmfam(1), ada.rmlib(1), ada.umgr(1), ada.unlock(1).

Ada User's Guide (Series 300), HP Part No. 98860-90000.

Ada Tools Manual (Series 300).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported within file names.

NAME

adb – absolute debugger

SYNOPSIS

adb [**−w**] [**−I***dir*] [*objfil* [*corfil*]]

DESCRIPTION

Adb is a general-purpose debugging program sensitive to the underlying architecture of the processor on which it runs. It can be used to examine files and provide a controlled environment for the execution of HP-UX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not, the symbolic features of *adb* cannot be used, although the file can still be examined. The default for *objfil* is **a.out**. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core**.

Requests to *adb* are read from standard input and *adb* responds to standard output. If the **−w** flag is present, *objfil* is created (if necessary) and opened for reading and writing, to be modified using *adb*. The **−I** option specifies a directory where files read with $< or $<< (see below) are sought; the default is **/usr/lib/adb**. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

Requests to *adb* follow the form:

[*address*] [, *count*] [*command*] [;]

If *address* is present, *dot* is set to *address*. Initially *dot* is set to **0**. For most commands, *count* specifies the number of times the command will be executed. The default *count* is **1**. *Address* and *count* are expressions.

The interpretation of an address depends on the context in which it is used. If a subprocess is being debugged, addresses are interpreted in the address space of the subprocess. (For further details of address mapping see Addresses below.)

Expressions

.	The value of *dot*.
+	The value of *dot* increased by the current increment.
^	The value of *dot* decreased by the current decrement.
"	The last *address* typed.
integer	A number. The prefixes **0o** and **0O** (zero oh) force interpretation in octal radix; the prefixes **0t**, **0T**, **0d** and **0D** force interpretation in decimal radix; the prefixes **0x** and **0X** force interpretation in hexadecimal radix. Thus **0o20** = **0t16** = **0x10** = sixteen. If no prefix appears, the *default radix* is used; see the $d command. The radix is initialized to the base used in the assembly language for the processor involved. Note that a hexadecimal number whose most significant digit would otherwise be an alphabetic character must have a **0x** (or **0X**) prefix (or a leading zero if the default radix is hexadecimal).
integer.fraction	A 32-bit floating point number.
'cccc'	The ASCII value of up to 4 characters. A backslash (\) can be used to escape a single quote (').
< *name*	*Name* can have the value of either a variable or a register. *Adb* maintains a number of variables named by single letters or digits; see Variables below. If *name* is a register, the value of the register is obtained from the system header in *corfil* (before the subprocess is initiated) or from the user area of the subprocess. Register names are implementation dependent; see the $r command.

symbol A *symbol* is a sequence of uppercase or lowercase letters, underscores or digits, not starting with a digit. A backslash (\) can be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial underscore (_) is prefixed to *symbol*, if needed.

_ *symbol* If the compiler prefixes _ to an external symbol, it may be necessary to cite this name to distinguish it from a symbol generated in assembly language.

(*exp*) The value of the expression *exp*.

The following are monadic operators:

*****exp* The contents of the location addressed by *exp* in *corfil*.

@ *exp* The contents of the location addressed by *exp* in *objfil*.

−*exp* Integer negation.

~*exp* Bitwise complement.

The following dyadic operators are left associative and are less binding than monadic operators:

e1+*e2* Integer addition.

e1−*e2* Integer subtraction.

*e1***e2* Integer multiplication.

e1%*e2* Integer division.

e1&*e2* Bitwise conjunction.

e1|*e2* Bitwise disjunction.

e1#*e2* *E1* rounded up to the next multiple of *e2*.

Commands

Most commands consist of an action character followed by a modifier or list of modifiers. The following action characters can take format specifiers. (The action characters **?** and **/** can be followed by *; see Addresses for further details.)

?*f* Locations starting at *address* in *objfil* are printed according to the format *f*. *Dot* is incremented by the sum of the increments for each format letter. If a subprocess has been initiated, *address* references a location in the address space of the subprocess instead of *objfil*.

/*f* Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is increased like **?**. If a subprocess has been initiated, *address* refers to a location in the address space of the subprocess instead of *corfil*.

=*f* The value of *address* is printed in the styles indicated by the format *f*. (For **i** format **?** is printed for the parts of the instruction that refer to subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character can be preceded by an integer that indicates how many times the format is repeated. While stepping through a format, *dot* is increased by the amount given for each format character. If no format is given then the last format is used.

The following format characters are available:

o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by **0**.

O 4 Print 4 bytes in octal.

q 2 Print 2 bytes in signed octal.

Q 4 Print 4 bytes in signed octal.

d 2 Print 2 bytes in decimal.

D 4 Print 4 bytes in decimal.

x 2 Print 2 bytes in hexadecimal.

X 4 Print 4 bytes in hexadecimal.

u 2 Print 2 bytes as an unsigned decimal number.

U 4 Print 4 bytes as an unsigned decimal number.

f 4 Print the 32 bit value as a floating point number.

F 8 Print double floating point.

b 1 Print the addressed byte in hexadecimal.

B 1 Print the addressed byte in octal.

c 1 Print the addressed character. (The sign bit is ignored.)

C 1 Print the addressed character using the following escape convention. First, the sign bit is discarded, then character values **000** to **040** are printed as **@** followed by the corresponding character in the range **0100** to **0140**. The character **@** is printed as **@@**.

s *n* Print the addressed characters until a zero character is reached.

S *n* Print a string using the **@** escape convention. The value *n* is the length of the string including its zero terminator.

Y 4 Print 4 bytes in date format (see *ctime*(3C)).

i *n* Print as machine instructions. The value of *n* is the number of bytes occupied by the instruction.

a 0 Print the value of *dot* in symbolic form.

p *n* Print the addressed value in symbolic form. The value of *n* is a machine dependent constant.

t 0 When preceded by an integer, moves to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

r 0 Print a space.

n 0 Print a new-line.

"..." 0 Print the enclosed string.

^ *Dot* is decreased by the current increment. Nothing is printed.

\+ *Dot* is increased by 1. Nothing is printed.

− *Dot* is decreased by 1. Nothing is printed.

new-line Repeat the previous command with a *count* of 1. New-line can also be used to repeat a **:s** or **:c** command; however, any arguments to the previous command are lost.

[**?**/]**l** *value mask* Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used, then *adb* looks to match 4 bytes at a time instead of 2. If no match is found, *dot* is left unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted **−1** is used.

[?/]w *value ...* Write the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m *b1 e1 f1*[**?/**]
Record new values for (*b1, e1, f1*). If less than three expressions are given, the remaining map parameters are left unchanged. If the **?** or **/** is followed by *, the second segment (*b2,e2,f2*) of the mapping is changed. If the list is terminated by **?** or **/**, the file (*objfil* or *corfil*, respectively) is used for subsequent requests. (For example, **/m?** will cause **/** to refer to *objfil.)*

>*name* Assign *dot* to the variable or register named.

! Call a shell to read the remainder of the line following **!**.

The following **$** commands take the form **$***modifier*:

$<*f* Read commands from the file *f*. If this command is executed in a file, further commands in the file are not seen. If a *count* is given, and is zero, the command will be ignored. The value of the count will be placed in variable 9 before the first command in *f* is executed.

$<<*f* Similar to **$<** except it can be used in a file of commands without causing the file to be closed. Variable 9 is saved when the command executes and is restored when it completes. Only five **$<<** files can be open at once.

$>*f* Send output to the file *f*, which is created if it does not already exist.

$r Print the general registers and the instruction addressed by the process counter. *Dot* is set to the process counter contents.

$f Print the floating-point registers.

$b Print all breakpoints and their associated counts and commands.

$c C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of the normal stack frame pointer). If *count* is given then only the first *count* frames are printed.

$e The names and values of external variables are printed.

$w Set the page width for output to *address* (default 80).

$s Set the limit for symbol matches to *address*. The default is system dependent.

$o The default for all integers input is octal.

$d Set the default radix to *address* and report the new value. Note that *address* is interpreted in the (old) current radix. Thus **10$d** never changes the default radix. To make decimal the default radix, use **0t10$d**.

$x The default for all integers input is hexadecimal.

$q Exit from *adb*.

$v Print all non-zero variables in the current radix.

$m Print the address map.

$new-line print the process id and register values.

The available : commands manage subprocesses, and take the form *:modifier:*

:b*c* Set breakpoint at *address*. The breakpoint is executed *count*−1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero, the breakpoint causes a stop.

:d Delete breakpoint at *address*. **:d*** will delete all breakpoints.

:r Run *objfil* as a subprocess. If *address* is given explicitly, the program is entered at this point; otherwise the program is entered at its standard entry point. The value *count* specifies how many breakpoints are ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on when entering the subprocess.

:e Set up a subprocess as in **:r**; no instructions are executed.

:c*s* Continue the subprocess with signal *s* (see *signal*(5)). If *address* is given, the subprocess continues at this address. If no signal is specified, the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **:r**.

:s*s* As for **c** except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for **:r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.

:S*s* As for **:c** except that a temporary breakpoint is set at the next instruction. Useful for stepping across subroutines.

:x *a* [*b*]... Execute subroutine *a* with parameters [*b*]...

:k Terminate the current subprocess, if any.

Variables

Adb provides named and numbered variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows:

0 The last value printed.

1 The last offset part of an instruction source.

2 The previous value of variable 1.

9 The count on the last **$<** command.

On entry the following named variables are set from the system header in the *corfil*. If *corfil* does not appear to be a **core** file, these values are set from *objfil*.

b The base address of the data segment.

d The data segment size.

e The entry point.

m The "magic" number as defined in <**magic.h**>.

s The stack segment size.

t The text segment size.

Addresses

The file address associated with a written address is determined by a mapping described below; see **$m**. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*).

The *file address* corresponding to a written *address* is calculated as follows:

If

$$b1 \leq address < e1, \textit{ file address} = address + f1 - b1,$$

otherwise, if

$$b2 \leq address < e2, \textit{ file address} = address + f2 - b2,$$

otherwise, the requested *address* is not valid. In some cases (for example, for programs with separated I and D space) the two segments for a file may overlap. If **?** or **/** is followed by *, only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected, *adb* sets to *b1* **0**, *e1* t to the maximum file size, and *f1* to **0**; in this way the entire file can be examined with no address translation.

Adb keeps all appropriate values as signed 32-bit integers, so that it can be used on large files.

RETURN VALUE

Adb comments about inaccessible files, syntax errors, abnormal termination of commands, etc. It echoes "adb" when there is no current command or format. Exit status is **0**, unless the last command failed or returned non-zero status.

DEPENDENCIES

Series 300

The **-I** option is not currently supported.

The **I** format prints machine instructions, like **i**, except that immediate constants are printed in decimal.

The command **$n** is provided to set the number of significant digits for floating-point dumps to *address*.

Variable **9** is not updated for the **$<** command, and the **$<<** command is not supported.

The following variables are also supported:

f If this is set to a non-zero value, any sequence of machine instructions which effectively constitute a single floating point accelerator instruction will be treated as a single instruction for machine level single-stepping and display.

r If **f** is set to a non-zero value, **r** indicates which address register is used in floating point accelerator instruction sequences. A **0** corresponds to register **a0**, **1** to **a1**, etc. The default value is **2**.

Series 800

The **$f** command will print floating point registers as 32-bit single precision and **$F** will print these registers as 64-bit doubles.

$R will print all registers available to *adb* users.

The **:x** and **:S** commands are not currently supported.

The following options are also supported:

−k Allows virtual-to-physical address translation, useful for kernel debugging. In this case, *core* should be an HP-UX crash dump or **/dev/mem**.

When *adb* is invoked with this option, it sets up the context of the currently running process using space registers four through seven. A user specified address is dereferenced by combining it with the appropriate space register depending on the quadrant in which the 32-bit address lies. The **$p** command is provided to change the current context. The *address* argument is

the address of the process structure corresponding to the desired context.

When the current radix is not (decimal) ten, the **-k** option allows *adb* to support the notion of long pointers or addresses in the form *space.offset*. Once a space is specified, all subsequent addresses are dereferenced using that space until the user enters another long address. If a space equal to (hexadecimal) 0xffffffff is used, *adb* reverts to the previous context and uses space registers four through seven to dereference 32-bit addresses.

-P*pid* Causes *adb* to adopt process *pid* as a "traced" process (see *ptrace*(2)). This option is helpful for debugging processes that were not originally run under the control of *adb*.

Adb can be used to inspect relocatable object files; it reads the symbol table and sets up the appropriate mappings for text and data. Note that relocatable object files do not necessarily contain an exact image of the initialized data; however, if this is the case, the data mapping is not set.

AUTHOR

Adb was developed by AT&T and HP.

FILES

a.out
core
/dev/mem
/dev/swap

SEE ALSO

ptrace(2), ctime(3C), a.out(4), core(4), signal(5).

The ADB Debugger, tutorial in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

adjust – simple text formatter

SYNOPSIS

adjust [**–bcjr**] [**–m** *column*] [**–t** *tabsize*] [*files...*]

DESCRIPTION

This command is a simple text formatter for filling, centering, left and right justifying, or right-justifying text paragraphs, designed for interactive use. It reads the concatenation of input files (or standard input if none are given) and produces on standard output a formatted version of its input, with each paragraph formatted separately. If "–" is given as an input filename, *adjust* reads standard input at that point. (You can use "--" as an argument to separate "-" from options.)

Adjust reads text from input lines as a series of words separated by blanks, tabs, or newlines. Text lines are grouped into paragraphs separated by blank lines. By default, text is carried over to the output subject only to simple filling (see below), with a right margin of 72, and leading blanks converted to tabs where possible.

Options are:

–b Do not convert leading blanks to tabs on output. Thus there are no tabs in the output.

–c Center text on each line. Lines are pre- and post-processed, but no filling is done.

–j Justify text. After filling, insert blanks in each line, except the last line of each paragraph, as needed to right-justify it, while keeping the justified left margin.

–r After filling text, adjust the indentation of each line for a smooth right margin (ragged left margin).

–m*column*

Set the right fill margin to the given column number, instead of 72. Text is filled, and optionally right justified, so that no output line extends beyond this column (if possible). If **–m0** is given, the current right margin of the first line of each paragraph is used for that and all subsequent lines in the paragraph.

By default, text is centered on column 40. With **–c**, the **–m** option sets the middle column of the centering "window", but **–m0** auto-sets the right side as before (which then determines the center of the "window").

–t*tabsize*

Set the tab size to other than the default (eight columns).

Only one of the **–c**, **–j**, and **–r** options is allowed in a single command line.

Details

Before doing anything else to a line of input text, *adjust* first handles backspaces, rubbing out preceding characters in the usual way. Next it ignores all non-printable characters except tab. It then expands all tabs to blanks.

For simple text filling, the first word of the first line of each paragraph is indented the same amount as in the input line. Each word is then carried to the output followed by one blank. "Words" ending in <terminal>[<quote>][<close>] are followed by two blanks, where <terminal> is any of ".:?!", <quote> is a single or double quote, and <close> is any of ")]}", e.g.:

end. of? sentence.' sorts!" of.) words?"]

The program starts a new output line whenever adding a word (other than the first one) to the current line would pass the right margin.

Adjust understands indented first lines of paragraphs (like this one) when filling. The second and subsequent lines of each paragraph are indented the same amount as the second line of the paragraph in the input, if there is a second line, else the same as the first line.

* *Adjust* also has a rudimentary understanding of tagged paragraphs (like this one) when filling. If the second line of a paragraph is indented more than the first, and the first line has a word beginning at the same indentation as the second line, the column positions of the tag words (prior to that one) are "frozen" (not altered).

Tag words are passed through without change of column position, even if they extend beyond the right margin. The rest of the line is filled or right-justified from the position of the first non-tag word.

When **−j** is given, *adjust* uses an intelligent algorithm to insert blanks in output lines where they are most needed, until the lines extend to the right margin. First, all one-blank word separators are examined. One blank is added first to those separators with the most total letters in the words on both sides. If all one-blank separators are increased to two blanks, and more blanks must be inserted, it repeats the algorithm, this time with two-blank separators, and so on.

Output line indentation is held to one less than the right margin. If a single word is larger than the line size (right margin minus indentation), that word appears on a line by itself, properly indented, and extends beyond the right margin. However, if **−r** is used, such words are still right-justified, if possible.

DIAGNOSTICS

Adjust complains to standard error and later returns a non-zero value if any input file cannot be opened (it skips the file). It does the same (but quits immediately) if the argument of **−m** or **−t** is out of range, or if the program is improperly invoked.

Input lines longer than BUFSIZ are silently split (before tab expansion) or truncated (afterwards). Lines too wide to center begin in column 1 (no leading blanks).

WARNINGS

This program is designed to be simple and fast. It does not recognize backslash to escape white space or anything else. It does not recognize tagged paragraphs where the tag is on a line by itself. It knows that lines end in newline or null, and how to deal with tabs and backspaces, but it does not do anything special with other characters like form feed (they are just ignored). For complex operations, the standard text processors are likely to be more appropriate.

This program could be implemented instead as a set of independent programs, fill, center, and justify (with **−r** option). However, this would be much less efficient in actual use, especially given the program's special knowledge of tagged paragraphs and last lines of paragraphs.

These options were considered but not added, because the creeping featurism had to stop somewhere, before this program became another *nroff*(1):

−h Hyphenate. Allows the program to break and join words at existing hyphens (only). Words are broken across lines, at single hyphens surrounded by non-whitespace characters. Likewise, a word ending in a single hyphen, followed by whitespace, followed by a non-hyphen character, is joined to the next word without whitespace if needed.

−n Nofill. Only allowed with **−j** or **−r**, it inhibits filling, i.e. words are not moved from one line to another. Thus existing text can be left/right or right justified without being otherwise modified. (Note that **−n** is always in effect for **−c**, centering.)

EXAMPLES

This command is useful for filtering text while in *vi*(1). For example,

```
!}adjust
```

reformats the rest of the current paragraph (from the current line down), evening the lines.

You can give the **vi** command:

```
:map ^X {!}adjust -j^V^M
```

(where "^" denotes control characters) to set up a useful "finger macro". Then typing ^X will reformat the entire current paragraph.

Note that **adjust -m1** is a simple way to break text into separate words, without white space, except for tagged paragraphs tags.

AUTHOR

Adjust was developed by HP.

SEE ALSO

nroff(1).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines what are valid space and printable characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *adjust* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

NAME

admin – create and administer SCCS files

SYNOPSIS

admin [−**n**] [−**i**[*name*]] [−**r***rel*] [−**t**[*name*]] [−**f***flag*[*flag-val*]] [−**d***flag*[*flag-val*]] [−**a***login*] [−**e***login*] [−**m**[*mrlist*]] [−**y**[*comment*]] [−**h**] [−**z**] *file* ...

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments, which begin with −, and named files (note that SCCS file names must begin with the characters **s.**). If a named file does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

−**n** This keyletter indicates that a new SCCS file is to be created.

−**i**[*name*] The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see −**r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created with an empty initial delta. Only one SCCS file may be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no −**i** keyletter). Note that the −**i** keyletter implies the −**n** keyletter.

−**r***rel* The *rel*ease into which the initial delta is inserted. This keyletter may be used only if the −**i** keyletter is also used. If the −**r** keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

−**t**[*name*] The *name* of a file from which descriptive text for the SCCS file is to be taken. If the −**t** keyletter is used and *admin* is creating a new SCCS file (the −**n** and/or −**i** keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a −**t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a −**t** keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

−**f***flag* This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single *admin* command line. The allowable *flag*s and their values are:

b Allows use of the −**b** keyletter on a *get*(1) command to create branch deltas.

c*ceil* The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a *get*(1) command for editing. The default

value for an unspecified **c** flag is 9999.

f*floor* The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get*(1) command for editing. The default value for an unspecified **f** flag is 1.

d*SID* The default delta number (SID) to be used by a *get*(1) command.

i*str* Causes the "No id keywords (cm7)" message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string, however the string must contain a keyword, and no embedded newlines.

j Allows concurrent *get*(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* A *list* of releases to which deltas can no longer be made (**get** **−e** against one of these "locked" releases fails). The *list* has the following syntax:
<list> ::= <range> | <list> , <range>
<range> ::= *RELEASE NUMBER* | **a**
The character **a** in the *list* is equivalent to specifying *all releases* for the named SCCS file. Omitting any list is equivalent to **a**.

n Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as 'anchor points' so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

q*text* User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get*(1).

m*mod* *Module* name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get*(1). If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

t*type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get*(1).

v[*pgm*] Causes *delta*(1) to prompt for Modification Request (*MR*) numbers as the reason for creating a delta. The optional value specifies the name of an *MR* number validity checking program (see *delta*(1)). (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null).

−d*flag* Causes removal (deletion) of the specified *flag* from an SCCS file. The **−d** keyletter may be specified only when processing existing SCCS files. Several **−d** keyletters may be supplied on a single *admin* command. See the **-f** keyletter for allowable *flag* names.

l*list* A *list* of releases to be "unlocked". See the **−f** keyletter for a description of the **l** flag and the syntax of a *list*.

−a*login* A *login* name, or numerical HP-UX group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several **a** keyletters may be used on a single *admin* command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a **!** they are to be denied permission to make deltas.

−e*login* A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **e** keyletters may be used on a single *admin* command line.

−y[*comment*] The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the −**y** keyletter results in a default comment line being inserted in the form:

date and time created *YY/MM/DD HH:MM:SS* by *login*

The −**y** keyletter is valid only if the −**i** and/or −**n** keyletters are specified (i.e., a new SCCS file is being created).

−**m**[*mrlist*] The list of Modification Requests (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The **v** flag must be set and the *MR* numbers are validated if the **v** flag has a value (the name of an *MR* number validation program). Diagnostics will occur if the **v** flag is not set or *MR* validation fails.

−**h** Causes *admin* to check the structure of the SCCS file (see *sccsfile*(4)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

−**z** The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see −**h**, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

Access Control Lists (ACLs)

Do not add optional ACL entries to SCCS files. SCCS removes them, which might cause unexpected and undesirable access modes.

DIAGNOSTICS

Use *help*(1) for explanations.

FILES

The last component of all SCCS file names must be of the form **s.***file-name*. New SCCS files are given mode 444 (see *chmod*(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called **x.***file-name*, (see *get*(1)), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures

that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed*(1). *Care must be taken!* The edited file should *always* be processed by an **admin** **−h** to check for corruption followed by an **admin** **−z** to generate a proper check-sum. Another **admin** **−h** is recommended to ensure the SCCS file is valid.

Admin also makes use of a transient lock file (called **z.***file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get*(1) for further information.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4), acl(5).

SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *admin* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte file names are *not* supported.

STANDARDS CONFORMANCE

admin: SVID2, XPG2, XPG3

NAME

ar – maintain portable archives and libraries

SYNOPSIS

ar *key* [*posname*] *afile* [*name*] ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor (see *ld*(1). It can be used, however, for any similar purpose. The magic string and file headers used by *ar* consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

Individual files are inserted without conversion into the archive file. When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar*(4). The archive symbol table (described in *ar*(4)) is used by the link editor (*ld*(1)) to search repeatedly and efficiently through libraries of object files. An archive symbol table is created and maintained by *ar* only when the archive contains at least one object file. The archive symbol table is in a specially named file that is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *ar*(1) command is used to create or update the contents of an archive, the symbol table is rebuilt. The **s** modifier described below forces the symbol table to be rebuilt.

Key must be present, and consists of an optional –, followed by one operation character from the set **drqtpmx**, optionally concatenated with one or more modifier characters from the set **vuaibcls**. *Afile* is the archive file. Constituent files in the archive file are specified by *name* arguments.

The **TMPDIR** environment variable can be set to specify a directory for temporary files (see *tmpnam*(3S)). **TMPDIR** overrides the default directory **/usr/tmp** as well as any use of the **l** modifier.

The following *key* operation characters are recognized:

d Delete the named files from the archive file.

r Replace the named files, or add a new file to the archive. If the modifier **u** is used with the operation character **r**, only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. In the absence of a positioning character, new files are placed at the end. *Ar* creates *afile* if it does not already exist. If no files are specified by *name* arguments, *ar* creates an empty archive file containing only the archive header (see *ar*(4)).

q Quickly append the named files to the end of the archive file. Positioning characters are invalid. The operation does not check to determine whether the added members are already in the archive. This is useful only to avoid quadratic behavior when creating a large archive piece-by-piece. *Ar* creates *afile* if it does not already exist.

t Print a table of contents of the archive file. If no names are given, all files in the archive are described. If names are given, information about only those files appears.

p Print the named files in the archive.

m Move the named files. By default, the files are moved to the end of the archive. If a positioning character is present, the *posname* argument must be present and, as in **r**, *posname* specifies where the files are to be moved. Note that, when used with a positioning character, the files are moved in the same

order that they currently appear in the archive, *not* in the order specified on the command line. See EXAMPLES.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter (that is, delete entries from) the archive file.

The following optional modifiers are recognized:

s Force the regeneration of the archive symbol table even if *ar*(1) is not invoked with an operation that modifies the archive contents. This modifier is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

v Verbose. Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, **v** gives a long listing of all information about the files. When used with the **d**, **m**, **p**, **q**, and **x** operations, the verbose modifier causes *ar* to print each *key* operation character and file name associated with that operation. For the **r** operation, *ar* shows an "a" if it adds a new file or an "r" if it replaces an existing one.

c Create. Normally *ar* creates *afile* if it does not already exist (for the **r** and **q** operations). The **c** modifier suppresses the message normally produced when *afile* is created.

l Local. Place temporary files in the local current working directory, rather than in the default directory **/usr/tmp**. Only the **d**, **m**, **r** and **s** modifiers use temporary files. The **TMPDIR** environment variable overrides use of the **l** modifier.

A Suppress warning messages regarding optional access control list entries. *Ar*(1) does not archive optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file having optional access control list entries.

Only the following combinations are meaningful; no other combination of modifiers with operations have any effect on the operation:

d:	**v**, **l**
r:	**u**, **v**, **c**, **l**, **A**, and **a** | **b** | **i**
q:	**v**, **c**
t:	**v**, **s**
p:	**v**, **s**
m:	**v**, **l**, and **a** | **b** | **i**
x:	**v**, **s**

DIAGNOSTICS

phase error on *file name*
The named file was modified by another process while *ar* was copying it into the archive. When this happens, *ar* exits and the original archive is left untouched.

ar write error: *file system error message*
Ar could not write to a temporary file or the final output file. If *ar* was trying to write the final output file, the original archive will be lost.

Ar reports **cannot create** *file.a*, where *file.a* is an *ar*-format archive file, even if *file.a* already exists. This message is triggered when *file.a* is write-protected or inaccessible.

EXAMPLES

The command:

ar r newlib.a f3 f2 f1 f4

will create a new file (if one does not already exist) in archive format with its constituents entered in the order shown in the above command line.

If you want to replace files f2 and f3 such that the new copies follow file f1 and f3 follows f2, the commands:

ar ma f1 newlib.a f2 f3
ar ma f2 newlib.a f3
ar r newlib.a f2 f3

will produce the desired effect. The archive will now be ordered:

newlib.a: f1 f2´ f3´ f4

where the single quote marks indicate updated files. The first command says "move f2 and f3 after f1 in newlib.a", thus creating the order:

f1 f3 f2 f4

Note that the relative order of f2 and f3 has not changed. The second command says "move f3 after f2 in newlib.a", creating the order:

f1 f2 f3 f4

The third command then replaces the files f2 and f3. Since the files f2 and f3 both already existed in the archive, this sequence of commands could not be simply replaced by:

ar ra f1 newlib.a f2 f3

because the previous position and relative order of f2 and f3 in the archive will be preserved (no matter how the files are specified on the command line), producing the following archive:

newlib.a: f3´ f2´ f1 f4

WARNINGS

If you are the super-user, *ar* will alter any archive file, even if it is write-protected.

If the same file is mentioned twice in an argument list, it might be put in the archive twice.

DEPENDENCIES

Series 800

The **−A** optional modifier is not supported.

FILES

/usr/tmp/ar* temporary files

SEE ALSO

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ar* behaves as if all internationalization variables are set to "C". See *environ*(5).

STANDARDS CONFORMANCE

ar: SVID2, XPG2, XPG3

NAME

as − assembler

SYNOPSIS

as [**−A**] [**−a** afile] [**−o** objfile] [file]

REMARKS

This is a generic page for a machine-dependent assembler. A specific page is provided for each assembler. Refer to manual entry *as_300*(1) for information about the Series 300 assembler or *as_800*(1) for information about the Series 800 assembler.

DESCRIPTION

As assembles the named *file*, or the standard input if no file name is specified. The optional arguments **−A** or **−a** may be used to obtain an assembly listing with offsets and instruction codes. If **−A** is used the listing goes to standard output. If **−a** is used the listing goes to *afile*.

All undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file *objfile*; if that is omitted, *.s* is stripped from the end of the file name (if there) and *.o* is appended to it. This becomes the name of the output file. *As* does not invoke *ld*.

FILES

/usr/tmp/*	temporary files
file.o	object file

SEE ALSO

adb(1), as_300(1), as_800(1), ld(1), nm(1), nm_300(1), nm_800(1), a.out(4), a.out_300(4), a.out_800(4).

DIAGNOSTICS

If the name chosen for the output file is of the form ***?.[cs]**, the assembler issues an appropriate complaint and quits. When syntactic or semantic errors occur, a single-line diagnostic is displayed on *stderr* together with the line number and the file name in which it occurred.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

as: SVID2, XPG2

NAME

as − assembler for MC68000, MC68010, MC68020, and MC68030

SYNOPSIS

as [*options*] [*file*]
as10 [*options*] [*file*]
as20 [*options*] [*file*]

DESCRIPTION

As assembles the named *file* (which usually has a **.s** suffix as in *my_prog.s*). If *file* is not specified or if − is given, standard input is used instead.

As (**/bin/as**) is a driver program that determines the type of processor residing in the host machine, then invokes either **/bin/as10** or **/bin/as20** as appropriate. The **+x** and **+X** options override the test for processor type and invoke either **/bin/as20** or **/bin/as10**, respectively. Both **/bin/as10** and **/bin/as20** can be invoked directly without using *as*.

If *as10* is invoked (directly or through *as*), MC68010 object code is produced, which is compatible with both the MC68010 and MC68020. If *as20* is invoked, whether directly or through *as*, MC68020 object code is produced. Some MC68020 processor instructions are not implemented on the MC68010. The MC68030 instruction set is compatible with the MC68020.

All undefined symbols in the assembly are treated as global.

Options

−L Generate entries in the linker symbol table for local as well as global symbols. Normally, only global and undefined symbols are entered into the table. This option is useful when using *adb*(1) to debug assembly language programs.

−l Generate entries in the linker symbol table for all global and undefined symbols, and for all local symbols except those with **.** or **L** as the first character. This option is useful when using tools like *prof*(1) on files generated by *cc*(1) or *f77*(1). Linker symbol table entries are generated for user-defined local names, but not for compiler-generated local names.

−m Process the input file using the *m4*(1) macro preprocessor before assembling it.

−d Cause *as20* to generate short-displacement forms for MC68010-compatible syntaxes, including forward references, when used with *as20* assembler. This option is ignored by *as10*.

−o *objfile* Cause output object code to be placed in file *objfile*. If **−o** is not specified and the source file is read from *stdin*, the object file is written to *a.out*. If **−o** is not specified and the source file is not *stdin*, the object file is written to a file whose name is created by removing the **.s** suffix (if present) from the basename of filename *file*, then adding a **.o** suffix to the base filename. The object **.o** file is placed in the current directory. To prevent improper interpretation of other options, the name of *objfile* cannot begin with the character − or +. To prevent accidental overwriting of source files, *objfile* cannot end with *.s* or *.c*.

−w Suppress warning messages (errors are not suppressed).

−A Generate an assembly listing with offsets, a hexadecimal dump of the generated code, and the source text. The assembly listing goes to standard out (*stdout*). This option cannot be used when input is *stdin*.

−a *listfile* Generate an assembly listing in file *listfile*. The listing option cannot be used when input is *stdin*. The name of *listfile* cannot end with *.c* or *.s* and cannot start with the character − or +.

−O Enable span-dependent optimization. Optimization is disabled by default.

−V *number* Set the *a_stamp* field in the *a.out* header to be *number*. The **-V** option overrides any **version** pseudo-op in the assembly source. By default the *a_stamp* field is set to zero (see the *HP Assembler Reference Manual*).

+X Invoke **/bin/as10**. Override the test for processor type. This option is ignored when *as10* or *as20* is invoked directly.

+x Invoke **/bin/as20**. Override the test for processor type. This option is ignored when *as10* or *as20* is invoked directly.

Wherever possible, the assembler should be accessed through a compilation system interface program, such as *cc*(1).

Both **/bin/as10** and **/bin/as20** assemblers support the complete MC68000 instruction set. However, if you are writing code for an MC68000 processor, you must limit instructions and program structures to those supported by the microprocessor. Using instructions supported by MC68010 or MC68020 processors on an MC68000 causes an illegal instruction trap during program execution, but might not produce an error during program assembly and loading.

The HP-UX Series 300 assemblers support the complete MC68010, MC68020, and MC68030 instruction sets. However, some instructions are not fully supported by the HP-UX Series 300 hardware, and should not be used in assembly code written for HP-UX Series 300 machines. These instructions are *tas*, *cas*, *cas2*, and *bkpt*.

DIAGNOSTICS

If the name chosen for the output file ends with **.c** or **.s** or starts with the character + or **-**, the assembler issues an appropriate complaint and quits. When a syntactic or semantic error occurs, a single-line diagnostic is produced that includes the line number and file name in which the error occurred.

WARNINGS

If the **−m** option is used, keywords for *m4* cannot be used as symbols in the input file because *m4* cannot determine which are assembler symbols and which are real *m4* macros.

The displacement value for the **movp** instruction must be a first-pass absolute 16-bit value.

Expressions cannot have more than one forward-referenced symbol, except for the special form <symbol>-<symbol>.

AUTHOR

As was developed by HP.

FILES

/usr/tmp/* Temporary files, which can be changed by using TMPDIR (see *tmpnam*(3S)).

file.o Object file

SEE ALSO

as_800(1) (Series 800 Implementation), adb(1), astrn(1), atrans(1), cc(1), f77(1), ld(1), m4(1), nm_300(1), nm_800(1), prof(1), tmpnam(3s), a.out_300(4), a.out_800(4).
HP-UX Assembler Reference Manual and ADB Tutorial for Series 300 Computers.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

as – assembler (Precision Architecture)

SYNOPSIS

as [[option] ... [file] ...] ...

DESCRIPTION

As assembles the named *file*, or the standard input if no file name is specified. The optional argument **–l** may be used to obtain an assembly listing with offsets.

The output of the assembly is left on the file *objfile*. If that is omitted, *.s* is stripped from the end of the file name (if there) and *.o* is appended to it. This becomes the name of the output file.

The output of *as* is not optimized. *As* creates relocatable object files which must be processed by *ld* to be made executable.

Cc assembles *.s* files together with *pcc_prefix.s* and subsequently invokes *ld*.

Options

As recognizes the following options.

–e	An unlimited number of errors will be tolerated before the assembly process is abandoned. Normally, only a hundred errors are allowed.
–f	Procedures by default will be callers of other procedures. The normal default is that procedures do not call other procedures.
–l	Listing to standard output is made of the program after assembly. This listing shows offsets of instructions and actual values for fields.
–o *outfile*	Produce an output object file by the name *outfile* instead of using the default *.o* suffix.
–s	The output file will have suffix *.ss* and be of a format suitable for conversion to the ROM burning programs.
–u	Unwind descriptors will not be created. In order to avoid the need for .CALLINFO, it must also be the case that .ENTER and .LEAVE have not been used.
–v *xrfile*	Provides the name of a file to which cross reference data is written.

DIAGNOSTICS

When syntactic or semantic errors occur, a single-line diagnostic is displayed on *stderr* together with the line number and the file name in which it occurred.

WARNINGS

As does not do macro processing.

Trailing operands (except for a pc_relative branch displacement) may be omitted and default to zero. Trailing commas may also be omitted. Leading commas are ignored.

FILES

/lib/pcc_prefix.s	space and register definitions
/usr/include/hard_reg.h	hardware register equates
/usr/include/soft_reg.h	follows calling convention
/usr/include/std_space.h	space and subspace declarations
/lib/as_msgs.cat	error message catalog
file.o	object file

SEE ALSO

cc(1), ld(1), adb(1), nm_800(1), nm_300(1).

Precision Architecture Assembler Technical Reference Manual.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

asa – interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

Asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names or "–" is supplied. The first character of each line is assumed to be a control character. The following control characters are interpeted as indicated:

(blank)	Output a single new-line character before printing.
0	Output two new-line characters before printing.
1	Output a new-page character before printing.
+	Overprint previous line.

Lines beginning with other than the above characters are treated the same as lines beginning with a blank. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic is sent to standard error. This program forces the first line of each input file to start on a new page.

To view the output of FORTRAN programs which use ASA carriage control characters and have them appear in normal form, *asa* can be used as a filter:

a.out | asa | lp

The output, properly formatted and paginated, is then directed to the line printer. FORTRAN output previously sent to a file can be viewed on a user terminal screen by using:

asa *file*

SEE ALSO

efl(1), f77(1), fsplit(1), ratfor(1).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

astrn – translate assembly language

SYNOPSIS

astrn [filename]

Remarks:

Astrn is implemented on the Series 300 only.

DESCRIPTION

Astrn translates an assembly language *source file* from previous HP-UX **Series 300** assembly language syntax to new **Series 300** HP-UX assembly language syntax. If no *filename* is given, input is assumed to come from **stdin.**

If an opcode is not recognized, a warning message is given and the entire line is passed through unchanged. For any syntax error detected such that translation cannot continue, *astrn* reports an error and translation terminates.

Lines longer than 132 characters are truncated to 132 characters.

For a line beginning with '*' (indicating a comment), the '*' is translated to a '#' but is preceded by a blank to allow preprocessing with *cpp(1).*

Absolute displacements off the program counter cannot be guaranteed to translate correctly. Any line referencing the program counter will be flagged by a warning message.

Certain capabilities supported on the old assembler are not accepted by the new assembler. These include:

The *alias* and *include* pseudo-ops are not supported. An error message is given and translation terminates.

The new assembler restricts expressions involving forward references for which *astrn* makes no check. Such references may involve only a single symbol, a symbol plus or minus an absolute expression, or the subtraction of two symbols.

The characters '$', ' '?', and '\177' are no longer accepted as valid identifier characters. These are translated to 'S', 'A', 'Q', and 'D' respectively, and a warning is issued.

Span-dependent branches **j***cc* are translated to **b***cc.w*.

An identifier equated to a register name will be translated but the assembler will report an error.

Local labels are translated to a concatenation of the nearest previous ordinary label and the local label itself. This includes changing the '$' to a 'S'.

SEE ALSO

as(1), atrans(1).

NAME

at, batch − execute commands at a later time

SYNOPSIS

at [−**q***queue*] *time* [*date*] [[**next** | +*increment*] *time_designation*] *job* ...
at −**r** *job* ...
at −**l** [*job* ...]

batch

DESCRIPTION

At and *batch* read commands from standard input to be executed at a later time. *At* allows you to specify when the commands are to be executed. Jobs queued with *batch* execute when system load level permits. The −**q** option specifies which queue the job should be submitted to (see *queuedefs*(4)). Queues **a** through **y** (except **c**) can be used. *At* uses queue **a** by default. All queues require a time designation except for queue **b** which runs as soon as system load level permits. *At* −**r** removes jobs previously scheduled with *at*. The −**l** option reports all jobs scheduled for the invoking user.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, *umask* (see *umask*(1)) and *ulimit* (see *ulimit*(2)) are retained when the commands are executed (see *proto*(4)). Open file descriptors, traps, and priority are lost.

Only users whose names appear in file **/usr/lib/cron/at.allow** can run *at*. If that file does not exist, file **/usr/lib/cron/at.deny** is checked to determine if the user should be denied access to *at*. If neither file exists, only root is allowed to submit a job. If only **at.deny** exists but is empty, global usage is permitted. The allow/deny files consist of one user name per line.

The argument *time* can be specified as one, two, or four digits. One- and two-digit numbers represent hours; four digits are hours and minutes. Alternately, *time* can be specified as two numbers separated by one of the following characters: **:** , **'** , **h** , **.** or **,** . If defined in *langinfo*(3C), special time unit characters can be used. A suffix **am** or **pm** can be appended. Otherwise a 24-hour clock time is understood. For example, 8:15, 8'15, 8h15, 8.15 and 8,15 are read as 15 minutes after 8 in the morning. The suffix **zulu** can be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

An optional argument *date* can be specified as either a day of the week (fully spelled out or abbreviated) or a date consisting of a day, a month, and optionally a year. The day and year fields must be numeric, and the month can be either fully spelled out, abbreviated, or numeric. These three fields can be in any order, and separated by punctuation marks such as **/**, **-**, **.** or **,**. If defined in *langinfo*(3C), special date unit characters can be present. Two special "days", **today** and **tomorrow**, are also recognized. If no *date* is given, **today** is assumed if the given time is greater than the current time and **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed. If a given date is ambiguous (such as 2/5), the D_T_FMT string (if defined in *langinfo*(3C)) is used to resolve the ambiguity.

The optional argument **next**, followed by a *time_designation* of **minutes**, **hours**, **days**, **weeks**, **months**, or **years**, lets the user schedule a task to be executed when the specified *time_designation* has elapsed. A numerical operator, +*increment*, enables the user to schedule the task several hours, days, weeks, months, or years in advance. (See EXAMPLES.) The **next** argument is equivalent to +**1** *increment*. Both plural and singular forms of *time_designation are accepted.*

The words **today**, **tomorrow**, **noon**, **midnight**, **now**, **minutes**, **hours**, **days**, **weeks**, **months**, **years** and their singular forms are replaced by the local language equivalent (see EXTERNAL INFLUENCES below).

At and *batch* write the job number and schedule time to standard error.

Batch submits a batch job. It is similar to **at now**, but with the following differences: *batch* goes into a different queue; **at now** responds with error messages.

At **-r** removes jobs previously scheduled by *at* or *batch*. The job number is the number returned by the *at* or *batch* command. You can also get job numbers by typing *at* **-l**. You can remove only your own jobs unless you are the superuser.

RETURN VALUE

Exit code 0 is returned upon successful completion, otherwise 1 is returned.

DIAGNOSTICS

At complains about syntax errors and out-of-range times.

If the given time is equal to the current time (in hours and minutes), *at* returns an error message that reads "too late."

If login shell is not **/bin/sh**, *at* produces a warning message as a reminder that *at* jobs are executed using **/bin/sh**.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *Sh*(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

The following sequence can be used at a terminal to redirect output:

```
batch
nroff filename >outfile
<control-D>
```

This sequence demonstrates redirecting standard error to a pipe and is useful in a shell procedure. Note that the sequence of output redirection specifications is significant:

```
batch <<! nroff filename 2>&1 >outfile | mail loginid !
```

To perform a task at 5:00 am next Tuesday, use

```
at 500 tuesday next week
```

To perform a task at 5:00 am one week from Tuesday (that is, 2 Tuesdays in advance) use

```
at 500 tuesday + 2 weeks
```

To have a job reschedule itself, invoke *at* from within the shell procedure by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

The following commands show several forms recognized by *at* and include native language usage:

```
at 0815 Jan 24
at 8:15 Jan 24
at 9:30am tomorrow
at now + 1 day
at 5 pm Friday
at 17:40 Tor.              /* in Danish */
at 17h46 demain            /* in French */
at 5:30  26. Feb. 1988     /* in German */
at 12:00 26-02             /* in Finnish */
```

WARNINGS

If the *date* argument begins with a number and the *time* argument is also numeric without

suffix, the *time* argument should be a four-digit number that can be correctly interpreted as hours and minutes.

Do not use both **next** and + *increment* within a single *at* command; only the first operator is accepted and the trailing operator is ignored. No warning or error is produced.

AUTHOR

At was developed by AT&T and HP.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/spool/cron/atjobs	spool area
/usr/lib/cron/queuedefs	scheduling information
/usr/lib/cron/.proto	prototype information

SEE ALSO

cron(1M), crontab(1), queuedefs(4), proto(4), kill(1), mail(1), nice(1), ps(1), sh(1), hpnls(5).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

LANG determines the translation of the words *today, tomorrow, noon, midnight, now, minutes, hours, days, weeks, months, years, next,* and their singular forms. LANG also determines the language in which messages are displayed.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *at* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

at: SVID2, XPG2, XPG3

batch: SVID2, XPG2, XPG3

NAME

atime – time an assembly language instruction sequence

SYNOPSIS

atime [*options*] *input_file* [*output_file*]

DESCRIPTION

The *atime* utility provides the means to time 680X0 assembly code sequences. It takes the *input_file* containing a code sequence and returns performance information to the user. This information can then be compared against information from other code sequence analyses to determine an optimal code sequence. Output is directed to standard output by default or if the *output_file* is "–", or to *output_file* if specified.

Additional features allow specifying sets of input data and the relative probability that each of these will occur, obtaining an execution profile of a code sequence being evaluated, automatically cross-checking results between analyses, and conveniently logging results.

Atime has three modes of operation. Performance analysis is the default mode where a code sequence is executed many times in a loop with *atime* calculating and reporting the average time per iteration. In the execution profiling mode, *atime* runs all or selected data sets and reports the number of times each executable instruction is hit. The third mode is assertion listing. Asserting particular values in a code sequence assures that various algorithms produce identical results. This mode causes asserted values for all or selected data sets to be listed. This output can be used as verification data for subsequent performance analyses and execution profiles.

Options

–**a***file*	Specify an assertion data file (created by a previous run with the –**l** option.)
–**i***count*	Specify the minimum number of timing iterations.
–**l**[*name*]	Print asserted values. The *name* specifies a **dataset** in the input file. Multiple –**l** options are allowed. Omitting *name* causes assertions for all data sets to be listed. Output can be used to create an assertion file for subsequent *atime* runs.
–**n**	Turn off code sequence listing.
–**p**[*name*]	Perform execution profiling and print the hit count for each timed instruction. The *name* specifies a **dataset** in the input file to analyze. If there are multiple –**p** options, printed counts will be the sum for all designated data sets. Omitting *name* causes profiling for all data sets.
–**t**"*text*"	Specify an output title.

Instructions

The following *atime* instructions can appear in the input file and have a format similar to standard assembler instructions. However, they cannot be labelled, each must be placed on a separate line, and comments cannot follow on the same line. For instructions that have a corresponding command line option, the latter will take precedence when used.

assert.{**b**|**w**|**l**} *name,location*

Verify a datum. The *name*, an alphabetic character followed by zero or more alphanumeric or underscore characters, identifies an asserted datum across *atime* executions. Any data addressing mode, such as *%d0* or *4(%a4,%d2.w)* can be used for *location*. Executing an **assert** instruction during performance analysis or execution profiling compares the asserted *location* with the corresponding name in an assertion data file. Verification also occurs in the assertion listing mode if an assertion file is specified, although the primary function here is to print name/value pairs.

assert "*file*" Specify an assertion data file created from a prior run of *atime* with the **−l** option.

code even
code odd Change the code to even or odd word alignment.

comment *text* Specify comment text for output.

dataname *name,name,...name*
Define the names of the data entries in **dataset** instructions. Names must begin with a **$** and are followed by one or more alphanumeric or underscore characters.

dataset *name[(count)],datum,datum,...datum*
Define one data set. *Name* identifies the data set for use with the **−l** or **−p** options and must be an alphabetic character followed by zero or more alphanumeric or underscore characters. An optional *count* indicates the relative number of times that the data set will be used during timing. For example, a data set with a *count* of 10 will execute ten times for each seven times that a data set with a *count* of 7 is executed. The default *count* is 1. As a particular data set is under consideration, each of its data is treated as a string for replacing its corresponding **dataname** name in any assembly instruction where that name occurs.

include "*file*" Include text from the given *file*. Nested **include**s are not allowed.

iterate *count* Specify the number of timing iterations for performance analysis. Because the actual count used by *atime* must be an integer multiple of the sum of the counts in all **dataset** instructions, the *count* specified here is taken as a minimum. If neither this instruction nor a **−i** option appears, the default count is 1000000.

ldopt *options* Specify options to pass to the link editor.

nolist Turn off code sequence listing.

output "*file*" Append output from a performance analysis to *file*.

stack even
stack odd Adjust the stack for even or odd word alignment.

time Designate a code section for timing beginning after this instruction and continuing up to a **verify** instruction or end-of-file.

title Specify an output title.

verify Designate a section of code to be used for algorithm verification beginning after this line and continuing up to the end-of-file. This section will usually contain one or more **assert** instructions.

Input file

The input file contains assembly code source text and *atime* instructions and has the following four sections.

atime initialization section - This starts at the beginning of the file and continues up to the first 680X0 assembly instruction or a **time**, **code**, or **stack** instruction. It can contain **assert** *file*, **comment**, **dataname**, **dataset**, **include**, **iterate**, **ldopt**, **nolist**, **output**, and **title**.

code initialization section - The code following the *atime* initialization section continues up to a **time** instruction and typically does the setup for the code to be timed. It can contain any valid 680X0 assembler instruction or pseudo-op or any of the following *atime* instructions: **code even**, **code odd**, **stack even**, **stack odd**, or **include**. It is only in this section that **dataname**

*name*s can be used and each *name* replacement must yield a valid 680X0 instruction.

timed section - This section starts at the **time** pseudo-op and continues up to a **verify** instruction or end-of-file. It can contain any valid 680X0 assembler instruction or pseudo-op or the *atime* **include** instruction.

verify section - This section starts at the **verify** instruction and continues up to the end-of-file. It can contain any valid 680X0 assembler instruction or pseudo-op or the *atime* instructions **include** or **assert.{b|w|l}**.

There must be no branching between input file sections. Each must be entered by falling into it from the previous section. Macros for *m4*(1) are not supported, nor are multiple instructions per line. Assembly code can have references to external variables or routines as long as it is guaranteed that these will be resolved during link editing.

DIAGNOSTICS

Error messages from *atime* are self-explanatory. Additional error messages may be generated from the assembler or link editor. If assembly fails, an intermediate temporary file will be retained with the error message indicating its name. This file contains a substantial number of comments to aid in correlating assembly errors back to the actual errors in the input file.

EXAMPLES

To evaluate an algorithm to find the maximum of three integers, the input file to *atime* could contain the following code sequence:

```
        title     Find the maximum of three integers
        comment   Developed by T. R. Crew
        comment   August 15, 1988
        nolist
        dataname            $arg1,   $arg2,   $arg3
        dataset   max1(70),    10,      4,      2
        dataset   max2(35),     5,     11,      0
        dataset   max3(20),     8,     13,     21
        iterate   500000
        assert    "assertfile"
        output    "logfile"
        ldopt     -lm -lc
        stack     even
        mov.l     &$arg1,%d0
        mov.l     &$arg2,%d1
        mov.l     &$arg3,%d2
        code      even
        time
        cmp.l     %d0,%d1
        bge.b     L1
        exg       %d0,%d1
L1:     cmp.l     %d0,%d2
        bge.b     L2
        exg       %d0,%d2
L2:
        verify
        assert.l  max,%d0
```

WARNINGS

Because *atime* determines performance information empirically, valid results will be obtained only if it is run on a quiescent single-user system.

AUTHOR

Atime was developed by HP.

FILES

/bin/as	assembler, as(1)
/bin/ld	link editor, ld(1)

SEE ALSO

as(1), gprof(1), ld(1), prof(1).

NAME

atrans – translate assembly language

SYNOPSIS

atrans [**–n**] [filename]

Remarks:

Atrans is implemented on the Series 300 only. This page describes Series 300 HP-UX starting at Release 5.15.

DESCRIPTION

Atrans translates an assembly language *source file* from **Series 300** Pascal workstation assembly language syntax to **Series 300** HP-UX assembly language syntax. If no *filename* is given, input is assumed to come from **stdin.**

If an opcode is not recognized, the entire line is passed through unchanged. For any syntax error detected such that a line cannot be translated, *atrans* issues an error message.

Lines longer than 132 characters are truncated to 132 characters.

Absolute displacements off the program counter cannot be guaranteed to translate correctly. Any line referencing the program counter will be flagged by a warning message.

The HP-UX assembler restricts expressions involving forward references for which *atrans* makes no check. Such references may involve only a single symbol, a symbol plus or minus an absolute expression, or the subtraction of two symbols.

The characters '$' and '@' are not accepted as valid identifier characters on the HP-UX assembler. These are translated to 'S' and 'A' respectively and a warning is issued.

Lines containing the following list of **Series 300** Pascal workstation pseudo-ops have no parallel in **Series 300** HP-UX syntax and are translated as comment lines: *decimal, end, llen, list, lprint, nolist, noobj, nosyms, page, spc, sprint, ttl.*

Lines containing the *mname, include,* or *src* pseudo-ops are translated as comment lines, and a warning is printed stating these are not supported by the *Series 300* HP-UX assembler.

The pseudo-ops, *def, refa,* and *refr,* are translated as *global.*

Certain pseudo-ops require manual intervention to translate. Each line containing these pseudo-ops will cause a message to be printed stating that an error will be generated by the **Series 300** HP-UX assembler. These pseudo-ops are: *com, lmode, org, rorg, rmode, smode, start.*

When specifying certain addressing modes, the Pascal workstation assembler may allow operands to appear out of order, whereas the HP-UX assembler does not. *Atrans* does not rearrange these into proper order.

The **–n** option converts groups of blanks to tabs.

SEE ALSO

as(1), astrn(1).

NAME

awk – text pattern scanning and processing language

SYNOPSIS

awk [**−F***c*] [*prog* | **−f** *awkfile*] [*parameters*] [*file* ...]

Remarks:

For a newer version of this program, see *nawk*(1).

DESCRIPTION

Awk scans each input *file* for lines that match any pattern specified in *prog*. Each pattern in *prog* is associated with an action that is performed when a line of a *file* matches the pattern. A set of pattern-action statements can appear literally as *prog*, or in a file specified by **−f** *awkfile*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

Parameters, in the form x=... y=... etc., can be passed to *awk*.

Awk reads each *file* in order; if none is given, *awk* reads the standard input. The file name "hyphen" (−) also represents the standard input. Each line of a *file* is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is comprised of fields separated by white space. (This default can be changed by using field separators (see FS below). The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
for ( variable in array ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the entire line. Expressions take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The **C** operators ++, −−, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted *x*[*i*]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted ("); single quotes (') are not recognized.

The *print* statement prints its arguments to the standard output (or to a file if *>expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format (see *printf*(3S)).

The built-in function *length* returns the length of its argument taken as a string, or the length of the entire line if no argument is given. There are also built-in functions *exp*, *log*, *sqrt*, and *int*.

The last truncates its argument to an integer; The built-in function *split(s1,arr.s2)* assigns the fields of the string s1 to successive elements of the array *arr* , using the characters in string s2 as field separators. Assignments start at array element *arr[1]*. *substr*(*s, m, n*) returns the *n*-character substring of *s* that begins at position *m*. The built-in function *index(s1,s2)* returns the leftmost position where the string s2 occurs in s1 or zero if s2 does not occur in s1. The *getline* function immediately reads the next input record. Fields NR and $0 are all set, but control is left at exactly the same spot in the *awk* program. The *getline* function returns 0 on end of file, and 1 otherwise. The function *sprintf*(*fmt, expr, expr, ...*) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (**!**, **||**, **&&**, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and must be Extended Regular Expressions (see *regexp*(5)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between each occurrence of the first and second pattern.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

where a relop is any of the six relational operators in C, and a matchop is either ~ (for *contains*) or !~ (for *does not contain*). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END can be used to capture control before the first input line and after the last input line is read. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

BEGIN { FS = *c* }

or by using the **−F***c* option.

Other variable names with special meanings include RS, the input record separator (set by new-line); NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default **%.6g**).

DIAGNOSTICS

Too many fields The number of fields in a record exceeded 99.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
        { s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
       { print }
```

command line: awk −f program n=5 input

WARNINGS

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ( ) to it.

Awk(1) is obsolescent and will be replaced by *nawk(1)* in a future HP-UX release.

SEE ALSO

lex(1), nawk(1), sed(1), malloc(3X), environ(5), lang(5), regexp(5).

Awk: A Programming Language for Manipulating Data, tutorial in *HP-UX Concepts and Tutorials: Text Editors and Processors.*

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used when evaluating regular expressions and by the relational operators when performing comparisons on string values.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character-class expressions in regular expressions.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *awk* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported except that variable names must contain only ASCII characters and regular expressions must contain only single-byte characters.

STANDARDS CONFORMANCE

awk: SVID2, XPG2, XPG3

NAME

banner – make posters in large letters

SYNOPSIS

banner *strings*

DESCRIPTION

Banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

Each argument is printed on a separate line. Note that multiple-word arguments must be enclosed in quotes in order to be printed on the same line.

EXAMPLES

The command:

banner "Good luck" Susan

prints the message **Good luck Susan** in large letters on the screen. The words **Good luck** are displayed in one line, and **Susan** is displayed in a second line.

SEE ALSO

echo(1).

STANDARDS CONFORMANCE

banner: SVID2, XPG2, XPG3

NAME

basename, dirname – extract portions of path names

SYNOPSIS

basename string [suffix]
dirname [string]

DESCRIPTION

Basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside command substitution marks (`...`) within shell procedures.

Dirname delivers all but the last level of the path name in *string*. If *string* does not contain a directory component, *dirname* returns ".", indicating the current working directory.

EXAMPLES

The following shell script, invoked with the argument **/usr/src/cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

The following example sets the shell variable **NAME** to **/usr/src/cmd**:

```
NAME=`dirname /usr/src/cmd/cat.c`
```

RETURNS

Both commands return **0** for success. Both commands return **1** when given an incorrect number of arguments.

SEE ALSO

expr(1), sh(1).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of string and, in the case of basename, suffix as single and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *basename* and *dirname* behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

basename: SVID2, XPG2, XPG3

dirname: SVID2, XPG2, XPG3

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [**–c**] [**–l**] [*file* ...]

DESCRIPTION

Bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input.

Options:

–c Compile only. **Bc** is actually a preprocessor for *dc*(1), which *bc* invokes automatically. Specifying **–c** prohibits invoking *dc*, and sends the *dc* input to standard output.

–l causes an arbitrary-precision math library to be predefined. As a side-effect, the scale factor is set.

Program Syntax:

a letter in the range a–z;
expression;
statement;
relational expression.

Comments:

Comments are enclosed in /* and */.

Names:

Names include:

simple variables: L
array elements: L [E]
The words "ibase", "obase", and "scale"
stacks: L

Other Operands

Other operands include:

Arbitrarily long numbers with optional sign and decimal point.
(E)
sqrt (E)
length (E) number of significant decimal digits
scale (E) number of digits right of decimal point
L (E , ... , E)
Strings of ASCII characters enclosed in quotes (").

Arithmetic Operators:

Arithmetic operators yield an E as a result and include:

+ – * / % ^ (% is remainder (not mod, see below); ^ is power)"
++ – – (prefix and append; apply to names)
= =+ =– =* =/ =% =^

Relational Operators

Relational operators yield an R when used as **E** *op* **E**:

== <= >= != < >

Statements

E
{ S ; ... ; S }
if (R) S

```
while ( R ) S
for ( E ; R ; E ) S
null statement
break
quit
```

Function Definitions:

```
define L ( L ,..., L ) {
        auto L, ... , L
        S; ... S
        return ( E )
}
```

Functions in −l Math Library:

Functions in the −l math library include:

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value. Trigonometric angles are in radians where 2 pi radians = 360 degrees.

The value of a statement that is an expression is printed unless the main operator is an assignment. No operators are defined for strings, but the string is printed if it appears in a context where an expression result would be printed. Either semicolons or new-lines can separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(1). Assignments to *ibase* or *obase* set the input and output number radix respectively, again as defined by *dc*(1).

The same letter can be used simultaneously as an array, a function, and a simple variable. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

The % operator yields the remainder at the current scale, not the integer modulus. Thus, at scale 1, **7 % 3** is .1 (one tenth), not 1. This is because (at scale 1) **7 / 3** is 2.3 with .1 as the remainder.

EXAMPLES

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

defines a function to compute an approximate value of the exponential function, and

```
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

WARNINGS

There are currently no && (AND) or || (OR) comparisons.
The *for* statement must have all three expressions.
Quit is interpreted when read, not when executed.
Bc's parser is not robust in the face of input errors. Some simple expression like 2+2 helps get it back into phase.

FILES

/usr/bin/dc	desk calculator proper
/usr/lib/lib.b	mathematical library

SEE ALSO

bs(1), dc(1).

BC: An Arbitrary-Precision Desk-Calculator Language, tutorial in *HP-UX Concepts and Tutorials: Shells and Miscellaneous Tools.*

NAME

bdiff – big diff

SYNOPSIS

bdiff file1 file2 [n] [**–s**]

DESCRIPTION

Bdiff is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is –, the standard input is read. The optional **–s** (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*. If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

bfs – big file scanner

SYNOPSIS

bfs [–] name

DESCRIPTION

Bfs is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 512 characters, including new-line, per line. *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional – suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

Bfs supports the Basic Regular Expression (RE) syntax (see *regexp*(5)) with the addition that a null RE (e.g., **//**) is equivalent to the last RE encountered. All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides / and **?**: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

The **e**, **g**, **v**, **k**, **n**, **p**, **q**, **w**, =, **!** and null commands operate as described under *ed*. Commands such as ––, +++–, +++=, **–12**, and **+4p** are accepted. Note that **1,10p** and **1,10** will both print the first ten lines. The **f** command only prints the name of the file being scanned; there is no *remembered* file name. The **w** command is independent of output diversion, truncation, or crunching (see the **xo**, **xt** and **xc** commands, below). The following additional commands are available:

xf *file* Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the **xf**. **Xf** commands may be nested to a depth of 10.

xo [*file*] Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: *label* This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the **:** and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(. , .)**xb**/*regular expression*/*label*

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, **.** is set to the line matched and a jump is made to *label*. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

xb/^/ label

is an unconditional jump.
The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xn List the marks currently in use (marks are set by the **k** command).

xt *number* Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

xv[*digit*][*spaces*][*value*]
The variable name is the specified *digit* following the **xv**. **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. **Xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6**:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

```
g/%5/p
```

would globally search for the characters **100** and print each line containing a match. To escape the special meaning of **%**, a \ must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.
Another feature of the **xv** command is that the first line of output from an HP-UX command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example:

```
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a \.

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*
xbn *label* These two commands will test the last saved *return code* from the execution of an HP-UX system command (**!***command*) for a zero or nonzero value, respectively, and cause a branch to the specified label. The two examples below both search for the next five lines containing the string **size**.

First example:

```
xv55
: l
/size/
xv5!expr %5 – 1
```

```
!if [ %5 != 0 ] ; then exit 2 ; fi
xbn 1
```

Second Example:

```
xv45
: 1
/size/
xv4!expr %4 - 1
!if [ %4 = 0 ] ; then exit 2 ; fi
xbz 1
```

xc [*switch*] If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it isn't. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the classification of characters as letters, and the characters matched by character class expressions in regular expressions.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *bfs* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

SEE ALSO

csplit(1), ed(1), lang(5), regexp(5).

NAME

bs – a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [*file* [*args*]]

DESCRIPTION

Bs is a remote descendant of Basic and Snobol4 with some C language added. *Bs* is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the *trace* and *dump* statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from *file* are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see *compile* and *execute* below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

Bs programs are made up of input lines. If the last character on a line is a \, the line is continued. *Bs* accepts lines of the following form:

```
statement
label  statement
```

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A *bs* statement is either an expression or a keyword followed by zero or more expressions. Some keywords (*clear, compile, !, execute, include, ibase, obase,* and *run*) are always executed as they are compiled.

Statement Syntax:

expression — The expression is executed for its side effects (value, assignment, or function call). The details of expressions follow the description of statement types below.

break — *Break* exits from the innermost *for/while* loop.

clear — Clears the symbol table and compiled statements. *Clear* is executed immediately.

compile [expression] — Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A *clear* is associated with this latter case. *Compile* is executed immediately.

continue — *Continue* transfers to the loop-continuation of the current *for/while* loop.

dump [name] — The name and current value of every non-local variable is printed. Optionally, only the named variable is reported. After an error or interrupt, the number of the last statement is displayed. The user-function trace is displayed after an error or *stop* that occurred in a function.

edit — A call is made to the editor selected by the EDITOR environment variable if it is present, or *ed*(1) if EDITOR is undefined or null. If the *file* option is present on the command line, that file is passed to the editor as the file to edit. (Otherwise no file name is used.) Upon exiting the editor, a *compile* statement (and associated *clear*) is executed giving that file name as it's argument.

exit [expression]
Return to system level. The expression is returned as process status.

execute Change to immediate execution mode (an interrupt has a similar effect). This statement does not cause stored statements to execute (see *run* below).

for name = expression expression statement
for name = expression expression
...

next

for expression , expression , expression statement
for expression , expression , expression
...

next The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

fun f([a, ...]) [v, ...]
...

nuf *Fun* defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested. Calling an undefined function is permissible, see function calls below.

freturn A way to signal the failure of a user-written function. See the interrogation operator (**?**) below. If interrogation is not present, *freturn* merely returns zero. When interrogation *is* active, *freturn* transfers to that expression (possibly bypassing intermediate function returns).

goto name Control is passed to the internally stored statement with the matching label.

ibase N *Ibase* sets the input base (radix) to *N*. The only supported values for *N* are the constants **8**, **10** (the default), and **16**. Hexadecimal values 10–15 are entered as **a**–**f**. A leading digit is required (i.e., **f0a** must be entered as **0f0a**). *Ibase* (and *obase*, below) are executed immediately.

if expression statement
if expression
...

[else ...]

fi The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings **0** and "" (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The concatenation of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if* ... *elif* ... [*else* ...] sequence.

include expression
The expression must evaluate to a file name. The file must contain *bs* source

statements. Such statements become part of the program being compiled. *Include* statements may not be nested.

obase N *Obase* sets the output base to *N* (see *ibase* above).

onintr label

onintr The *onintr* command provides program control of interrupts. In the first form, control will pass to the label given, just as if a *goto* had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause *bs* to terminate.

return [expression]
The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run The random number generator is reset. Control is passed to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop Execution of internal statements is stopped. *Bs* reverts to immediate mode.

trace [expression]
The *trace* statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace* expression value.

while expression statement
while expression
...

next *While* is similar to *for* except that only the conditional expression for loop-continuation is given.

! shell command
An immediate escape to the Shell.

... This statement is ignored. It is used to interject commentary in a program.

Expression Syntax:

name A name is used to specify a variable. Names are composed of a letter (uppercase or lowercase) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open*( ) below).

name ([expression [, expression] ...])
Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value. If the function is undefined, the call history to the call of that function is printed, and a request for a return value (as an expression) is made. The result of that expression is taken to be the result of the undefined function. This permits debugging programs where not all the functions are yet defined. The value is read from the current input file.

name [expression [, expression] ...]
This syntax is used to reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as

a specifier for the name. The resulting array reference is syntactically identical to a name; **a[1,2]** is the same as **a[1][2]**. The truncated expressions are restricted to values between 0 and 32 767.

number — A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an **e** followed by a possibly signed exponent.

string — Character strings are delimited by " characters. The \ escape character allows the double quote (**\"**), new-line (**\n**), carriage return (**\r**), backspace (**\b**), and tab (**\t**) characters to appear in a string. Otherwise, \ stands for itself.

(expression) — Parentheses are used to alter the normal order of evaluation.

(expression , expression [, expression ...]) [expression]
The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero.
The expression:

(False, True)[a == b]

has the value **True** if the comparison is true.

? expression — The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the *Programming Tips* section below), the result of the *eval* built-in function, and for checking the return from user-written functions (see *freturn*). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

− expression — The result is the negation of the expression.

++ name — Increments the value of the variable (or array reference). The result is the new value.

−− name — Decrements the value of the variable. The result is the new value.

! expression — The logical negation of the expression. Watch out for the shell escape command.

expression — *operator* expression Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence):

= — = is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

_ — _ (underscore) is the concatenation operator.

& | — **&** (logical and) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.

< <= > >= == != — The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to) return one if their

arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: *a>b>c* is the same as *a>b & b>c*. A string comparison is made if both operands are strings.

`+ -` Add and subtract.

`* / %` Multiply, divide, and remainder.

`^` Exponentiation.

Built-in Functions:

Dealing with arguments

arg(i) is the value of the *i*-th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument (*arg*(0) returns **bs**).

narg() returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

abs(x) is the absolute value of *x*.

atan(x) is the arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.

ceil(x) returns the smallest integer not less than *x*.

cos(x) is the cosine of *x* (radians).

exp(x) is the exponential function of *x*.

floor(x) returns the largest integer not greater than *x*.

log(x) is the natural logarithm of *x*.

rand() is a uniformly distributed random number between zero and one.

sin(x) is the sine of *x* (radians).

sqrt(x) is the square root of *x*.

String operations

size(s) the size (length in bytes) of *s* is returned.

format(f, a) returns the formatted value of *a*. *F* is assumed to be a format specification in the style of *printf*(3S). Only the **%...f**, **%...e**, and **%...s** types are safe. Since it is not always possible to know whether **a** is a number or a string when the **format** call is coded, coercing **a** to the type required by **f** by either adding zero (for **e** or **f** format) or concatenating (_) the null string (for **s** format) should be considered.

index(x, y) returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

trans(s, f, t) Translates characters of the source *s* from matching characters in *f* to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

substr(s, start, width)
returns the sub-string of *s* defined by the *start*ing position and *width*.

match(string, pattern)
mstring(n) The *pattern* is a regular expression according to the Basic Regular Expression definition (see *regexp*(5)). The *mstring* function returns the *n*-th (1 <= *n* <= 10) substring of the subject that occurred between pairs of the pattern symbols

\(and \) for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with ^). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a-z]\)") == 6
mstring(1) == "b"
```

File handling

open(name, file, function)

close(name) The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may be **1)** a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively; **2)** a string representing a file name; or **3)** a string beginning with an **!** representing a command to be executed (via *sh* −*c*). The *function* argument must be either **r** (read), **w** (write), **W** (write without new-line), or **a** (append). After a *close*, the *name* reverts to being an ordinary variable. If *name* was a pipe, a *wait*(2) is executed before the close completes. The *bs* **exit** command does not do such a wait. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

access(s, m) executes *access*(2).

ftype(s) returns a single character file type indication: **f** for regular file, **p** for FIFO (i.e., named pipe), **d** for directory, **b** for block special, or **c** for character special.

Tables

table(name, size)

A table in *bs* is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a *bs* variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. *Bs* prints an error message and stops on table overflow. The result of *table* is *name*.

item(name, i)

key() The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the "subscript" of the previous *item* call. It fails (or in the absence of an *interrogate* operator, returns null) if there was no valid subscript for the previous *item* call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table; for example:

```
table("t", 100)
 ...
# If word contains "party", the following expression adds one
# to the count of that word:
++t[word]
 ...
# To print out the the key/value pairs:
for i = 0, ?(s = item(t, i)), ++i  if key()  put = key()_":"_s
```

If the interrogation operator is not used, the result of *item* is null if there are no further elements in the table. Null is, however, a legal "subscript".

iskey(name, word)

The *iskey* function tests whether the key **word** exists in the table **name** and returns one for true, zero for false.

Odds and ends

eval(s) The string argument is evaluated as a *bs* expression. The function is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

```
name = "xyz" eval("++"_ name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control *bs* error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a *goto* to the label *L* (if it exists):

```
label="L"
if !(?eval("goto "_ label)) puterr = "no label"
```

plot(request, args)

If the *tplot* command is available, the *plot* function produces output on devices recognized by *tplot*. The *requests* are as follows:

Call	*Function*
plot(0, *term*)	causes further *plot* output to be piped into *tplot* with an argument of −T*term*. *Term* may be up to 40 characters in length.
plot(1)	"erases" the plotter.
plot(2, *string*)	labels the current point with *string*.
plot(3, *x1, y1, x2, y2*)	draws the line between (*x1,y1*) and (*x2,y2*).
plot(4, *x, y, r*)	draws a circle with center (*x,y*) and radius *r*.
plot(5, *x1, y1, x2, y2, x3, y3*)	draws an arc (counterclockwise) with center (*x1,y1*) and endpoints (*x2,y2*) and (*x3,y3*).
plot(6)	is not implemented.
plot(7, *x, y*)	makes the current point (*x,y*).
plot(8, *x, y*)	draws a line from the current point to (*x,y*).
plot(9, *x, y*)	draws a point at (*x,y*).
plot(10, *string*)	sets the line mode to *string*.
plot(11, *x1, y1, x2, y2*)	makes (*x1,y1*) the lower left corner of the plotting area and (*x2,y2*) the upper right corner of the plotting area.
plot(12, *x1, y1, x2, y2*)	causes subsequent x (y) coordinates to be multiplied by *x1* (*y1*) and then added to *x2* (*y2*) before they are plotted. The initial scaling is **plot(12, 1.0, 1.0, 0.0, 0.0)**.

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot*.

Each statement executed from the keyboard re-invokes *tplot*, making the results unpredictable if a complete picture is not done in a single operation. Plotting should thus be done either in a function or a complete program, so all the output can be directed to *tplot* in a single stream.

last() in immediate mode, *last* returns the most recently computed value.

EXAMPLES

Using *bs* as a calculator:

```
$ bs
#    Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...
#    Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4  bal = bal + bal*int
bal - 1000
346.855007
...
exit
```

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while  ?(str = read)
        ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
#    Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

#    Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls)  ...
...
# be sure to close (wait for) these:
close("ls")
```

```
close("pr")
```

WARNINGS

The graphics mode is nearly useless unless the *tplot* command is provided on your system.

Bs is not extremely tolerant of some errors. Mistyping a *fun* declaration is painful, as a new definition cannot be made without doing a *clear*. Starting using the *edit* command is the best solution in this case.

SEE ALSO

ed(1), sh(1), access(2), printf(3S), stdio(3S), lang(5), regexp(5).
See Section (3M) for a further description of the mathematical functions (*pow* on *exp*(3M) is used for exponentiation); *bs* uses the Standard Input/Output package.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the characters matched by character class expressions in regular expressions.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *bs* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

NAME

cal – print calendar

SYNOPSIS

cal [[*month*] *year*]

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

EXAMPLES

The command:

cal 9 1850

prints the calendar for September, 1850 on the screen as follows:

September 1850

S	M	Tu	W	Th	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

WARNINGS

The year is always considered to start in January even though this is historically naive. Beware that "cal 83" refers to the early Christian era, not the 20th century.

STANDARDS CONFORMANCE

cal: SVID2, XPG2, XPG3

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. On weekends "tomorrow" extends through Monday.

When an argument **-** is present, *calendar* searches for a file **calendar** in each user's home directory and sends any positive results to the user by *mail*(1). Normally this is done daily in the early morning hours under the control of *cron*(1M). When invoked by *cron*(1M), *calendar* reads the first line in the **calendar** file to determine the user's environment.

Language-dependent information such as spelling and date format (described below) are determined by the user-specified **LANG** statement in the **calendar** file. This statement should be of the form **LANG**=*language* where *language* is a valid language name (see *lang*(5)). If this line is not in the **calendar** file, the action described in the EXTERNAL INFLUENCES Environment Variable section is taken.

Calendar is concerned with two fields, month and day. A month field can be expressed in three different formats: a string representing the name of the month (either fully spelled out or abbreviated), a numeric month, or an asterisk (representing any month). If the month is expressed as a string representing the name of the month, the first character can be either upper or lowercase; other characters must be lowercase. The spelling of a month name should match the string returned by calling *langinfo*(3C). The day field is a numeric value for the day of the month.

Month-Day Formats

If the month field is a string, it can be followed by zero or more blanks. If the month field is numeric, it must be followed by either a slash (/) or a hyphen (-). If the month field is an asterisk (*), it must be followed by a slash (/). The day field can be followed immediately by a blank or non-digit character.

Day-Month Formats

The day field is expressed as a numeral. What follows the day field is determined by the format of the month. If the month field is a string, the day field must be followed by zero or one dot (.) followed by zero or more blanks. If the month field is a numeral, the day field must be followed by either a slash (/) or a hyphen (-). If the month field is an asterisk, the day field must be followed by a slash (/).

EXAMPLES

The following **calendar** file includes several formats recognized by *calendar*:

```
LANG=american
Friday, May 29th: group coffee meeting
meeting with Boss on June 3.
3/30/87 - quarter end review
4-26 Management council meeting at 1:00 pm
It is first of the month ( */1 ); status report due.
```

In the following **calendar** file, dates are expressed according to European English usage:

```
LANG=english
On 20 Jan. code review
Jim's birthday is on the 3. February
30/3/87 - quarter end review
26-4 Management council meeting at 1:00 pm
```

It is first of the month (1/*); status report due.

WARNINGS

To get reminder service, either your calendar must be public information or you must run *calendar* from your personal *crontab* file, independent of any *calendar* − run systemwide. Note that if you run *calendar* yourself, the calendar file need not reside in your home directory.
Calendar's extended idea of "tomorrow" does not account for holidays.

AUTHOR

Calendar was developed by AT&T and HP.

FILES

calendar
/tmp/cal*
/usr/lib/calprog to figure out today's and tomorrow's dates
/usr/lib/crontab
/etc/passwd

SEE ALSO

cron(1M), langinfo(3C), mail(1), environ(5).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings when no LANG statement is specified in the **calendar** file.

LANG determines the language in which messages are displayed.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *calendar* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

calendar: SVID2, XPG2, XPG3

NAME

cat – concatenate, copy, and print files

SYNOPSIS

cat [**–u**] [**–s**] [**–v** [**–t**] [**–e**]] file ...

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output. Thus:

```
cat file
```

prints the file, and:

```
cat file1 file2 >file3
```

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument – is encountered, *cat* reads from the standard input file, enabling you to combine standard input with other files.

The options are:

–u causes output to be unbuffered (character-by-character); normally, output is buffered.

–s makes *cat* silent about non-existent files, identical input and output, and write errors. Normally, no input file may be the same as the output file unless it is a special file. (The 4.2BSD **cat -s** feature is provided by *ssp*(1).)

–v causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. Control characters are printed ^X (control-X); the DEL character (octal 0177) is printed ^**?**. All other non-printing characters are printed as **M**–*x*, where *x* is the character specified by the seven low order bits. This option is influenced by the LANG environment variable and its corresponding code set.

–t when used with the **–v** option, **–t** causes tabs to be printed as ^**I**'s.

–e when used with the **–v** option, causes a **$** character to be printed at the end of each line (prior to the new-line).

The **–t** and **–e** options are ignored if the **–v** option is not specified.

RETURN VALUE

Exit values are:

0 Successful completion.
>0 Error condition occured.

SEE ALSO

cp(1), pg(1), pr(1), rmnl(1), ssp(1).

WARNING

Command formats such as

```
cat file1 file2 >file1
```

overwrites the data in *file1* before the concatenation begins. Therefore, take care when using shell special characters.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text and filenames as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cat* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

cat: SVID2, XPG2, XPG3

NAME

cb – C program beautifier, formatter

SYNOPSIS

cb [**–s**] [**–j**] [**–l** leng] [file ...]

DESCRIPTION

Cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. Under default options, *cb* preserves all user new-lines. Under the **–s** flag *cb* converts the code to the canonical style of Kernighan and Ritchie in *The C Programming Language*. The **–j** flag causes split lines to be put back together. The **–l** flag causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).

The C Programming Language by B. W. Kernighan and D. M. Ritchie.

BUGS

Hidden punctuation in preprocessor statements causes indentation errors.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of comments and string literals as single and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting *cb* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

cc − C compiler

SYNOPSIS

cc [*options*] *files*

DESCRIPTION

Cc is the HP-UX C compiler. It accepts several types of arguments as *files*:

- Arguments whose names end with **.c** are understood to be C source files. Each is compiled and the resulting object file is left in a file having the corresponding name, but suffixed with **.o** instead of **.c**. However, if a single C file is compiled and linked all in one step, the **.o** file is deleted.
- Similarly, arguments whose names end with **.s** are understood to be assembly source files and are assembled, producing a **.o** file for each **.s** file.
- Arguments whose names end with **.i** are taken to be the output of *cpp*(1). (See the **−P** option below.) They are compiled without again invoking *cpp*(1). Each object file is left in a file having the corresponding name, but suffixed **.o** instead of **.i**.
- Arguments of the form **−l***x* that cause the linker to search the library **lib***x***.a** in an attempt to resolve currently unresolved external references. Because a library is searched when its name is encountered, placement of a **−l** is significant. If a file contains an unresolved external reference, the library containing the definition must be placed *after* the file on the command line. See *ld*(1) for further details.
- All other arguments, such as those whose names end with **.o** or **.a**, are taken to be relocatable object files that are to be included in the link operation.

Arguments and options can be passed to the compiler through the **CCOPTS** environment variable as well as on the command line. The compiler picks up the value of **CCOPTS** and places its contents before any arguments on the command line. For example (in *sh*(1) notation),

```
CCOPTS=−v
export CCOPTS
cc −g prog.c
```

is equivalent to

```
cc −v −g prog.c
```

When set, the TMPDIR environment variable specifies a directory to be used for temporary files, overriding the default directories **/tmp** and **/usr/tmp**.

Options

The following options are recognized by *cc*:

−A*mode* Specify the compilation standard to be used by the compiler. The *mode* can be one of the following letters:

- **c** Compile in a mode compatibile with HP-UX releases prior to 7.0. (See *The C Programming Language*, First Edition by Kernighan and Ritchie). This option is the default for release 7.0 of HPUX. However the default may change in future releases.
- **a** Compile under ANSI mode (December 7, 1988 Draft proposed ANSI C standard.)

−c Suppress the link edit phase of the compilation, and force an object (**.o**) file to be produced for each **.c** file even if only one program is compiled. Object files produced from C programs must be linked before being executed.

−C Prevent the preprocessor from stripping C-style comments. See *cpp*(1) for details.

−D*name=def*
−D*name* Define *name* to the preprocessor, as if by '#define'. See *cpp*(1) for details.

−E Run only *cpp*(1) on the named C or assembly files, and send the result to the standard output.

−g Cause the compiler to generate additional information needed by the symbolic debugger. This option cancels the effect of any specified optimization options.

−G Prepare object files for profiling with *gprof* (see *gprof*(1)).

−I*dir* Change the algorithm used by the preprocessor for finding include files to also search in directory *dir*. See *cpp*(1) for details.

−l*x* Refer to item (4) at the beginning of the DESCRIPTION section.

−L *dir* Change the algorithm used by the linker to search for **lib***x***.a**. The **−L** option causes *cc* to search in *dir* before searching in the default locations. This option is effective only if it precedes the **−l** option on the command line. See *ld*(1) for details.

−n Cause the output file from the linker to be marked as shareable. For details and system defaults, see *ld*(1).

−N Cause the output file from the linker to be marked as unshareable. For details and system defaults, see *ld*(1).

−o*outfile* Name the output file from the linker *outfile*. The default name is **a.out**.

−O Invoke the optimizer.

−p Arrange for the compiler to produce code that counts the number of times each routine is called. Also, if link editing takes place, replace the standard startoff routine by one that automatically calls *monitor*(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

−P Run only *cpp*(1) on the named C files and leave the result on corresponding files suffixed **.i**. The **−P** option is also passed along to *cpp*(1).

−q Cause the output file from the linker to be marked as demand loadable. For details and system defaults, see *ld*(1).

−Q Cause the output file from the linker to be marked as not demand loadable. For details and system defaults, see *ld*(1).

−s Cause the output of the linker to be stripped of symbol table information. The use of this option prevents the use of a symbolic debugger on the resulting program. See *ld*(1) for more details.

−S Compile the named C files, and leave the assembly language output on corresponding files suffixed **.s**.

−tx,name Substitute or insert subprocess *x* with *name* where *x* is one or more of a set of identifiers indicating the subprocess(es). This option works in two modes: 1) if *x* is a single identifier, *name* represents the full path name of the new subprocess; 2) if *x* is a set of identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path names of the new subprocesses.

The *x* can take one or more of the values:

p Preprocessor (standard suffix is **cpp**)
c Compiler body (standard suffix is **ccom**)
0 Same as **c**
a Assembler (standard suffix is **as**)
l Linker (standard suffix is **ld**)

−U*name*
Remove any initial definition of *name* in the preprocessor. See *cpp*(1) for details.

−v Enable verbose mode, which produces a step-by-step description of the compilation process on the standard error.

−w Suppress warning messages.

−W *x,arg1*[*,arg2...*]
Hand off the argument[s] *argi* to pass *x*, where *x* can assume one of the values listed under the **−t** option as well as **d** (driver program). The **−W** option specification allows additional, implementation-specific options to be recognized by the compiler driver. For example, on the Series 300,

−W c,−M

causes the compiler to generate calls to math library routines instead of inline code for the MC6881 math coprocessor. For some options, a shorthand notation for this mechanism can be used by placing "+" in front of the option name as in

+M

which is equivalent to the previous option example. Some other commonly used subprocess options can also be abbreviated in a similar fashion. Note that for simplicity, this shorthand must be applied to each option individually. Options that can be abbreviated using "+" are implementation dependent, and are listed under DEPENDENCIES.

−y Generate additional information needed by static analysis tools, and ensure that the program is linked as required for static analysis. This option is incompatible with optimization.

−Y Enable support of 16-bit characters inside string literals and comments. Note that 8-bit parsing is always supported. See *hpnls*(5) for more details on International Support.

−z Do not bind anything to address zero. This option allows runtime detection of null pointers. See the note on *pointers* below.

−Z Allow dereferencing of null pointers. See the note on *pointers* below.

Any other options encountered generate a warning to standard error.

Other arguments are taken to be C-compatible object programs, typically produced by an earlier *cc* run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the name **a.out**.

The first edition of "The C Programming Language", by Kernighan and Ritchie, and the various addenda to it, are intentionally ambiguous in some areas. HP-UX specifies some of these below for compatability mode compilations.

char The **char** type is treated as signed by default. It may be declared **unsigned**.

pointers Accessing the object of a NULL (zero) pointer is technically illegal (see Kernighan and Ritchie), but many systems have permitted it in the past. The following is provided to maximize portability of code. If the hardware is able to

return zero for reads of location zero (when accessing at least 8- and 16-bit quantities), it must do so unless the **−z** flag is present. The **−z** flag requests that SIGSEGV be generated if an access to location zero is attempted. Writes of location zero may be detected as errors even if reads are not. If the hardware cannot assure that location zero acts as if it was initialized to zero or is locked at zero, the hardware should act as if the **−z** flag is always set.

identifiers — Identifiers are significant up to 255 characters.

types — Certain programs require that a type be a specific number of bits wide. It can be assumed that an **int** can hold at least as much information as a **short**, and that a **long** can hold at least as much information as an **int**. Additionally, either an **int** or a **long** can hold a pointer.

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor.

If any errors occur before *cc* is completed, a non-zero value is returned. Otherwise, zero is returned.

EXAMPLES

The following compiles the C file **prog.c**, to create a **prog.o** file, and then invoke the link editor *ld*(1) to link **prog.o** and **procedure.o** with all the C startup routines in **/lib/crt0.o** and library routines from the C library **libc.a**. The resulting executable program is output in **prog**:

```
cc prog.c procedure.o −o prog
```

WARNINGS

Options not recognized by *cc* are not passed on to the link editor. The option **−W l,***arg* can be used to pass any such option to the link editor.

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value are to explicitly call *exit*(2) or to leave the function **main()** with a '**return** *expression;*' construct.

DEPENDENCIES

Series 300

The **−A**, **−z**, and **−y** options are not supported.

The default is to allow null pointer dereferencing, hence using **−Z** has no effect.

The compiler supports the following options, which may also be passed to it from *cc* using the **−W c** option.

+bfpa — Cause the compiler to generate code that uses the HP 98248A floating point accelerator card, if it is installed at run-time. If the card is not installed, floating point operations are done on the MC68881 math coprocessor.

+ffpa — Cause the compiler to generate code for the HP 98248A floating point accelerator card. This code does not run unless the card is installed.

+M — Cause the compiler not to generate inline code for the MC68881 math coprocessor. Library routines are referenced for *matherr* capability.

+N<*secondary*><*n*>
Adjust the size of internal compiler tables. The compiler uses fixed size arrays for certain internal tables. *Secondary* is one of the letters from the set {**abdepstw**}, and *n* is an integer value. *Secondary* and *n* are *not* optional. The table sizes can be re-specified using one of the secondary letters and the number *n* as follows:

a Maximum size of the asciz table. Default is 10000 table entries.

b Maximum size of the bc table. This table saves break and continue labels within loops and switch statements. Default is 100 table entries.

d Initial size of the dimtab table. This table maintains information about the definitions of all structures, unions, and arrays. Default is 1000 table entries.

e Maximum number of nodes per statement. Default is 350 table entries.

p Maximum size of the parameter stack. Default is 150 table entries.

s Maximum size of the symbol table. Default is 2000 table entries.

t Maximum size of the tasciz table. Default is 20000 table entries.

w Maximum size of the switch table stack. Default is 250 table entries.

+O*opt* Invoke optimizations selected by *opt*.
If *opt* is **1**, only level 1 optimizations are handled. If *opt* is **2**, all optimizations are performed. The **−O** option is equivalent to **+O2**. If *opt* is **V**, optimization level 2 is selected, but all global variables and objects dereferenced by global pointers are treated as if they were declared with the keyword "volatile," meaning that references to the object cannot be optimized away.

−t*x,name* Specify additional subprocess identifiers.

0 First pass of the compiler with level 2 optimization. It is not the same as subprocess **c** (standard suffix is **cpass1**).

1 Second pass of the compiler with level 2 optimization (standard suffix is **cpass2**).

g Level 2 global optimizer. (standard suffix is **c1**)

2 Peephole optimizer (standard suffix is **c2**)

−W c,−YE Cause source code lines to be printed on the assembly (**.s**) file as assembly comments, thus showing the correspondence between C source and the resulting assembly code. This option is incompatible with optimization.

Series 800

The default is to allow null pointer dereferencing, hence using **−Z** has no effect.

Nonsharable, executable files generated with the **−N** option cannot be executed via *exec*(2). For details and system defaults, see *ld*(1).

The compiler supports the following additional options. The *+opt1* notation can be used as a shorthand notation for some **−W** options.

+a *or* **−W d,−a** When processing files which have been written in assembly language, does not assemble with the prefix file which sets up the space and subspace structure required by the linker. Files assembled with this option cannot be linked unless they contain the equivalent information.

+e *or* **−W c,−e** Enables HP value added features while compiling in ANSI C mode, **−Aa.** This option is ignored with **−Ac** since these features are already

provided. Features enabled:

- long pointers
- missing parameters on intrinsic calls

+L *or* **−W c,−L** Enable the listing facility and any listing pragmas. A straight listing prints:

- A header on the top of each page
- Line numbers
- The nesting level of each statement
- The postprocessed source file with expanded macros, included files, and no user comments (unless the **−C** option is used).

If the **−Aa** option is used to compile under ANSI C, the listing will show the original source file rather than the postprocessed source file.

+Lp Print a listing as described above, but show the postprocessed source file even if one of the ANSI compilation levels is selected. This option is ineffective if the **−y** option is used.

+m *or* **−W c,−m**
Cause the identifier maps to be printed. First, all global identifiers are listed, then all the other identifiers are listed by function at the end of the listing. For struct and union members, the address column contains B@b, where B is the byte offset and b is the bit offset. Both B and b are in hexadecimal.

+o *or* **−W c,−o** Cause the code offsets to be printed in hexadecimal; they are grouped by function at the end of the listing.

+O*opt* *or* **−W c,−O***opt*
Invoke optimizations selected by *opt*. If *opt* is **1**, only level 1 optimizations are handled. If *opt* is **2**, all optimizations are performed. The option **+O2** is the same as **−O**. If *opt* is **V** all memory references are treated as if they were declared with the keyword "volatile," meaning that references to the object cannot be optimized away.

+r *or* **−W c,-r** Inhibits the automatic promotion of float to double when evaluating expressions and passing arguments. This option is ignored and a warning produced if the **−Aa** option is in effect.

+R*num* *or* **−W c,−R***num*
Allow only the first num 'register' variables to actually have the 'register' class. Use this option when the register allocator issues an "out of general registers" message.

+w*n* *or* **−W c,−wn**
Specify the level of the warning messages. The value of *n* can be one of the following values:

1 All warnings are issued.
2 Only warnings indicating that code generation might be affected are issued. Equivalent to the compiler default without any **w** opts.
3 No warnings are issued. Equivalent to the **−w** option.

FILES

file.c	input file
file.o	object file
a.out	linked output

/tmp/ctm*	default temporary files
/usr/tmp/ctm*	default temporary files
/lib/ccom	compiler
/lib/cpp	preprocessor
/lib/cpp.ansi	preprocessor for ANSI C
/bin/as	assembler, *as*(1)
/bin/ld	link editor, *ld*(1)
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling via *prof*(1)
/lib/gcrt0.o	startoff for profiling via *gprof*(1)
/lib/libc.a	standard C library, see *HP-UX Reference* Section (3).
/lib/libp/libc.a	C library for profiled programs
/usr/include	standard directory for **#include** files

Series 300

/lib/cpass1	pass 1 of the optimizing compiler
/lib/cpass2	pass 2 of the optimizing compiler
/lib/c1	global optimizer
/lib/c2	peephole optimizer
/bin/as20	assembler

Series 800

/usr/lib/nls/$LANG/cc.cat C Compiler message catalog

SEE ALSO

adb(1), as(1), cdb(1), cpp(1), gprof(1), ld(1), prof(1), exit(2), monitor(3C), matherr(3M).

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978.

Draft Proposed American National Standard for Information Systems – Programming language C, December 7, 1988, Doc. No. X3J11/88-090

EXTERNAL INFLUENCES

Environment Variables

When the **−Y** option is invoked, LC_CTYPE determines the interpretation of string literals and comments as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cc* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

cc: SVID2, XPG2, XPG3

NAME

cd – change working directory

SYNOPSIS

cd [*directory*]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **$HOME** is used as the new working directory. If *directory* specifies a complete path starting with /, ., .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable. *Cd* must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and is internal to the shell.

EXAMPLES

To change the working directory to a directory at a lower level, type:

cd *directoryname*

To change the working directory to a directory at the same level, type:

cd **../***directoryname*

To change from any working directory to the home directory, type:

cd

VARIABLES

HOME	default working directory
CDPATH	directories to search for directory.

SEE ALSO

pwd(1), sh(1), chdir(2).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

cd: SVID2, XPG2, XPG3

NAME

cdb, fdb, pdb – C, FORTRAN, Pascal symbolic debugger

SYNOPSIS

cdb [**–d** *dir*] [**–r** *file*] [**–p** *file*] [**–P** *process ID*] [**–L**] [**–i** *file*] [**–o** *file*] [**–e** *file*] [**–S** *num*] [*objectfile* [*corefile*]]

fdb [*cdb options*]

pdb [*cdb options*]

TABLE OF CONTENTS

DESCRIPTION

Cdb, *fdb*, and *pdb* are alternate names for a source level debugger for C, HP FORTRAN, and HP Pascal programs. It provides a controlled environment for their execution.

Objectfile is an executable program file having zero or more of its component modules compiled with the **–g** option of *cc*(1), *fc*(1), or *pc*(1). The support module **/usr/lib/end.o** must be included as the last object file in the list of those linked, except for libraries included with the **–l** option to *ld*(1) (done automatically with **-g** option). The default for *objectfile* is **a.out**.

Corefile is a core image from a failed execution of *objectfile*. The default *corefile* is **core**.

Options

−d *dir* Specify *dir* as an alternate directory where source files are located. If a source file is not found in any alternate directory, the current directory is searched last.

−r *file* Specify a record *file* which is invoked immediately (for overwrite, not for append). See the section below entitled *Record and Playback Commands* for a description of this feature.

−p *file* Specify a playback *file* which is invoked immediately. See the section below entitled *Record and Playback Commands* for a description of this feature.

−P *process ID* Specify the process ID of an existing process that you wish to debug (see Adopting an Existing Process below).

−L Use the line-oriented interface even if the debugger can support the window-oriented interface on the terminal type specified by $TERM.

−i *file* Redirect standard input to the child process from the designated file.

−o *file* Redirect standard output from the child process to the designated file.

−e *file* Redirect standard error from the child process to the designated file.

−S *num* Set the size of the string cache to *num* bytes (default is 1024, which is also the minimum). The string cache holds data read from *objectfile*.

Only one *objectfile* and one *corefile* are allowed per debugging session. The program (*objectfile*) is not invoked as a child process until an appropriate job control command is given (see Job Control Commands below). The same program can be restarted many times as different child processes during a single debugging session.

At startup, *cdb* executes commands from the file **.cdbrc** (**.fdbrc** for FORTRAN and **.pdbrc** for Pascal) if it exists in the user's home directory as specified by the environment variable HOME.

This debugger is a complex, interactive tool whose many capabilities are often limited only by your imagination. However, the debugger is also only a "window" to the world of the program being debugged and the system on which it runs. If something puzzling happens, consult a manual that describes the program or the system to better understand the behavior.

USER INTERFACE

Environment

Cdb reads the environment variable TERM to determine the applicable terminal type and user interface. The locale category LC_CTYPE, which defaults to the 'C' locale, is used to determine the character-set and semantics to be used when reading or writing character and string data.

Window Oriented Interface

This user interface is only supported on HP terminals with memory-lock. The top of the screen is a "window" into the current source file and the bottom of the screen is for the debugger and user program input and output. Separating the two areas is a line (in inverse video) indicating the current file, procedure, and line number. Within the source file window, a ">" points to the current location (which might not be the location at which the user program is currently stopped).

To determine the number of lines and columns, the debugger looks first for the environment variables LINES and COLUMNS. If these environment variables are not found, the debugger uses the line and column information from the **/usr/lib/terminfo** file for the terminal type found in the environment variable TERM.

Note that a $TERM setting of 'hp' implies that the terminal supports memory-lock, which is not true of some HP terminals (such as the HP2621). For this and other cases, the debugger attempts to provide a window-oriented interface even if the terminal itself does not provide the

necessary capabilities. For such situations, the debugger can be forced into the line oriented interface mode by invoking it with the **-L** command line option.

Line Oriented Interface

This is the user interface currently supported on HP terminals without memory-lock and on all non-HP terminals. This interface does not use windows. The source file is displayed one line at a time, but some commands compensate by supplying more information than with the "window" interface.

CONVENTIONS

The debugger remembers the current file, procedure, line, and data location that you have been viewing (although not necessarily executing) most recently. Many commands use these current locations as defaults, and set them as a result. Keep this in mind when deciding what a command does in any particular situation.

For example, if you stop in procedure "abc", then view procedure "def", then ask for the value of local variable "xyz", the debugger assumes that the variable belongs to procedure "def".

Notational Conventions

Most commands are of the form "[*modifier*] *command* [*options*]". Numeric modifiers before and after commands can be any numeric expression. They need not be just simple numbers. A blank is required before any numeric *option*. Multiple commands on one line must be separated by "**;**".

These are common modifiers and other special notations:

(A | B | C) Any one of A or B or C is required.

[A | B | C] Any one of A or B or C is optional.

commands A series of debugger commands, separated by "**;**", entered on the command line or saved with a breakpoint or assertion. Semicolons are ignored (as commands) so they can be freely used as command separators. Commands can be grouped with "{}" for the "**a**", "**b**", "**if**", and "**!**" commands. In all other cases commands inside "{}" are ignored.

count The number of repetitions specified for a command.

depth A stack depth, as printed by the "**t**" command. The top procedure is at a *depth* of zero. A negative *depth* acts like a *depth* of zero. Stack depth usually means "exactly at the specified depth", not "the first instance at or above the specified depth".

expr Any expression, but with limitations stated below.

file A file name.

format A style for printing data. Used with Data Viewing Commands.

line A *number* that refers to a particular line in a file.

location A particular *line* in a file and its corresponding address in the user's program, if executable code exists for that line. The *location* has the following general forms:

line
file [: *line*]
proc [: *proc* [. . .]] [: (*line* | *#label*)]

number A specific, constant number (e.g. "9", not "4+5"). Floating point (real) numbers can be used anywhere a constant is allowed.

proc A procedure, function, or subroutine name.

var A variable name.

Variable Name Conventions

Variables are referenced exactly as they are named in your source file(s). Case sensitivity is controlled by the "**Z**" command. Be careful with one letter variable names, since they can be confused with commands. If an expression begins with a variable that might be mistaken for a command, just enclose the expression in "**()**" (e.g. "(k)"), or eliminate any white space between the variable and the first operator (use "k= 9" instead of "k = 9").

If you are interested in the value of some variable *var*, there are a number of ways of getting it, depending on where and what it is:

var Search the stack for the most recent instance of the current procedure. If found, see if *var* is a parameter or local variable of that procedure. If not, search for a global variable named either *var* or *_var*, in that order.

proc.var Search the stack for the most recent instance of *proc*. If found, see if it has a parameter or local variable named *var*, as before.

proc.depth.var Use the instance of *proc* that is at depth *depth* (exactly), instead of the most recent instance. This is very useful for debugging recursive procedures where there are multiple instances on the stack.

:var Search for a global (not local) variable named either *var* or *_var*, in that order.

. *Dot* is shorthand for the last thing you viewed. It has the same size it did when you last viewed it. For example, if you look at a **long** as a **char**, then "." is considered to be one byte long. This is useful for treating things in unconventional ways, like changing the second highest byte of a **long** without changing the rest of the **long**. *Dot* can be treated like any other variable.

NOTE: "." is the *name* of this magic location. If you use it, it is dereferenced like any other name. If you want the *address* of something that is, say, 30 bytes farther on in memory, do not say ".+30". That would take the contents of *dot* and add 30 to it. Instead, say "&.+30", which adds 30 to the *address* of *dot*.

Special variables are names for things that are not normally directly accessible. Special variables include:

$*var* The debugger has room in its own address space for several user-created special variables. They are all of type **long**, and they do not take on the type of any expression assigned to them. Names are defined when they are first seen. For example, saying "$xyz = 3*4" creates special symbol "$xyz", and assigns to it the value 12. Special variables can be used just like any other variables. Names are limited to 100 characters.

$pc, **$fp**, **$sp**, **$d0**, etc.
These are the names of the program counter, the frame pointer, the stack pointer, the registers, etc. To find out which names are available on your system, use the "**lr**" (list registers) command. All registers act as type **integer**.

$result This is used to reference the return value from the last command-line procedure call. Where possible, it takes on the type of the procedure. **$short** and **$long** are available as alternate ways of looking at **$result**.

$signal This lets you see and modify the current child process signal number.

$lang This lets you see and modify the current language. The current language determines the operators that can be used in expressions, and the format in

which variables are displayed. Values that can be assigned to $lang are "C", "FORTRAN", "Pascal" and "default" ("default" means use whatever language the current procedure is written in).

$print Alters the behavior of the "print" command when printing character data. Values that can be assigned are "ascii", "native", and "raw". Default is "ascii". "Ascii" causes all non-ASCII characters to be displayed as octal-escapes. "Native" causes unprintable characters, as determined by the locale category (environemnt variable) LC_CTYPE, to be displayed as octal-escapes. "Raw" causes all bytes to be output unaltered. This also affects the default display format for character types (see Display Formats).

$line This lets you see and modify the current source line number, which is also settable with a number of different commands.

$malloc This lets you see the current amount of memory (bytes) allocated at run-time for use by the debugger itself.

$cBad This lets you see and modify the number of machine instructions the debugger will step while in a non-debuggable procedure before setting an up-level breakpoint and free-running to it. Setting it to a small value can improve debugger performance at the risk of taking off free-running after missing the up-level break for some reason.

$pagelines This lets you set the number of lines per "page" of debugger output. The prompt "--More--" occurs between pages. Values of zero or less turn off paging.

$fpa If this is set to a non-zero value, any sequence of machine instructions which effectively constitute a single floating point accelerator instruction will be treated as a single instruction for machine level single-stepping and display.

$fpa_reg If **$fpa** is set to a non-zero value, **$fpa_reg** indicates which address register is used in floating point accelerator instruction sequences. A 0 corresponds to register a0, 1 to a1, etc. The default value is 2.

To see all the special variables, including the predefined ones, use the "**ls**" (list specials) command.

You can also look up code addresses with

proc#*line*

which searches for the given procedure name and line number (which must be an executable line within *proc*) and uses the code address of that line. Just referring to a procedure *proc* by name uses the code address of the entry point to that procedure.

Expression Conventions

Every expression has a value, even simple assignment statements, as in C. "Naked" expression values (those which aren't command modifiers) are always printed unless the next token is "**;**" (command separator) or "}" (command block terminator). Thus breakpoint and assertion commands are normally silent. To force an expression result to be printed, follow the expression with "**/n**" (print in normal format).

Integer constants might begin with "**0**" for octal or "**0x**" or "**0X**" for hexadecimal (the forms are equivalent). If followed immediately by "**l**" or "**L**", they are forced to be of type **long**. Likewise, "**u**" and "**U**" force the type to **unsigned**. "**ul**" or "**UL**" corresponds to **unsigned long**. If no suffix is used, the smallest type in which the value will fit is used.

Floating point constants must be of the form *digits.digits*[**e**|**E**|**d**|**D**|**L**|**l**[+|-]*digits*][**f**|**F**], for example, "1.0", "3.14e8f", or "26.62D-31". One or more leading digits is required to avoid

confusion with "." (*dot*). A decimal point and one or more following digits is required to avoid confusion for some command formats. If the exponent doesn't exactly fit the pattern shown, it is not taken as part of the number, but as separate token(s). The "**d**" and "**D**" exponent forms are allowed for compatibility with FORTRAN. The "**l**" and "**L**" exponent forms are allowed for compatibility with Pascal.

In the absence of a suffix character, the constant is assumed to be of type "**double**" (8-byte IEEE real). The suffixes "**f**" and "**F**" cause the value to be evaluated as type "**float**" (4-byte IEEE real). Unless a direct assignment is made, float types are converted to type double before the expression is evaluated.

Character constants must be entered in ' and are treated as **integer**s. String constants must be entered in " and are treated like "**char ***" (i.e. pointer to **char**). Character and string constants can contain the standard backslashed escapes understood by the C compiler and the *echo*(1) command, including "**\a**", "**\b**", "**\f**", "**\n**", "**\r**", "**\t**", "**\?**", "****", "**\'**", "*****nnn*", and "**\x***nnn...*". In the case of hex-escapes, the longest possible value is evaluated and then truncated to size of the destination type (1 byte). "\<newline>" is not supported, neither in quotes nor at the end of a command line.

Expressions are composed of any combination of variables, constants, and C operators. If the debugger is invoked as *cdb*, the C operator "**sizeof**" is also available. If the debugger is invoked as *fdb*, FORTRAN operators are also available and FORTRAN meanings take precedence where there is a conflict. The same is true for Pascal if the debugger is invoked as *pdb*. The user may create a composite program built from two or more of the supported languages. The **$lang** variable may be set to "default" so that the debugger recognizes the language for the code currently being debugged.

If there is no active child process and no *corefile*, you can only evaluate expressions containing constants.

Expressions approximately follow the C rules of promotion, e.g. **char**, **short**, and **int** become **long**, and **float** becomes **double**. If either operand is a **double**, floating math is used. If either operand is **unsigned**, unsigned math is used. Otherwise, normal (integer) math is used. Results are then cast to proper destination types for assignments.

If a floating point number is used with an operator that doesn't normally permit it, the number is cast to **long** and used that way. For example, the C binary operator "~" (bit invert) applied to the constant "3.14159" is the same as "~3".

Note that "=" means "assign" except for Pascal; to test for equality, use "==" or "**.EQ.**" for FORTRAN. In Pascal, "=" is a comparison operator; use ":=" for assignments. For example, if you invoke the debugger as **cdb**, then set "$lang = Pascal", you must say "$lang := C" to return to C.

Use "//" for division, instead of "/", to distinguish from display formatting (see *Data Viewing Commands*).

The special unary operator "**$in**" (not to be confused with debugger local variables) evaluates to 1 (true) if the operand is an address inside a debuggable procedure and **$pc** (the current child process program location) is also in that procedure, else it is 0 (false). For example, "**$in** main" is true if the child process is stopped in main().

If the first expression on a line begins with "+" or "−", use "**()**" around it to distinguish from the "+" and "−" commands (see *Data Viewing Commands*). Parentheses might also be needed to distinguish an expression from a command it modifies.

You can attempt to dereference any constant, variable, or expression result using the C "*" operator. If the address is invalid, an error is given.

Type casting is allowed. For simple types, the syntax is identical to C. For example:

```
(short) size
(double *) mass_ptr
```

These casts are limited to **char**, **short**, **long**, **int**, **unsigned**, **float**, and **double**, appropriate combinations of these keywords, and single level pointer types. Also supported are structure and union pointer type dereferences. For example:

```
bar_ptr = &bar
(struct foo) &bar
(struct foo) bar_ptr
```

Both of these casts treat "bar" as a struct of type "foo" during printing. Structure and union casts may only include the keyword "struct" or "union" and an appropriate tag. No pointers ("*") are allowed. The argument of the cast is simply treated as an address.

Whenever an array variable is referenced without giving all its subscripts, the result is the address of the lowest element referenced. For example, consider an array declared as "x[5][6][7]" in C, "x(5,6,7)" in FORTRAN, or "x[1..5,2..6,3..7]" in Pascal. Referencing it simply as "x" is the same as just "x" in C, the address of "x(1,1,1)" in FORTRAN, or the address of "x[1,2,3]" in Pascal. Referencing it as "x[4]" is the same as "& (x[4][0][0])" in C, the address of "x(1,1,4)" in FORTRAN, or the address of "x[4,2,3]" in Pascal.

If a not-fully-qualified array reference appears on the left side of an assignment, the value of the right-hand expression is stored into the element at the address specified.

Array indices are not checked against declared bounds.

String constants are stored in a magic buffer in the file **/usr/lib/end.o**, which you link with your program. The debugger starts storing strings at the beginning of this buffer, and moves along as more assignments are made. If the debugger reaches the end of the buffer, it goes back and reuses it from the beginning. In general this doesn't cause any problems. However, if you use very long strings, or if you assign a string constant to a global pointer, problems could arise.

Procedure Call Conventions

Procedures can be invoked from the command line, even within expressions. For example:

```
xyz = $abc * (3 + def (ghi - 1, jkl, "Hi Mom"))
```

calls procedure "def" when its value is needed in the expression.

Any breakpoints encountered during command line procedure invocation are handled as usual. However, the debugger has only one active command line at a time. If it stops in a called procedure for any reason, the remainder (if any) of the old command line is discarded with notice given.

If you attempt to call a procedure when there is no active child process, one is started for you as if you gave a single-step command first. Unfortunately, this means that the data in *corefile* (if any) might disappear or be reinitialized.

If you send signal SIGINT (e.g., hit the BREAK key) while in a called procedure, the debugger aborts the procedure call and returns to the previous stopping point (the start of the main program for a new process).

You can call any procedure that is in your *objectfile*, even if it is not debuggable (was not compiled with debug on). For example, assume that you reference "printf()" in your program, so the code for it is in your *objectfile*. Then you can enter on the command line:

```
printf ( "This works! %d %c\n", 9, '?');
```

If you wonder what procedures are available, do a list labels command ("**ll**").

COMMANDS

The debugger has various commands for viewing and manipulating the program being debugged. The command line editing and history features from *ksh*(1) are available during command input (see *ksh*(1)). The environment variables CDBEDIT, EDITOR, or VISUAL are checked (in that order) to determine which of the three available editing modes (vi, emacs, or gmacs) is used. The command history file is specified by the CDBHIST environment variable and its size is derived from HISTSIZE. If any of these environment variables is not set, the default is the same as with *ksh*(1) except that CDBHIST defaults to "$HOME/.cdbhist".

Window Mode Commands

These commands control what is displayed in the source window. The source window has three different modes. In source mode, the window is filled with source lines from the user program. In disassembly mode, the top five lines of the source window display one of three sets of registers (see the **gr** and **fr** commands), and the remainder of the window contains assembly language instructions. In split-screen mode, the top half of the window is source code, and the bottom half is the corresponding assembly instructions.

td Toggle disassembly mode. When in disassembly mode, the source window displays a set of registers (see **gr** and **fr**) and the code in assembly language. The assembly language display consists of the source line number, the address in hex, the address in the form of nearest label plus offset, and the assembly instruction.

ts Toggle split screen mode. When in split-screen mode, half the source window displays source code and half assembly instructions.

gr Display general registers when the debugger is in assembly (non-split screen) mode. When the value of a register changes, the register is highlighted until after the next command is executed. These registers can be modified by using the debugger special variables $d0 through $a7. When displaying the general registers or floating point registers, the line dividing the registers from the assembly code displays corresponding special registers. For registers containing flags, each flag is represented by a letter. A lowercase letter indicates that the flag is off, while uppercase means on.

fr Display floating point registers when the debugger is in assembly (non-split screen) mode. When the value of a register changes, that register is highlighted until after the next command. If more than one set of floating-point registers exist, you will be queried about which you want displayed.

+r Scroll the floating-point register display forward four lines.

-r Scroll the floating-point register display back four lines.

ws *size* Set the size of the source viewing window. It is normally set to fifteen lines for a twenty-four line terminal.

u Update the screen to reflect the current location. This command is best used as part of an assertion: the assertion **a u** will show step-by-step progress through the program.

U Clear and redraw the screen.

File Viewing Commands

These commands might change the current viewing position, but they do not affect the next statement to be executed in the child process, if any.

dir *directory* Add *directory* to the list of alternate source directories. This is the same as using the **-d** invocation option. The main procedure file must reside in the current directory or be specified with the **-d** option.

e	Show the current file, procedure, line number, and source line.
e *location*	View the source at the specified location.
[*depth*] **E**	Like "**e**", but sets viewing location to the current location in *proc* on the stack at *depth* (not necessarily the first executable line in the procedure). The default *depth* is zero (where program is currently stopped).
L	This is a synonym for **0E**.
line	View the source line number *line* in the current file.
[*line*] **p** [*count*]	View one (or *count*) lines starting at the current line (or line number *line*). With the line oriented interface, if multiple lines are printed, the current line is marked with a "=" in the leftmost column.
+[*lines*]	Move to *lines* (default one) lines after the current line.
−[*lines*]	Move to *lines* (default one) lines before the current line.
[*line*] **w** [*size*]	For the line oriented interface, print a window of text containing *size* (default 11) lines centered around the current line (or *line*). The target line is marked with a "=" in the leftmost column if multiple lines are printed.
[*line*] **W** [*size*]	Same as "**w**", but *size* defaults to 21 lines.
+**w** [*size*] +**W** [*size*]	View a window of text of given or default *size*, beginning at the end of the previous window if the previous command was a window command; otherwise at the current line.
−**w** [*size*] −**W** [*size*]	View a window of text of given or default *size*, ending at the beginning of previous window if the previous command was a window command; otherwise at the current line.
/[*string*]	Search forward through the current file for *string*, starting at the line after the current line.
?[*string*]	Search backward for *string*, starting with the line before the current line.

Searches wrap around the end or beginning of the file, respectively. If *string* is not specified, the previous one is used. Wild cards and regular expressions are not supported; *string* must be literal.

n	Repeat the previous "**/**" or "**?**" command using the same *string* as before.
N	The same as "**n**", but the search goes in the opposite direction from that specified by the previous "**/**" or "**?**" command.

Display Formats

A *format* is of the form "[*][*count*]*formchar*[*size*]". Display formats apply only to Data Viewing Commands, described below. next sub−section.

"*" means "use alternate address map".

Count is the number of times to apply the format style *formchar* (must be a *number*).

Size is the number of bytes to be formatted for each *count* (overrides the default *size* for the format style); it must be a positive decimal *number* (except short hand notations). Do not use *size* with *formchar* unless it makes sense!

For example, "abc/4x2" prints, starting at the location of "abc", four two-byte numbers in hexadecimal.

The formats which print numbers allow an uppercase character to be used instead for the same results as appending "**l**" (see below). For example, "**O**" prints in long octal. The following formats are available:

n	Print in the "normal" format, based on the type. Arrays of **char** and pointers to **char** are interpreted as strings, and structures are fully dumped.
(**d** | **D**)	Print in decimal (as **integer** or **long**).
(**u** | **U**)	Print in unsigned decimal (as **integer** or **long**).
(**o** | **O**)	Print in octal (as **integer** or **long**).
(**x** | **X**)	Print in hexadecimal (as **integer** or **long**).
(**b** | **B**)	Print a byte in decimal (either way).
(**c** | **C**)	Print a character (either way).
(**e** | **E**)	Print in "e" floating point notation (as **float** or **double**) (see *printf*(3S)). Remember that floating point expressions are always doubles!
(**f** | **F**)	Print in "f" floating point notation (as **float** or **double**).
(**g** | **G**)	Print in "g" floating point notation (as **float** or **double**).
i	Print a machine instruction.
a	Print a string using *expr* as the address of the first byte.
s	Print a string using *expr* as the address of a pointer to the first byte (same as "**expr*/**a**", except for arrays).
t	Show the type of *expr* (usually a variable or procedure name). For true procedure types you must actually call the procedure, e.g. "def(arg)/t".
p	Print the name of the procedure containing address *expr*.
S	Do a formatted dump of a structure. *expr* must be the address of a structure, not the address of a pointer to a structure.

There are some short hand notations for *size*:

b	1 byte (**char**).
s	2 bytes (**short**).
l	4 bytes (**long**).
D	8 bytes (double). Can only be used with floating-point formats.

These can be appended to *formchar* instead of a numeric *size*. For example, "abc/xb" prints one byte in hexadecimal.

If you view an object with a *size* (explicitly or implicitly) less than or equal to the size of a **long**, the debugger changes the basetype to something appropriate for that *size*. This is so "**.**" (*dot*) works correctly for assignments. For example, "abc/c2" sets the type of "**.**" to **short**. One side effect is that if you look at a **double** using a **float** format, *dot* loses accuracy or has the wrong value.

The value of the **$print** special variable affects the default format for character types as follows: for "ascii" mode the default format is "**x**" for unsigned char. In "native" and "raw" modes, the default for unsigned char is "**c**". Likewise, "**s**" is used for pointer to unsigned char.

Data Viewing Commands

expr	If *expr* does not resemble anything else (such as a command), it is handled as "*expr*/**n**" (print expression in normal format), unless followed by "**;**" or "**}**", in

which case nothing is printed.

expr/*format* Print the contents (value) of *expr* using *format*.

expr?*format* Print the address of *expr* using *format*.

^[[/]*format*] Back up to the preceding memory location (based on the size of the last thing displayed). Use *format* if supplied, or the previous *format* if not. No "/" is needed after the "^". To reverse direction again (e.g. start going forward), enter "." (*dot*) (always an alias for the current location) followed by carriage return.

l [*proc*[.*depth*]] List all parameters and local variables for current procedure (or *proc*, if given, at the specified *depth*, if any). Data display uses "**/n**" format, except arrays and pointers are shown as addresses; only the first word of a structure is shown.

l (**a** | **b** | **d** | **z**) List all **a**ssertions, **b**reakpoints, **d**irectories (where to search for files), or **z**ignals (signal actions).

l (**f** | **g** | **l** | **m** | **p** | **r** | **s**) [*string*]
List all **f**iles (source files which built *objectfile*), **g**lobal variables, **l**abels (program entry points known to the linker), **m**acros, **p**rocedure names, **r**egisters, or **s**pecial variables (except registers). If *string* is present, only those things with the same initial characters are listed.

Stack Viewing Commands

[*depth*] **t** Trace the stack for the first *depth* (default 20) levels.

[*depth*] **T** The same as "**t**", but local variables are also displayed using "**/n**" format (except that arrays and pointers are shown as addresses; structures show first word only).

Job Control Commands

The parent (debugger) and child (*objectfile*) processes take turns running. The debugger is only active while the child process is stopped due to a signal (includes hitting a breakpoint) or terminated for whatever reason.

r [*arguments*] Run a new child process with the given argument list, if any (an existing child process is terminated first). If no *arguments* are given, those used with the last "**r**" command are used again (none if "**R**" was used last).

Arguments can contain "<" and ">" for redirecting standard input and standard output. ("<" does an *open*(2) of file descriptor 0 for read-only; ">" does a *creat*(2) of file descriptor 1 with mode 0666.) Redirection may also be done with ">>" and ">&". *Arguments* can contain shell variables, quote marks, or other special syntax (expanded by Bourne shell). The remainder of the input-line following the "**r**" command is used as the argument-list, so it cannot be enclosed in a command list ("{}"). Thus, "**r**" cannot be used within a breakpoint, assertion, or "**if**" command.

R Run a new child process with no argument list.

k Terminate (kill) the current child process, if any.

[*count*] **c** [*line*] Continue after a breakpoint or a signal, ignoring the signal, if any. If *count* is given, the current breakpoint, if any, has its *count* set to that value. If *line* is given, a temporary breakpoint is set at that line number, with a *count* of −1 (see Breakpoint Commands).

[*count*] **C** [*line*] Continue like "**c**", but allow the signal (if any) to be received.

[*count*] **s** Single step 1 (or *count*) statements (successive carriage-returns repeat with a *count* of 1). If *count* is less than one, the child process is not stepped. The child process continues with the current signal, if any (set "$signal = 0" to prevent this).

[*count*] **S** Single step like "**s**", but treat procedure calls as single statements (don't follow them down). If a breakpoint is hit in such a procedure, or in one that it calls, its *commands* are executed. (This is usually acceptable unless there is a "**c**" command in that breakpoint's command list.)

[*count*] **j** Single step 1 (or *count*) machine instructions. Successive carriage-returns repeat with a *count* of 1. If *count* is less than one, the child process is not stepped. The child process continues with the current signal, if any. (Set "$signal = 0" to prevent the signal.)

[*count*] **J** Single step like "**j**", but treat procedure calls as single instructions (Don't follow them down.) If a breakpoint is hit in such a procedure, or in one that it calls, its *commands* are executed. (This is usually all right unless there is a "**c**" command in that breakpoint's command list.)

The debugger has no knowledge of or control over child processes forked in turn by the process being debugged. Programs being debugged should not execute a different program via *exec*(2).

Child process output might be buffered, so it might not appear immediately after you step through an output statement such as *printf*(3S). It might not appear at all if you kill the process.

Breakpoint Commands

The debugger provides a number of commands for setting and deleting breakpoints. Associated wtih a breakpoint are three attributes:

address All the commands that set a breakpoint are simply alternate ways to specify the breakpoint address. The breakpoint is encountered whenever *address* is about to be executed, regardless of the path taken to get there. Only one breakpoint at a time (of a particular type) can be set at a given *address*. Setting a new breakpoint at *address* replaces an old one of the same type, if any.

count This is the number of times the breakpoint is encountered prior to recognition. If *count* is positive, the breakpoint is "permanent", and *count* decrements with each encounter. Each time *count* goes to zero, the breakpoint is recognized and *count* is reset to one (so it stays there until explicitly set to a different value by "**c**" or "**C**").

If *count* is negative, the breakpoint is "temporary", and *count* increments with each encounter. Once *count* goes to zero, the breakpoint is recognized, then deleted.

commands These are the actions to be taken upon recognition of a breakpoint before waiting for command input. These are separated by "**;**" and can be enclosed in "{}" to delimit the list saved with the breakpoint from other commands on the same line.

Results of expressions followed by "**;**" or "}" are not printed unless you specify a print format.

Saved commands are not parsed until the breakpoint is recognized. If there are no *commands*, the debugger will wait for command input when the breakpoint is recognized. For immediate continuation, finish the command list with

"**c**".

The debugger has only one active command line at a time. When it begins to execute breakpoint commands, the remainder (if any) of the old command line is discarded with notice given.

Here are the breakpoint commands:

lb
B Both forms list all breakpoints in the format:

num: count: *nnn proc*: *ln*: *contents*
{*commands*}

The leftmost number *num* is an index number for use with the "**d**" (delete) command.

[*line*] **b** [*commands*]
Set a permanent breakpoint at the current line (or at *line* in the current procedure).

[*expr*] **d** Delete breakpoint number *expr*. If *expr* is absent, delete the breakpoint at the current line, if any. If there is none, the debugger executes a "**B**" command instead.

bp [*commands*] Set permanent breakpoints at the beginning (first executable line) of every debuggable procedure. When any procedure entry breakpoint is hit, *commands* are executed.

It is permissible to set other permanent or temporary breakpoints at the same locations as procedure breakpoints. If a procedure and non-procedure breakpoint are both hit at the same location, the non-procedure breakpoint has priority. It is not possible to alter the "count" of a procedure breakpoint. Procedure entry breakpoints must be activated and deleted as a group; it is not possible to set or delete individual ones.

Procedure entry breakpoints are useful for procedure stepping and tracing. For example, the command

```
bp Q;t 1;c
```

sets up procedure tracing by printing the current procedure at each breakpoint.

bpx [*commands*] Set permanent breakpoints at the exit (final executable line) of every debuggable procedure. When any procedure exit breakpoint is hit, commands are executed.

bpt [*commands*] Set permanent breakpoints at the entry and exit (first and final executable line) of every debuggable procedure. The commands, if any, are associated wih the entry breakpoint. Commands for the exit breakpoint are "**Q;L;c**".

D [**b**] Delete all breakpoints (including "procedure" breakpoints). The "**b**" is optional.

D p Delete all "procedure entry" breakpoints. All breakpoints set by commands other than "**bp**" remain set.

Dpx Delete all "procedure exit" breakpoints. All breakpoints set by commands other than "**bpx**" will remain set.

Dpt Delete all "procedure trace" breakpoints. All breakpoints set by commands other than "**bpt**" will remain set.

abc *commands* Define a global breakpoint command list that will be executed whenever any breakpoint is hit (normal, procedure, procedure exit, procedure trace.)

dbc Delete the global breakpoint command list.

For the following commands, if the second character is uppercase, for example, **"bU"** instead of **"bu"**, the breakpoint is temporary (*count* is −1), not permanent (*count* is 1).

[*depth*] **bb** [*commands*]
[*depth*] **bB** [*commands*]
Set a breakpoint at the beginning (first executable line) of the procedure at the specified stack *depth*. If *depth* is not specified, use the current procedure (might not be the same as the one at *depth* zero).

[*depth*] **bx** [*commands*]
[*depth*] **bX** [*commands*]
Set a breakpoint at the exit (last executable line) of the procedure at the given stack *depth*. If *depth* is not specified, use the current procedure (might not be the same as the one at *depth* zero). The breakpoint is set such that all returns of any kind go through it.

[*depth*] **bu** [*commands*]
[*depth*] **bU** [*commands*]
Set an up-level breakpoint. The breakpoint is set immediately after the return to the procedure at the specified stack *depth* (default one, not zero). Zero *depth* means "current location".

[*depth*] **bt** [*proc*] [*commands*]
[*depth*] **bT** [*proc*] [*commands*]
Trace current procedure (or procedure at *depth*, or *proc*). Set breakpoints at entrance and exit of a procedure. Default entry breakpoint *commands* are **"Q;2t;c"** (show the top two procedures on stack and continue). The exit breakpoint always executes **"Q;L;c"** (print the current location and continue).

If *depth* is given, *proc* must be absent or it is taken as part of *commands*. If *depth* is missing but *proc* is specified, the named procedure is traced. If both *depth* and *proc* are omitted, the current procedure is traced, which might not be the same as the one at *depth* zero.

If *commands* are present, they are used for the entrance breakpoint, instead of the default shown above.

address **ba** [*commands*]
address **bA** [*commands*]
Set a breakpoint at the given code address. The *address* can be the name of a procedure or an expression containing such a name. If the *address* is in a non-debuggable procedure, or in prologue code (before the first executable line of a procedure), results might seem a little strange. Unexpected results or errors may also result from using an *address* which is not in the child process.

sb [*num*] Suspend breakpoint number *num*. If no breakpoint is given, suspend the breakpoint at the current line.

sb * Suspend all breakpoints.

ab [*num*] Activate breakpoint number *num*. If no breakpoint is given, activate the breakpoint at the current line.

ab * Activate all breakpoints.

The next few commands are not strictly part of the breakpoint group, but are used almost exclusively as arguments to breakpoints (or assertions).

if [*expr*] {*commands*}[{*commands*}]
: If *expr* evaluates to a non-zero value, the first group of commands (the first "{}" block) is executed, otherwise it (and the following "{", if any) is skipped. All other "{}" blocks are always ignored (skipped), except when given as an argument to an "**a**", "**b**", or "**!**" command. The "**if**" command is nestable, and can be abbreviated to "**i**".

Q
: If the "**Q**uiet" command appears as the first command in a breakpoint's command list, the normal announcement of "*proc: line: text*" is not made. This allows quiet checks of variables, etc. to be made without cluttering up the screen with unwanted output. The "**Q**" command is ignored if it appears anywhere else.

"*any string you like*"
: Print the given string. The string may contain standard backslashed character escapes, including "**\n**" for newline. This is useful for labelling output from breakpoint commands.

Assertion Control Commands

Assertions are command lists that are executed before every statement. Thus, if there is even one active assertion, the program is single stepped at the machine instruction level (runs very slowly). They are primarily used for tracking down nasty bugs (such as the corruption of a global variable).

Assertions can be activated or suspended individually, and there is an overall mode. If any assertion is added or activated, or if all assertions become suspended, the global mode follows suit.

a *commands*
: Create a new assertion with the given command list. This list is not parsed until execution time. The command list can be enclosed in "{}" to delimit it from other commands on the same line. The "**la**" command lists all current assertions and the overall mode.

expr **a** (**a** | **d** | **s**)
: Modify the assertion numbered *expr*: **a**ctivate it, **d**elete it, or **s**uspend it. Suspended assertions continue to exist, but do nothing until reactivated.

A
: Toggle the overall state of the assertions mechanism between *active* and *suspended*.

D a
: Delete all assertions.

[*flag*] **x**
: Force exit from assertions mode. If *flag* is absent or evaluates to zero, exit immediately. Otherwise, finish executing the current assertion first. If an assertion executes an "**x**" command, the child process stops and the assertion doing the "**x**" is identified.

The debugger has only one active command line at a time. The current command line is discarded when assertion execution begins.

Commands "**r**", "**R**", "**c**", "**C**", "**s**", "**S**", and "**k**" are not allowed while assertions are running. They must appear after the "**x**", if at all.

A useful assertion might be:

```
a L
```

This just traces execution a line at a time until "something" happens (e.g., you hit the BREAK key).

Another example:

```
a L; if(xyz > (def − 9) * 10) {ta; x 1; c} {abc −= 10}
```

This assertion prints the line just executed, then checks the condition. If it is false, "abc" is decremented by 10. If it is true, assertions are suspended, assertion mode is exited, and the program continues at normal speed. Without the number after the "**x**" command, the "**c**" command is not executed.

Another example:

```
a if (abc != $abc) {$abc = abc; abc/d; if (abc > 9) {x}}
```

This command sets up an assertion to report the changing value of some global variable ("abc"), and stop if it ever exceeds some value. It uses a debugger local variable ("$abc") to keep track of the old value of "abc".

Signal Control Commands

The debugger catches all signals bound for a child process before the child process sees them (a function of *ptrace*(2)). For many signals, this is a reasonable thing to do. Most processes are not set up to handle segmentation errors, etc. However, some processes do quite a bit with signals and the constant need to continue from a signal catch can be tedious.

[*signal*] **z [i][r][s][Q]**

Modify the "zignal" (signal) handling table. *Signal* is a valid signal number (default is the current signal). The options (which must be all one word) toggle the state of the appropriate flag: **i**gnore, **r**eport, or **s**top. If "**Q**" is present, the new signal state is not printed.

Use "**lz**" to list the current handling of all signals. Note that just "**z** *signal*" with no options tells you the state of the selected signal.

For example, assuming a start up state of (don't ignore, don't report, don't stop), the command "**14z sr**" sets the alarm clock signal to **stop** (but still don't **ignore**) and **report** that it occurred. Doing "**14z sr**" again toggles the flags back to the original state.

When the child process stops or terminates on a signal it is always reported, except for the breakpoint signal when the breakpoint commands start with "**Q**".

When the debugger ignores a signal, the "**C**" command does not know about it. A signal is never ignored when the child process terminates, only when it stops.

Record and Playback Commands

The debugger supports a record/playback feature to help recreate program states and to record all debugger output. It is particularly useful for bugs requiring long setups. Note: The file name cannot be "**t**", "**f**", or "**c**", or begin with a "**@**".

Commands are:

>*file* Set or change the recordfile to *file* and turn recording on. This rewrites *file* from the start. Only commands are recorded to this file.

>>*file* This is the same as >>*file* but appends to *file* instead of overwriting.

>**@***file*
>>**@***file* Set or change the record-all file to *file* for overwriting or appending. The record-all file can be opened or closed independent of the recordfile. All debugger standard output is copied to the record-all file, including prompts, commands entered, and command output (does not capture child process output).

>(**t** | **f** | **c**) Turn recording on ("**t**") or off ("**f**"), or close the recording file ("**c**"). When recording is resumed, new commands are appended to previous file contents.

In this context, ">>" is equivalent to ">".

>@(t | f | c) Turn record-all on, off, or close the record-all file. In this context, ">>@" is equivalent to ">@".

> Tell the current recording status (same as ">>").

>@ Tell the current record-all status (same as ">>@").

<file Start playback from *file*.

<<file Start playback from *file*, using the single-step feature of playback.

Only command lines read from the keyboard or a playback file are recorded in the recordfile. For example, if recording is turned on in an assertion, it does not "take effect" until assertion execution stops.

Command lines beginning with ">", "<", or "!" are not copied to the current recordfile (they are copied to the record-all file). To override this, begin such lines with blanks.

NOTE: The debugger can be invoked with standard input, standard output, and/or standard error redirected, independent of record and playback. If the debugger encounters an end-of-file while standard input is redirected from anything other than a terminal, it prints a message to standard output and exits, returning zero.

Macro Definition Commands

def *name* [*replacement-text*]
Define *name* as the macro whose value is *replacement-text*. *Name* can be any string of letters or digits. *Replacement-text* can be any string of letters, digits, blanks, tabs, or other printing characters; however, it cannot be continued onto a new line.

undef *name* Remove the macro definition from *name*, causing *name* to cease existing as a replacement string macro. As a special case "*" can be entered for *name* to undefine all macros.

tm Toggle the macro substitution mechanism between active and suspended states. When macro substitution is suspended, the currently defined macros continue to exist, but they are not replaced in the command line by their definitions. Additional macros can be defined while macro substitution is suspended.

Miscellaneous Commands

sm Suspend the "more" (pagination) facility of the debugger output. This is most useful when breakpoints or assertions are printing a great deal of information to the screen, and you do not want the debugger to keep waiting for you to hit the space bar.

am Activate the "more" facility to paginate the debugger output.

<carriage-return>
~ Repeat the last command, if possible, with an appropriate increment, if any. Repeatable commands are those which print a line, print a window of lines, print a data value, single step, and single step over procedures. <carriage-return> is saved in a *record* file as a "~" command, to distinguish from ^**D**.

^**D** Control-D is like <carriage-return>, but repeats the previous command ten times. This command is saved in a *record* file as an empty line.

! [*command-line*]
Invoke a shell program. If *command-line* is present, it is executed via *system*(3S). Otherwise, the environment variable SHELL gives the name of the shell program to invoke with a **−i** option, also using *system*(3S). If SHELL is

not found, the debugger executes "/bin/sh −i". In any case, the debugger then waits for the shell or *command-line* to complete.

As with breakpoints, *command-line* can be enclosed in "{}" to delimit it from other (debugger) commands on the same line. For example,

```
14b {!{date};c}; t; la
```

sets a breakpoint at line 14 that calls *date*(1), then continues; then (after setting the breakpoint), the debugger does a stack trace, then lists assertions.

[*text*] Flag *text* as a comment to be echoed to the command window. The **#** must appear as the first non-blank character on the line and the remainder of the line is treated as a comment. It is also written to the open record file.

f [**"***printf-style-format***"**]
Set address printing format, using *printf*(3S) format specifications (**not** debugger format styles). Only the first 19 characters are used. If there is no argument, the format is set to a system-dependent default. All addresses are assumed to be of type **long**, so you should handle all four bytes to get something meaningful.

F Find and fix bug (a useless but humorous command).

g *line* | *#label* Go to an address in the procedure on the stack at *depth* zero (not necessarily the current procedure). It changes the program counter, making *line* the next line to be executed.

h
help Print the debugger help file (command summary) using *more*(1).

I Print information (inquire) about the state of the debugger.

M Print the current text (*objectfile*) and core (*corefile*) address maps.

M (t | **c)** [*expr*; [*expr*;...]]
Set the **t**ext (*objectfile*) or **c**ore (*corefile*) address map. The first zero to six map values are set to the *expr*s given (refer to adb(1) for details on address maps).

q Quit the debugger. This requests confirmation.

Z Toggle case sensitivity in searches. This affects everything: file names, procedure names, variables, and string searches! The debugger starts out as case insensitive.

ADOPTING AN EXISTING PROCESS

The debugger has the capability to adopt and debug a free running process. This is accomplished by using the −**P** *process ID* option when the debugger is invoked.

To adopt a process, the effective user IDs of the debugger and the process to be adopted must match, or the effective user ID of the debugger must be root. When a process is adopted, it halts, and the debugger displays where the program is halted, at which point the program can be debugged. If the user quits the debugger without killing the process, the debugger removes all breakpoints from the process and allows it to continue running. If a program is designed to be adopted by the debugger when in a certain state (such as an error condition), it is important that the program infinite loop, rather than calling the system routine *sleep()*. A sleeping program cannot be adopted correctly by the debugger.

DIAGNOSTICS

Most errors cause a reasonably accurate message to be given. Normal debugger exits return zero and error exits return one. All debugger output goes to standard output except error messages given just before non-zero exits, which go to standard error.

Debugger errors are preceded by "panic: ", while user errors are not. If any error occurs during initialization, the debugger then prints "cannot continue" and quits. If any error happens after initialization, the debugger attempts to reset itself to an idle state, waiting for command input. If any error occurs while executing a procedure call from the command line, the context is reset to that of the normal program.

Child process (program) errors result in signals which are communicated to the debugger via the *ptrace*(2) mechanism. If a program error occurs while executing a procedure call from the command line, it is handled like any other error (i.e. you can investigate the called procedure). To recover from this, or to abort a procedure call from the command line, type DEL, BREAK, ^C, or whatever your interrupt character is.

WARNINGS

The debugger does not terminate on an interrupt (SIGINT); it jumps to its main loop and awaits another command. However, this does not imply that sending the debugger an interrupt is harmless. It can result in internal tables being left in an inconsistent state, which might produce incorrect behavior.

Code that cannot be debugged or does not have a corresponding source file is dealt with in a half-hearted manner. The debugger shows "unknown" for unknown file and procedure names, cannot show code locations or interpret parameter lists, etc. However, the linker symbol table provides procedure names for most procedures, even if they cannot be debugged.

If you set the address printing format to something *printf*(3S) doesn't like, you might get an error (usually memory fault) each time you try to print an address, until you fix the format with another "**f**" command.

Do not use the "**z**" command to manipulate the SIGTRAP signal. If you change its state you had better know what you are doing or be a very good sport!

If you single step or run with assertions through a call to *longjmp*(3C), the child process will probably take off free-running as the debugger sets but never hits an up-level breakpoint.

Do not modify any file while the debugger has it open. If you do, the debugger gets confused and might display garbage.

Although the debugger tries to do things reasonably, it is possible to confuse the recording mechanism. Be careful about trying to playback from a file currently open for recording, or vice versa; strange things can happen.

Some compilers only issue source line symbols at the end of each logical statement or physical line, whichever is greater. Therefore, if you are accustomed to saying "a = 0; b = 1;" on one line, you cannot put a breakpoint after the assignment to "a" but before the assignment to "b".

Some statements do not emit code where you would expect it. For example, assume:

```
 99:    for (i = 0; i < 9; i++) {
100:            xyz (i);
101:    }
```

A breakpoint placed on line 99 will be hit only once in some cases. The code for incrementing is placed at line 101. Each compiler is a little different; you must get used to what your particular compiler does. A good way of finding out is to use single stepping to see in what order the source lines are executed.

The output of some program generators, such as *yacc*(1), have compiler line number directives in them that can confuse the debugger. It expects source line entries in the symbol table to appear in sorted order. Removal of line directives fixes the problem, but makes it more difficult to find error locations in the original source file. The following script, run after *yacc*(1) and before *cc*(1), comments out line number changes in C programs:

```
sed "/# *line/s/^.*$/\/*&*\//" y.tab.c >temp.c
```

yacc(1) will leave out line directives if invoked with the **-l** option. In general, line number directives (or compiler options) are only safe so long as they never set the number backwards.

The C operators "++", "--", and "**?:**" are not available. The debugger always understands all the other C operators, except "**sizeof**", if the default language is FORTRAN or Pascal.

For FORTRAN, only the additional operators "**.NE.**", "**.EQ.**", "**.LT.**", "**.LE.**", "**.GT.**", "**.GE.**", "**.OR.**", "**.NOT.**", "**.AND.**", "**.EQV.**", and "**.NEQV.**" are supported.

For Pascal, only the operators "**:=**", "**<>**", "**^**", "**^.**" (as in "x^.y"), "**and**", "**or**", "**not**", "**div**", "**mod**", "**addr**", and "**sizeof**" are added.

There is no support for FORTRAN **complex** variables, except as a series of two separate **float**s or **double**s.

The C operators "**&&**" and "**||**" aren't short circuit evaluated as in the compiler. All parts of expressions involving them are evaluated, with any side-effects, even if it's not necessary.

The debugger doesn't understand C pointer arithmetic. "*(a+n)" is not the same as "a[n]" unless "a" has an element size of 1.

There is no support for C local variables declared in nested blocks, nor for any local overriding a parameter with the same name. When looking up a local by name, parameters come first, then locals in the order of the "}"s of the blocks in which they are declared. When listing all locals, they are shown in the same order. When there is a name overlap, the address or data shown is that of the first variable with that name.

Pdb does not support identically named procedures (legal in Pascal if the procedures are in different scopes). *Pdb* always uses the first procedure with the given name.

There is no support for Pascal packed arrays where the element size is not a whole number of bytes. Any reference into such an array might produce garbage or a bad access.

Pascal WITH statements are not understood. To access any variable you must specify the complete "path" to it.

The debugger supports call-by-reference only for known parameters of known (debuggable) procedures. If the object to pass lives in the child process, you can fake such a call by passing "& object", i.e. the address of the object.

Array parameters are always passed to command-line procedure calls by address. This is correct except for Pascal call-by-value parameters. Structure parameters are passed by address or value, as appropriate, but only a maximum of eight bytes is passed, which can totally confuse the called procedure. Functions which return complex numbers are not called correctly; insufficient stack space is allocated for the return area, which can lead to overwriting the parameter values.

Assignments into objects greater than four bytes in size, from debugger special variables, result in errors or invalid results.

Command lines longer than 1024 bytes are broken into pieces of that size. This might be relevant if you run the debugger with playback or with input redirected from a file.

When a C parameter is declared as an array of anything, the highest type qualifier (array) shows up as a pointer instead. For example, "int x[]" looks like "int *x", and "char (*x)[]" looks like "char **x", but "char *x[]" is treated correctly as "pointer to array of **char**".

There is limited support for command-line calls of functions which return structures. The debugger interprets the start of heap as a structure of the return type. However, a call such as "abc()/t" displays the return type correctly.

$short and **$long** are available in addition to **$result**. If command-line procedure call returns a **double**, **$result** is set to the value cast to **long**.

Procedures in FORTRAN and Pascal can have alias names in addition to normal names. Aliases are shown by the "**lp**" (list procedures) command. They can be used in place of the normal name, as desired.

The procedure name "_MAIN_" is used as the alias name for the main program (main procedure) in all supported languages. Do not use it for any debuggable procedures.

FORTRAN ENTRY points are flagged "ENTRY" by the "**lp**" command.

When a compiler does not know array dimensions, such as for some C and FORTRAN array parameters, it uses 0:MAXINT or 1:MAXINT, as appropriate. The "**/t**" format shows such cases with "**[]**" (no bounds specified), and subscripts from 0 (or 1) to MAXINT are allowed in expressions.

There is no support for: C structure, union, and enumeration tags, C typedefs, and Pascal types.

Some variables are indirect, so a child process must exist in order for the debugger to know their addresses. When there is no child process, the address of any such variable is shown as 0xfffffffe.

Symbol names in the Value Table are never preceded by underscores, so the debugger never bothers to search for names of that form. The only time a prefixed underscore is expected is when searching the Linker Symbol Table for names of non-debuggable procedures.

Two types of string formats are supported in addition to null-terminated C strings. FORTRAN *character* variables consist of a string of bytes (no null terminator). Pascal *string* variables consist of a length byte, followed by the string characters. The "/s" and "/a" formats will display these types correctly only if the current language is FORTRAN or Pascal.

AUTHOR

Cdb was developed by Third Eye Software.

FILES

a.out	Default *objectfile* to debug.
core	Default *corefile* to debug.
/usr/lib/cdb.help	Text file listed by the "**help**" command.
/usr/lib/end.o	Object file to link with all debuggable programs.
$HOME/.cdbrc	Startup file for cdb.
$HOME/.fdbrc	Startup file for fdb.
$HOME/.pdbrc	Startup file for pdb.

SEE ALSO

cc(1), echo(1), fc(1), pc(1), ld(1), more(1), creat(2), exec(2), fork(2), open(2), setjmp(3C), printf(3S), system(3S), a.out(4), LANG(5).

On some systems any of the following might exist: adb(1), ptrace(2), core(4), ksh(1).

C Debugger (cdb), tutorial in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LANG determines the local language equivalent of **y** (for yes/no queries). LANG also determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cdb* behaves as if all internationalization variables are set to "C". See *environ*(5).

NAME

cdc − change the delta commentary of an SCCS delta

SYNOPSIS

cdc −r*SID* [−**m**[*mrlist*]] [−**y**[*comment*]] *files*

DESCRIPTION

Cdc changes the *delta commentary*, for the **SID** specified by the −**r** keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (**MR**) and comment information normally specified via the *delta*(1) command (−**m** and −**y** keyletters).

If a directory is named, *cdc* behaves as if each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of − is given, the standard input is read (see WARNINGS); each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to *cdc*, which can appear in any order, consist of *keyletter* arguments and file names.

All the described *keyletter* arguments apply independently to each named file:

−r*SID* Used to specify the *S*CCS *ID*entification (**SID**) string of a delta for which the delta commentary is to be changed.

−**m**[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)), a list of **MR** numbers to be added and/or deleted in the delta commentary of the **SID** specified by the −**r** keyletter *may* be supplied. A null **MR** list has no effect.

MR entries are added to the list of **MR**s in the same manner as that of *delta*(1). To delete an **MR**, precede the **MR** number with the character **!** (see EXAMPLES). If the **MR** to be deleted is currently in the list of **MR**s, it is removed and changed into a "comment" line. A list of all deleted **MR**s, is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If −**m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see −**y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR**s list.

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) that validates the correctness of the **MR** numbers. If a non-zero exit status is returned from the **MR** number validation program, *cdc* terminates and the delta commentary remains unchanged.

−**y**[*comment*] Arbitrary text used to replace the *comment* or *comment*s already existing for the delta specified by the −**r** keyletter. Previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If −**y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before standard input is read; if standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in the *SCCS* tutorial in *HP-UX Concepts and Tutorials*. Simply stated, they are either: (1) if you made the delta, you

can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

```
cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file
```

adds bl78-12345 and bl79-00001 to the **MR** list, removes bl77-54321 from the **MR** list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc -r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (– on the command line), the **–m** and **–y** keyletters must also be used.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).
SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

FILES

x-file (see *delta*(1))
z-file (see *delta*(1))

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

cflow − generate C flow graph

SYNOPSIS

cflow [−**r**] [−**ix**] [−**i_**] [−**d**num] files

DESCRIPTION

Cflow analyzes a collection of C, YACC, LEX, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in **.y**, **.l**, **.c**, and **.i** are YACC'd, LEX'd, and C-preprocessed (bypassed for **.i** files) as appropriate and then run through the first pass of *lint*(1). (The −**I**, −**D**, and −**U** options of the C-preprocessor are also understood.) Files suffixed with **.s** are assembled and information is extracted (as in **.o** files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the −**i** inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (e.g., **char** *), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only <> is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}

f()
{
        i = h();
}
```

the command

```
cflow −ix file.c
```

produces the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the **−e** option of *pr*(1) can be used to compress the tab expansion to something less than every eight spaces.

The following options are interpreted by *cflow*:

−r Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is sorted in ascending collation order by callee (see Environment Variables below).

−ix Include external and static data symbols. The default is to include only functions in the flowgraph.

−i_ Include names that begin with an underscore. The default is to exclude these functions (and data if *−ix* is used).

−dnum The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

DEPENDENCIES

Series 300

The following option is supported:

−Y Enable support of 16-bit characters inside string literals and comments. Note that 8-bit parsing is always supported. See *hpnls*(5) for more details on International Support.

SEE ALSO

as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating order output by the **−r** option.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cflow* behaves as if all internationalization variables are set to "C". See *environ*(5).

STANDARDS CONFORMANCE

cflow: SVID2, XPG2, XPG3

NAME

chacl – add, modify, delete, copy, or summarize access control lists (ACLs) of files

SYNOPSIS

chacl *acl file ...*
chacl **–r** *acl file ...*
chacl **–d** *aclpatt file ...*
chacl **–f** *fromfile tofile ...*
chacl **–[z|Z|F]** *file ...*

Remarks:

To ensure continued conformance with emerging industry standards, features described in this manual entry are likely to change in a future release.

DESCRIPTION

Chacl extends the capabilities of *chmod*(1), by enabling the user to grant or restrict file access to additional specific users and/or groups. Traditional file access permissions, set when a file is created, grant or restrict access to the file's owner, group, and other users. *Chacl* enables a user to designate up to thirteen additional sets of permissions (called optional access control list (ACL) entries) which are stored in the access control list of the file.

To use *chacl*, the owner (or superuser) constructs an *acl*, a set of *(user.group, mode)* mappings to associate with one or more files. A specific user and group can be referred to by either name or number; any user (*u*), group (*g*), or both can be referred to with a % symbol, representing no specific user or group. The person executing the command can refer to the file's owner or group using the @ symbol.

Read, write, and execute/search (**rwx**) *modes* are identical to those used by *chmod*; symbolic operators (*op*) add (+), remove (–), or set (=) access rights. The entire *acl* should be quoted if it contains whitespace or special characters. Although two variants for constructing the *acl* are available (and fully explained in *acl*(5)), the following syntax is suggested:

entry[, *entry*] ...

where the syntax for an *entry* is

u.g op mode [*op mode*] ...

By default, *chacl* modifies existing ACLs. It adds ACL entries or modifies access rights in existing ACL entries. If *acl* contains an ACL entry already associated with a file, the entry's mode bits are changed to the new value given, or are modified by the specified operators. If the file's ACL does not already contain the specified entry, that ACL entry is added. *Chacl* can also remove all access to files. Giving it a null *acl* argument means either "no access" (when using the **–r** option) or "no changes."

For a summary of the syntax, run *chacl* without arguments.

If *file* is specified as –, *chacl* operates on the file open as standard input.

Options

–r Replace old ACLs with the given ACL. All optional ACL entries are first deleted from the specified files's ACLs, their base permissions are set to zero, and the new ACL is applied. If *acl* does not contain an entry for the owner (*u*.%), the group (%.*g*), or the other (%.%) users of a file, that base ACL entry's mode is set to zero (no access). The command affects all of the file's ACL entries, but does not change the file's owner or group ID.

In *chmod*(1), the "modify" and "replace" operations are distinguished by the syntax (string or octal value). There is no corollary for ACLs because they have a variable number of entries. Hence *chacl* modifies specific entries by default,

and optionally replaces all entries.

−d Delete the specified entries from the ACLs on all specified files. The *aclpatt* argument can be an exact ACL or an ACL pattern (see *acl*(5)). *Chacl* **−d** updates each file's ACL only if entries are deleted from it.

If you attempt to delete a base ACL entry from any file, the entry remains but its access mode is set to zero (no access). If you attempt to delete a non-existent ACL entry from a file (that is, if an ACL entry pattern matches no ACL entry), *chacl* informs you of the error, continues, and eventually returns non-zero.

−f *fromfile* Copy the ACL from *fromfile* to the specified *tofile*, transferring ownership, if necessary (see *acl*(5), *chown*(2), or *chownacl*(3C)). *Fromfile* can be − to represent standard input.

This option implies the **−r** option. If the owner and group of *fromfile* are identical to those of *tofile*, *chacl* **−f** is identical to:

chacl −r 'lsacl fromfile' tofile ...

To copy an ACL without transferring ownership, the above command is suggested instead of *chacl* **−f**.

−z Delete ("zap") all optional entries in the specified file's ACLs, leaving only base entries.

−Z Delete ("zap") all optional entries in the specified file's ACLs, and set the access modes in all base entries to zero (no access). This is identical to replacing the old ACL with a null ACL:

chacl −r '' file ...

or using *chmod*(1), which deletes optional entries as a side effect:

chmod 0 file ...

−F Incorporate ("fold") optional ACL entries into base ACL entries. The base ACL entry's permission bits are altered, if necessary, to reflect the caller's effective access rights to the file; all optional entries, if any, are deleted.

For ordinary users, only the access mode of the owner base ACL entry can be altered. Unlike *getaccess*(1), the write bit is not turned off for a file on a read-only file system or a shared-text program being executed.

For superusers, only the execute mode bit in the owner base ACL entry might be changed, only if the file is not an regular file or if an execute bit is not already set in a base ACL entry mode, but is set in an optional ACL entry mode.

Acl also can be obtained from a string in a file:

chacl 'cat file' files...

Using @ in *acl* to represent "file owner or group" might cause *chacl* to run more slowly because it must reparse the ACL for each file (except with the **−d** option).

RETURN VALUE

If *chacl* succeeds, it returns a value of zero.

If *chacl* encounters an error before it changes any file's ACL, it prints an error message to standard error and returns 1. Such errors include invalid invocation, invalid syntax of *acl* (*aclpatt*), a given user name or group name is unknown, or inability to get an ACL from *fromfile* with the **−f** option.

NAME

chatr – change program's internal attributes

SYNOPSIS

chatr [**−n**] [**−q**] [**−s**] *file* ...

DESCRIPTION

Chatr, by default, prints each *file*'s magic number and file attributes to the standard output. With one or more optional arguments, *chatr* performs the following operations:

−n Change *file* from demand loaded to shared.

−q Change *file* from shared to demand loaded.

−s Perform its operation silently.

Upon completion, *chatr* prints the file's old and new values to standard output unless **−s** is specified.

RETURN VALUE

Chatr returns zero on success. If the call to *chatr* is syntactically incorrect, or one or more of the specified files cannot be acted upon, *chatr* returns the number of files whose attributes could not be modified. If no files are specified, *chatr* returns decimal 255.

DIAGNOSTICS

The error messages produced by *chatr* are self-explanatory.

AUTHOR

Chatr was developed by HP.

SEE ALSO

ld(1), a.out(4), magic(4).

NAME

checknr – check nroff/troff files

SYNOPSIS

checknr [**−s**] [**−f**] [**−a**.x1.y1.x2.y2. ... .xn.yn] [**−c**.x1.x2.x3 ... .xn] [*file* ...]

DESCRIPTION

Checknr checks a list of *nroff*(1) or *troff* input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, *checknr* checks the standard input. Delimiters checked are:

(1) Font changes using \fx ... \fP.

(2) Size changes using \sx ... \s0.

(3) Macros that come in open ... close forms, such as the .TS and .TE macros, both of which must appear.

Checknr knows about the *ms* and *me* macro packages.

Additional pairs of macros can be added to the list using the **−a** option. This must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair .BS and .ES, use

−a.BS.ES

The **−c** option defines commands that *checknr* would interpret otherwise as undefined.

The **−f** option requests *checknr* to ignore \f font changes.

The **−s** option requests *checknr* to ignore \s size changes.

DIAGNOSTICS

Checknr complains about unmatched delimiters, unrecognized commands, and bad syntax of commands.

EXAMPLES

The command:

checknr -f sorting

checks the file **sorting** for errors that involve mismatched opening and closing delimiters and unknown commands, but disregards errors caused by font changes.

WARNINGS

Checknr is intended for use on documents prepared with *checknr* in mind, much the same as *lint*. It expects a certain document writing style for \f and \s commands, in which each \fx is terminated with \fP and each \sx is terminated with \s0. While text formats properly when the next font or point size is specified instead of \fP or \s0, such practice produces complaints from *checknr*. If the user intends to run *checknr*, the \fP and \s0 delimiting conventions should be used.

There is no way to define a single-character macro name using **−a**.

Checknr does not recognize certain reasonable constructs, such as conditionals.

AUTHOR

Checknr was developed by the University of California, Berkeley.

SEE ALSO

checkeq(1), lint(1), nroff(1).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

chfn – change finger entry

SYNOPSIS

chfn [*loginname*]

DESCRIPTION

Chfn is used to change information about users. This information is used by the *finger*(1) program, among others. It consists of the user's "real life" name, location, office phone number, and home phone number. *Chfn* prompts the user for each field. Included in the prompt is a default value, which is enclosed in brackets. The default value is accepted simply by typing <return>. To enter a blank field, type the word "none". Below is a sample run:

Name [Tracy Simmons]:
Location (Ex: 47U-P5) []: 42L-P1
Office Phone (Ex: 1632) []: 1863
Home Phone (Ex: 9875432) [5551546]: none

Chfn allows phone numbers to be entered with or without hyphens.

It is a good idea to run *finger*(1) after running *chfn* to make sure everything is the way you want it.

The optional argument *loginname* is used to change another person's *finger*(1) information. This can only be done by the superuser.

WARNINGS

The encoding of the office and extension information is installation dependent.

For historical reasons, the user's name, etc., are stored in the **passwd** file. This is an inappropriate place to store the information.

Because two users may try to write the **passwd** file at once, a synchronization method was developed. On rare occasions, a message that the password file is "busy" will be printed. In this case, *chfn* sleeps for a while and then tries to write to the **passwd** file again.

AUTHOR

Chfn was developed by the University of California, Berkeley.

FILES

/etc/passwd
/etc/ptmp

SEE ALSO

finger(1), passwd(4).

NAME

chmod – change file mode

SYNOPSIS

chmod [**–A**] *mode file ...*

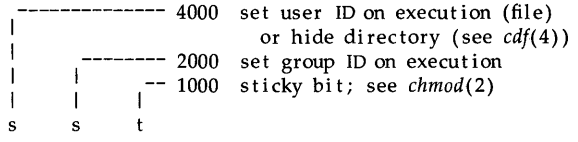
DESCRIPTION

Options

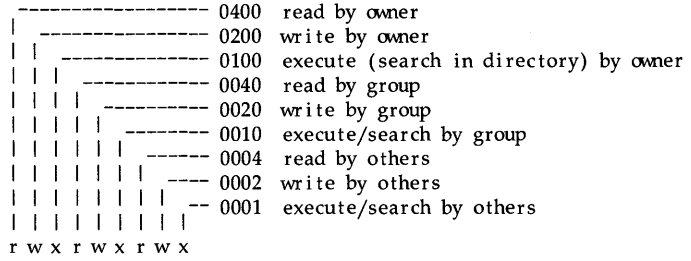
–A Preserve any optional access control list (ACL) entries associated with the file. (By default, in conformance with the IEEE Standard POSIX 1003.1-1988, optional ACL entries are deleted.) For information about access control lists, see *acl*(5).

The permissions of any named *file* are changed according to *mode*, which can be absolute or symbolic. An absolute *mode* is an octal number constructed from the logical OR of the following mode bits:

Miscellaneous mode bits:



Permission mode bits:



A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

Op can be + to add *permission* to the file's mode, – to take away *permission*, or = to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID), **H** (hide directory), and **t** (save text or sticky); **u**, **g** or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas can be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g** and **t** only works with **u**.

Only the owner of a file (or the super-user) can change its mode. Only the super-user can set the sticky bit. In order to set the group ID, the group of the file must correspond to your current group ID.

When using *chmod* on a symbolic link, the mode of the file referred to by the link is changed.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occured.

EXAMPLES

Deny write permission to others:

chmod o−w *file*

Make a file executable:

chmod +x *file*

Assign read and execute permission to everybody, and sets the set-user-ID bit:

chmod 4555 *file*

Assign read and write permission to the file owner, and read permission to everybody else:

chmod 644 *file*

DEPENDENCIES

RFA and NFS

The **−A** option is not supported for networked files.

HP Clustered Environment

The absolute *mode* of 4000 also serves to hide a directory. For symbolic *mode*, the *permission* letter **H** is used to hide a directory (see *cdf*(4)).

AUTHOR

Chmod was developed by AT&T and HP.

SEE ALSO

chacl(1), ls(1), chmod(2), cdf(4), acl(5).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

chmod: SVID2, XPG2, XPG3

NAME

chown, chgrp – change file owner or group

SYNOPSIS

chown *owner file ...*

chgrp *group file ...*

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner can be either a decimal user ID or a login name found in the password file.

Chgrp changes the group ID of the *files* to *group*. The group can be either a decimal group ID or a group name found in the group file.

In order to change the owner or group, you must own the file or be the super-user. If either command is invoked by other than the super-user on a regular file, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

When using *chown* or *chgrp* on symbolic links, the owner or group of the symbolic link is changed.

Access Control Lists (ACLs)

Users can permit or deny specific individuals and groups to access a file by setting optional ACL entries in the file's access control list (see *acl*(5)). When using *chown* in conjunction with ACLs, if the new owner and/or group of a file does not have an optional ACL entry corresponding to *u.%* and/or *%.g* in the file's access control list, the file's access permission bits remain unchanged. However, if the new owner and/or group is already designated by an optional ACL entry of *u.%* and/or *%.g* in the file's ACL, *chown* sets the corresponding file access permission bits (and the corresponding base ACL entries) to the permissions contained in that entry.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occured.

EXAMPLES

The following command changes the owner of the file *jokes* to *sandi:*

chown sandi jokes

To execute this command, you must be either the owner of *jokes* or the superuser.

FILES

/etc/group
/etc/passwd

SEE ALSO

chmod(1), chown(2), group(4), passwd(4), acl(5).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

chown: SVID2, XPG2, XPG3

chgrp: SVID2, XPG2, XPG3

NAME

chsh – change default login shell

SYNOPSIS

chsh *name* [*shell*]

DESCRIPTION

Chsh is a command similar to *passwd*(1), but is used to change the login shell field of the password file rather than the password entry. Any of **/bin/sh**, **/bin/rsh**, **/bin/csh**, **/bin/ksh**, **/bin/rksh** or **/bin/pam** can be specified as the shell. If unspecified, the shell reverts to the default login shell **/bin/sh**.

NETWORKING FEATURES

NFS

The file **/etc/passwd** can be implemented as a Yellow Pages database. *Chsh* can change only the local **/etc/passwd** file.

AUTHOR

Chsh was developed by the University of California, Berkeley.

SEE ALSO

csh(1), ksh(1), passwd(1), pam(1), sh(1), passwd(4).

NAME

ci – check in RCS revisions

SYNOPSIS

ci [*options*] *file* ...

DESCRIPTION

Ci stores new revisions into RCS files. Each file name ending in **",v"** is taken to be an RCS file; all others are assumed to be working files. *Ci* deposits the contents of each working file into the corresponding RCS file (see *rcsintro*(5)).

If the RCS file does not exist, *ci* creates it and deposits the contents of the working file as the initial revision. The default number is "1.1". The access list is initialized to empty. Instead of the log message, *ci* requests descriptive text (see the **–t** option below).

An RCS file created by *ci* inherits the read and execute permissions from the working file. If the RCS file exists, *ci* preserves its read and execute permissions. *Ci* always turns off all write permissions of RCS files.

The caller of the command must have read/write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself. A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. Because *ci* always creates a new RCS file and unlinks the old one, links to RCS files are useless.

For *ci* to work the caller's login must be in the access list, unless the access list is empty, or the caller is the owner of the file, or the caller is superuser.

Normally, *ci* checks whether the revision to be deposited is different from the preceding one. If it is not different, *ci* either aborts the deposit (if **–q** is given) or asks whether to abort (if **–q** is omitted). A deposit can be forced with the **–f** option.

For each revision deposited, *ci* prompts for a log message. The log message should summarize the change and must be terminated with a line containing a single "." or a control-D. If several files are being checked in, *ci* asks whether or not to reuse the log message from the previous file. If the standard input is not a terminal, *ci* suppresses the prompt and uses the same log message for all files, see the **–m** option below.

The number of the deposited revision can be given with any of the options **–r**, **–f**, **–k**, **–l**, **–u**, or **–q** (see the **–r** option below).

To add a new revision to an existing branch, the head revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file, unless locking is set to *strict* (see *rcs*(1)). A lock held by someone else may be broken with the *rcs*(1) command.

Options

–f[*rev*] Forces a deposit. The new revision is deposited even if it is not different from the preceding one.

–k[*rev*] Searches the working file for keyword values to determine its revision number, creation date, author, and state (see *co*(1)), and assigns these values to the deposited revision, rather than computing them locally. A revision number given with a command option overrides the number in the working file. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the **–k** option at these sites to preserve its original number, date, author, and state.

–l[*rev*] Works like **–r**, except it performs an additional *co* **–l** for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue

editing it after the check in.

−**m***msg* Uses the string *msg* as the log message for all revisions checked in.

−**n***name* Assigns the symbolic name *name* to the checked-in revision. *Ci* prints an error message if *name* is already assigned to another number.

−**N***name* Same as −**n**, except that it overrides a previous assignment of *name*.

−**q**[*rev*] Quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless −**f** is given.

−**r**[*rev*] Assigns the revision number *rev* to the checked-in revision, releases the corresponding lock, and deletes the working file. This is the default.

If *rev* is omitted, *ci* derives the new revision number from the caller's last lock. If the caller has locked the head revision of a branch, the new revision is added to the head of that branch and a new revision number is assigned to the new revision. The new revision number is obtained by incrementing the head revision number. If the caller locked a non-head revision, a new branch is started at the locked revision, and the number of the locked revision is incremented. The default initial branch and level numbers are 1. If the caller holds no lock, but is the owner of the file and locking is not set to *strict*, the revision is added to the head of the trunk.

If *rev* indicates a revision number, it must be higher than the latest one on the branch to which *rev* belongs, or must start a new branch.

If *rev* indicates a branch instead of a revision, the new revision is added to the head of that branch. The level number is obtained by incrementing the head revision number of that branch. If *rev* indicates a non-existing branch, that branch is created with the initial revision numbered *rev*.1.

NOTE: On the trunk, revisions can be added to the head, but not inserted.

−**s***state* Sets the state of the checked-in revision to the identifier *state*. The default is **Exp**.

−**t**[*txtfile*] Writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, *ci* prompts the user for text from standard input, terminated with a line containing a single "." or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. During initialization, descriptive text is requested even if −**t** is not given. The prompt is suppressed if standard input is not a terminal.

−**u**[*rev*] Works like −**l**, except that the deposited revision is not locked. This is useful if one wants to process (e.g., compile) the revision immediately after check in.

Access Control Lists (ACLs)

Optional ACL entries should not be added to RCS files, because they might be deleted.

DIAGNOSTICS

For each revision, *ci* prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status always refers to the last file checked in, and is 0 if the operation was successful, 1 if unsuccessful.

EXAMPLES

If the current directory contains a subdirectory **RCS** with an RCS file **io.c,v**, all of the following commands deposit the latest revision from **io.c** into **RCS/io.c,v**.

```
ci io.c
```

```
ci RCS/io.c,v
ci io.c,v
ci io.c RCS/io.c,v
ci io.c io.c,v
ci RCS/io.c,v io.c
ci io.c,v io.c
```

WARNINGS

The names of RCS files are generated by appending **,v** to the end of the working file name. If the resulting RCS file name is too long for the file system on which the RCS file should reside, *ci* terminates with an error message.

The log message may not exceed 2046 bytes.

A file with approximately 240 revisions may cause a hashtable overflow. *Ci* cannot add another revision to the file until some of the old revisions have been removed. Use the *rcs* command with the **−o** (obsolete) option to remove old revisions.

AUTHOR

Ci was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

co(1), ident(1), rcs(1), rcsdiff(1), rcsintro(5), rcsmerge(1), rlog(1), rcsfile(4), acl(5).

NAME

clear – clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It reads the **TERM** environment variable for the terminal type and then reads the appropriate terminfo data base to figure out how to clear the screen.

FILES

/usr/lib/terminfo/?/* terminal database files

AUTHOR

Clear was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

terminfo(4).

NAME

cmp − compare two files

SYNOPSIS

cmp [**−l**] [**−s**] file1 file2

DESCRIPTION

The two files are compared. (If *file1 or file2* is −, the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:

−l Print the byte number (decimal) and the differing bytes (octal) for each difference (byte numbering begins at 1 rather than 0).

−s Print nothing for differing files; return codes only.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cmp* behaves as if all internationalization variables are set to "C". See *environ(5).*

International Code Set Support

Single- and multi-byte character code sets are supported.

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

STANDARDS CONFORMANCE

cmp: SVID2, XPG2, XPG3

NAME

cnodes – display information about specified cluster nodes

SYNOPSIS

cnodes [**–almnrsxAC1**] [*name* ...]

DESCRIPTION

Cnodes displays information about the cluster nodes described in the **/etc/clusterconf** file for clustered systems. When no argument is given, all currently active cluster nodes in the cluster are listed. When one or more *name* is given, information about each named cluster node is displayed.

There are two major listing formats. The format chosen depends on whether the output is going to a terminal, and may also be controlled by option flags. The default format for a terminal is to list cluster node names in multi-column format. If the standard output is not a terminal, the default format is to list one cluster node per line.

In order to determine output formats for multi-column output, *cnodes* uses the environment variable **COLUMNS** to determine the number of character positions available on the output line. If this variable is not set, the *terminfo* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns is assumed.

Options

The options are as follows:

–a List all specified cluster nodes whether or not the nodes are clustered. If no *name* is given, *cnodes* lists all entries in the **/etc/clusterconf** file. Cluster nodes not currently clustered are displayed with an asterisk (*****) following the cluster node name.

–l List cluster nodes in long format, giving cluster node name, cnode ID, the cnode's swap server and whether or not the cnode is a root server. If the output is going to a terminal, a header is also displayed.

–m List information about the local cluster node only. If **–m** is specified, any *name* arguments are ignored.

–n List cluster node IDs instead of cluster node names.

–r Display information about the cluster's root server only. If **–r** is specified, any *name* arguments are ignored.

–s Suppress all output. This option is useful for testing the exit value.

–x Do not display information about the local cluster node.

–A Same as **–a**, except cluster nodes currently not clustered will not have an asterisk appended to their names.

–C Force multi-column output.

–1 Force single column output.

RETURN VALUE

Cnodes exits with a value of **0** if the local machine is a member of a cluster, and **1** if not.

DIAGNOSTICS

Cnodes writes "Machine is not clustered" to the standard error output if the specified machine is not currently a member of a cluster and neither the **–a** nor the **–s** option was specified.

"Machine does not exist" is written to the standard error output if the specified machine is not listed in **/etc/clusterconf** and **–s** was not specified.

AUTHOR

Cnodes was developed by HP.

FILES

/etc/clusterconf

SEE ALSO

ccck(1M), cnodeid(2), cnodes(2), terminfo(4), clusterconf(4).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cnodes* behaves as if all internationalization variables are set to "C". See *environ*(5).

NAME

co – check out RCS revisions

SYNOPSIS

co [*options*] *file ...*

DESCRIPTION

Co retrieves revisions from RCS files. Each file name ending in **",v"** is taken to be an RCS file. All other files are assumed to be working files. *Co* retrieves a revision from each RCS file and stores it in the corresponding working file (see also *rcsintro*)(5).

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later checked in must normally be locked. Locking a revision currently locked by another user fails. (A lock may be broken with the *rcs*(1) command.) *Co* with locking requires the caller to be on the access list of the RCS file, unless he is the owner of the file or the superuser, or the access list is empty. *Co* without locking is not subject to access list restrictions.

A revision is selected by number, check in date/time, author, or state. If none of these options are specified, the latest revision on the trunk is retrieved. When the options are applied in combination, the latest revision that satisfies all of them is retrieved. The options for date/time, author, and state retrieve a revision on the selected branch. The selected branch is either derived from the revision number (if given), or is the highest branch on the trunk. A revision number may be attached to the options **−l**, **−p**, **−q**, or **−r**.

The caller of the command must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory that contains the RCS file.

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless the file is checked out unlocked and locking is set to *strict* (see *rcs*(1)).

If a file with the name of the working file exists already and has write permission, *co* aborts the check out if **−q** is given, or asks whether to abort if **−q** is not given. If the existing working file is not writable, it is deleted before the check out.

A number of temporary files are created. A semaphore file is created in the directory of the RCS file to prevent simultaneous update.

A *co* command applied to an RCS file with no revisions creates a zero-length file. *Co* always performs keyword substitution (see below).

Options

−l[*rev*] Locks the checked out revision for the caller. If omitted, the checked out revision is not locked. See option **−r** for handling of the revision number *rev*.

−p[*rev*] Prints the retrieved revision on the standard output rather than storing it in the working file. This option is useful when *co* is part of a pipe.

−q[*rev*] Quiet mode; diagnostics are not printed.

−d*date* Retrieves the latest revision on the selected branch whose check in date/time is less than or equal to *date*. The date and time may be given in free format and are converted to local time. Examples of formats for *date*:

Tue−PDT, 1981, 4pm Jul 21	(free format)
Fri April 16 15:52:25 EST 1982	(output of *ctime*(3C))
86/4/21 10:30am	(format: yy/mm/dd hh:mm:ss)

Most fields in the date and time may be defaulted. *Co* determines the defaults in the order year, month, day, hour, minute, and second (from most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date "20, 10:30" defaults to 10:30:00 of the 20th of the current month and current year. Date/time fields may be delimited by spaces or commas. If spaces are used, the string must be surrounded by double quotes.

−r[*rev*] Retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. *Rev* is composed of one or more numeric or symbolic fields separated by ".". The numeric equivalent of a symbolic field is specified with the **−n** option of the commands *ci*(1) and *rcs*(1).

−s*state* Retrieves the latest revision on the selected branch whose state is set to *state*.

−w[*login*] Retrieves the latest revision on the selected branch that was checked in by the user with login name *login*. If the argument *login* is omitted, the caller's login is assumed.

−j*joinlist* Generates a new revision that is the result of the joining of the revisions on *joinlist*. *Joinlist* is a comma-separated list of pairs of the form *rev2:rev3*, where *rev2* and *rev3* are (symbolic or numeric) revision numbers. For the initial pair, *rev1* denotes the revision selected by the options **−l**, ..., **−w**. For all other pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, *co* joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is particularly useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1* < *rev2* < *rev3* on the same branch, joining generates a new revision that is similar to *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap with changes from *rev2* to *rev3*, *co* prints a warning and includes the overlapping sections, delimited by the lines **<<<<<<<rev1,** **=======**, and **>>>>>>>rev3.**

For the initial pair, *rev2* may be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. If the option **−l** is present, the initial *rev1* is locked.

Keyword Substitution

Strings of the form *$keyword$* and *$keyword:...$* embedded in the text are replaced with strings of the form *$keyword: value $*, where *keyword* and *value* are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form *$keyword$*. On check out, *co* replaces these strings with strings of the form *$keyword: value $*. If a revision containing strings of the latter form is checked back in, the value fields will be replaced during the next check out. Thus, the keyword values are automatically updated on check out.

Keywords and their corresponding values:

$Author$ The login name of the user who checked in the revision.

$Date$ The date and time the revision was checked in.

$Header$ A standard header containing the RCS file name, the revision number, the date, the author, and the state.

$Locker$	The login name of the user who locked the revision (empty if not locked).
Log	The log message supplied during check in, preceded by a header containing the RCS file name, the revision number, the author, and the date. Existing log messages are NOT replaced. Instead, the new log message is inserted after *$Log:...$*. This is useful for accumulating a complete change log in a source file.
$Revision$	The revision number assigned to the revision.
$Source$	The full pathname of the RCS file.
$State$	The state assigned to the revision with *rcs* **−s** or *ci* **−s**.

Access Control Lists (ACLs)

Optional ACL entries should not be added to RCS files because they might be deleted.

DIAGNOSTICS

The RCS file name, the working file name, and the revision number retrieved are written to the diagnostic output. The exit status always refers to the last file checked out, and is 0 if the operation was successful, 1 if unsuccessful.

EXAMPLES

Suppose the current directory contains a subdirectory named **RCS** with an RCS file named **io.c,v**. Each of the following commands retrieves the latest revision from **RCS/io.c,v** and stores it into **io.c**:

```
co io.c
co RCS/io.c,v
co io.c,v
co io.c RCS/io.c,v
co io.c io.c,v
co RCS/io.c,v io.c
co io.c,v io.c
```

WARNINGS

The *co* command generates the working file name by removing the **,v** from the end of the RCS file name. If the given RCS file name is too long for the file system on which the RCS file should reside, *co* terminates with an error message.

There is no way to suppress the expansion of keywords, except by writing them differently. In *nroff*(1) and *troff*, this is done by embedding the null-character "\&" into the keyword.

The option **−d** gets confused in some circumstances, and accepts no date before 1970.

The option **−j** does not work for files that contain lines with a single ".".

AUTHOR

Co was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

ci(1), ident(1), rcs(1), rcsdiff(1), rcsintro(5), rcsmerge(1), rlog(1), rcsfile(4), acl(5).

NAME

col – filter reverse line-feeds and backspaces

SYNOPSIS

col [**–blfxp**]

DESCRIPTION

Col reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code **ESC-7**), and by forward and reverse half-line feeds (**ESC-9** and **ESC-8**). *Col* is particularly useful for filtering multicolumn output made with the **.rt** command of *nroff*(1) and output resulting from use of the *tbl*(1) preprocessor.

If the **–b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

If the **–l** option is given, *col* assumes the output device is a line printer (rather than a character printer) and removes backspaces in favor of multiply overstruck full lines. It generates the minimum number of print operations necessary to generate the required number of overstrikes. (All but the last print operation on a line are separated by carriage returns (\r); the last print operation is terminated by a newline (\n).)

Although *col* accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the **–f** (fine) option; in this case, the output from *col* may contain forward half-line feeds (**ESC-9**), but will still never contain either kind of reverse line motion.

Unless the **–x** option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters **SO** (\016) and **SI** (\017) are assumed by *col* to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output **SI** and **SO** characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, **SI**, **SO**, **VT** (\013), and **ESC** followed by **7**, **8**, or **9**. The **VT** character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, *col* will ignore any unrecognized escape sequences found in its input; the **–p** option may be used to cause *col* to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

EXAMPLES

The *col* command is used most often with *nroff*(1) and *tbl*(1). A common usage is:

tbl *filename* **| nroff -man | col | more -s**

This command allows the vertical bars to be printed for tables. The file is run through the *tbl*(1) command, and the output is then piped through *nroff*(1), formatting the output using the **–man** macros. The formatted output is then piped through *col*, which sets up the vertical bars and aligns the columns in the file. The file is finally piped through the *more*(1) command, which prints the output to the screen. The **–s** option deletes excess space from the output so that multiple blank lines are not printed to the screen.

SEE ALSO

nroff(1), tbl(1).

NOTES

The input format accepted by *col* matches the output produced by *nroff* with either the **−T37** or **−Tlp** options. Use **−T37** (and the **−f** option of *col*) if the ultimate disposition of the output of *col* will be a device that can interpret half-line motions, and **−Tlp** otherwise.

BUGS

Cannot back up more than 128 lines.

There is a maximum limit for the number of characters, including backspaces and overstrikes, on a line. The maximum limit is at least 800 characters.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *col* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

col: SVID2, XPG2, XPG3

NAME

comb – combine SCCS deltas

SYNOPSIS

comb [−p*sid*] [−c*list*] [−**o**] [−**s**] *file* ...

DESCRIPTION

Comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored. The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

−p*sid* — The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

−c*list* — A *list* (see *get*(1) for the syntax of a *list*) of deltas to be preserved. All other deltas are discarded.

−**o** — For each **get** −**e** generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the −**o** keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

−**s** — This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original – combined) / original

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

DIAGNOSTICS

Use *help*(1) for explanations.

EXAMPLES

The following command:

```
comb -c1.1,1.3,1.6 s.document > save_file
```

creates a shell script named **save_file**, which if executed, creates a new **s.document** using only the deltas **1.1**, **1.3**, and **1.6** from the old **s.document**. The script overwrites the old **s.document**; thus, is might be desireable to copy it elsewhere. Thus, the following might be executed:

```
cp s.document s.save
comb -c1.1,1.3,1.6 s.document > save_file
sh save_file
```

WARNINGS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is

possible for the reconstructed file to actually be larger than the original.

FILES

s.COMB?????	Temporary file
comb?????	Temporary file

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sh(1), sccsfile(4).

Source Code Control System User Guide in the *HP-UX User's Guide*.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [– [**123**]] *file1 file2*

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in increasing collating sequence (see *sort*(1) and Environment Variables below), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name – means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm –12** prints only the lines common to the two files; **comm –23** prints only lines in the first file but not in the second; **comm –123** is a no-op.

EXAMPLES

The following command prints all lines common to the files **green** and **yellow**:

comm -12 green yellow

It is assumed that the contents of **green** and **yellow** have been ordered in the collating sequence defined by the LC_COLLATE or LANG environment variable.

SEE ALSO

cmp(1), diff(1), sdiff(1), sort(1), uniq(1).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence *comm* expects from the input files.

LANG determines the language in which messages are displayed.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *comm* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

STANDARDS CONFORMANCE

comm: SVID2, XPG2, XPG3

NAME

compact, uncompact, ccat – compact and uncompact files, and cat them

SYNOPSIS

compact [*name* ...]
uncompact [*name* ...]
ccat [*file* ...]

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted to the standard output. *Compact* operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to attach a decoding tree in front of the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, *compact* and *uncompact* can operate as filters. In particular,

... | compact | uncompact | ...

operates as a (very slow) no-op.

When an argument *file* is given, it is compacted and the resulting file is placed in *file.C; file* is unlinked. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

Uncompact restores the original file from a file compressed by *compact*. If no file names are given, the standard input is uncompacted to the standard output.

Ccat cats the original file from a file compressed by *compact*, without uncompressing the file.

Access Control Lists (ACLs)

On systems that implement access control lists, when a new file is created with the effective user and group ID of the caller, the original file's ACL is copied to the new file after being altered to reflect any change in ownership (see *acl*(5)).

WARNINGS

The last segment of the file name must contain fewer than thirteen characters to allow space for the appended **.C**.

DEPENDENCIES

RFA and NFS

Access control list entries of networked files are summarized (as returned in *st_mode* by *stat*(2)), but not copied to the new file.

FILES

*.C compacted file created by compact, removed by uncompact

SEE ALSO

compress(1), pack(1), acl(5).
Gallager, Robert G., "Variations on a Theme of Huffman," *I.E.E.E. Transactions on Information Theory*, vol. IT-24, no. 6, November 1978, pp. 668 - 674.

AUTHOR

Colin L. Mc Master

NAME

compress, uncompress, zcat – compress and expand data

SYNOPSIS

compress [**−d**] [**−f**] [**−v**] [**−c**] [**−V**] [**−b** *maxbits*] [*name...*]
uncompress [**−f**] [**−v**] [**−c**] [**−V**] [*name ...*]
zcat [**−V**] [*name ...*]

DESCRIPTION

Compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with the extension **.Z,** while keeping the same ownership, modes, access and modification times. If no files are specified, the standard input is compressed to the standard output. Compressed files can be restored to their original form using *compress* **−d** or *uncompress* or *zcat*.

The amount of compression obtained depends on the size of the input, the maximum number of bits (*maxbits*) per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50–60%. Compression is generally much better than that achieved by Huffman coding (as used in *pack*), or adaptive Huffman coding (*compact*), and takes less time to compute.

Exit status is normally 0; if the last file is larger after (attempted) compression, the status is 2; if an error occurs, exit status is 1.

Options

The following options are available:

−d Decompress *name*. *Compress* **−d** is equivalent to *uncompress*.

−f Force compression of *name*. This is useful for compressing an entire directory, even if some of the files do not actually shrink. If **−f** is not given and *compress* is run in the foreground, the user is prompted as to whether an existing file should be overwritten.

−v Print a message describing the percentage of reduction for each file compressed.

−c Force *compress* and *uncompress* to write to the standard output; no files are changed. The nondestructive behavior of *zcat* is identical to that of *uncompress* **−c**.

−V Print the current version and compile options onto the standard error.

−b *maxbits* Specify the maximum number of bits the *compress* algorithm will use. The default is 16 and the range can be any integer between 9 and 16.

Compress uses the modified Lempel-Ziv algorithm popularized in "A Technique for High Performance Data Compression," Terry A. Welch, *IEEE Computer*, vol. 17, no. 6 (June 1984), pp. 8-19. Common substrings in the file are first replaced by 9-bit codes 257 and up. When code 512 is reached, the algorithm switches to 10-bit codes and continues to use more bits until the limit specified by the **−b** flag is reached (default 16).

After the *maxbits* limit is attained, *compress* periodically checks the compression ratio. If it is increasing, *compress* continues to use the existing code dictionary. However, if the compression ratio is decreasing, *compress* discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next "block" of the file.

Note that the **−b** flag is omitted for *uncompress* since the *maxbits* parameter specified during compression is encoded within the output, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is attempted.

Access Control Lists

Compress retains a file's access control list when compressing and expanding data.

DIAGNOSTICS

Usage: compress [−dfvcV] [−b maxbits] [file ...]
: Invalid options were specified on the command line.

Missing maxbits
: *Maxbits* must follow **−b**.

file: not in compressed format
: The file specified to *uncompress* has not been compressed.

file: compressed with *xx* bits, can only handle *yy* bits
: *File* was compressed by a program that could deal with a higher value of *maxbits* than the compress code on this machine. Recompress the file with a lower value of *maxbits*.

file: already has .Z suffix -- no change
: The file is assumed to be already compressed. Rename the file and try again.

file: filename too long to tack on .Z
: The output file name, which is the source file name with a **.Z** extension, is too long for the file system on which the source file resides. Make the source file name shorter and try again.

file already exists; do you wish to overwrite (y or n)?
: Respond "y" if you want the output file to be replaced; "n" if not.

uncompress: corrupt input
: A SIGSEGV violation was detected which usually means that the input file has been corrupted.

Compression: *xx.xx%*
: Percentage of the input saved by compression. (Relevant only for **−v**.)

-- not a regular file: unchanged
: When the input file is not a regular file (for example, a directory), it is left unaltered.

-- has *xx* other links: unchanged
: The input file has links; it is left unchanged. See *ln* on *cp*(1) for more information.

-- file unchanged
: No savings is achieved by compression. The input remains virgin.

EXAMPLES

The command:

compress -v zenith

compresses the file **zenith** and might print the following information to the screen:

zenith: Compression: 23.55% -- replaced with zenith.Z

indicating that 23.55% of the room was saved, or

zenith: Compression: -12.04% -- file unchanged

indicating that an extra 12.04% space must be used to compress the file.

To undo the compression, type:

uncompress zenith.Z

or

compress -d zenith.Z

This returns the file **zenith.Z** to its original uncompressed form and name.

WARNINGS

Although compressed files are compatible between machines with large memory, **−b12** should be used for file transfer to architectures with a small process data space (64KB or less).

RFA and NFS

Access control lists of networked files are summarized (as returned in *st_mode* by *stat*(2)), but not copied to the new file.

AUTHOR

Compress was developed by Joseph M. Orost, Kenneth E. Turkowski, Spencer W. Thomas, and James A. Woods.

FILES

*.Z Compressed file created by *compress* and removed by *uncompress*.

SEE ALSO

compact(1), pack(1), acl(5).

NAME

cp – copy files and directory subtrees

SYNOPSIS

cp *file1 new_file*
cp *file1* [*file2* ...] *dest_directory*
cp **–r** *directory1* [*directory2* ...] *dest_directory*

DESCRIPTION

Cp copies:

- *file1* to new or existing *new_file*,
- *file1* to existing *dest_directory*,
- *file1*, *file2*, ... to existing *dest_directory*,
- directory subtree *directory1*, to new or existing *dest_directory*. or
- multiple directory subtrees *directory1*, *directory2*, ... to new or existing *dest_directory*.

Under no circumstance can *file1* and *new_file* be the same (be cautious when using shell metacharacters). When destination is a directory, one or more files are copied into that directory. If two or more files are copied, the destination must be a directory. When copying a single file to a new file, if *new_file* exists, its contents are destroyed.

If the access permissions of the destination *dest_directory* or existing destination file *new_file* forbid writing, *cp* aborts and produces an error message "cannot create *file*".

To copy one or more directory subtrees to another directory, the **–r** option is required. The **–r** option is ignored if used when copying a file to another file or files to a directory.

If *new_file* is a link to an existing file with other links, *cp* overwrites the existing file and retains all links. If copying a file to an existing file, *cp* does not change existing file access permission bits, owner, or group.

When copying files to a directory or to a new file that does not already exist, *cp* creates a new file with the same file miscellaneous bits as *file1*, except that the sticky bit is not set unless you are superuser (see *chmod*(2)). The owner and group of the new file or files are those of the user. The last modification time of *new_file* (and last access time, if *new_file* did not exist) and the last access time of the source *file1* are set to the time the copy was made.

Options

–r Cause *cp* to copy the subtree rooted at each source directory to *dest_directory*. If *dest_directory* exists, it must be a directory, in which case *cp* creates a directory within *dest_directory* with the same name as *file1* and copies the subtree rooted at *file1* to *dest_directory/file1*. An error occurs if *dest_directory/file1* already exists. If *dest_directory* does not exist, *cp* creates it and copies the subtree rooted at *file1* to *dest_directory*. Note that **cp** **–r** cannot merge subtrees.

Only normal files and directories are copied; character special devices, block special devices, network special files, named pipes, symbolic links, and sockets are not copied, and a warning is printed stating that the file was skipped. *dest_directory* should not reside within *directory1*, nor should *directory1* have a cyclic directory structure, since in both cases *cp* attempts to copy an infinite amount of data.

Access Control Lists (ACLs)

If *new_file* is a new file, or if a new file is created in *dest_directory*, it inherits the access control list of the original *file1*, *file2*, etc., altered to reflect any difference in ownership between the two

files (see *acl*(5)).

EXAMPLES

The following command moves the directory *sourcedir* and its contents to a new location (*targetdir*) in the file system. Since *cp* creates the new directory, the destination directory *targetdir* should not already exist.

cp -r *sourcedir targetdir* **&& rm -rf** *sourcedir*

The **-r** option copies the subtree (files) in the directory *sourcedir* to the directory *targetdir*. The double ampersand (**&&**) causes a conditional action. If the operation on the left side of the **&&** is successful, the right side is executed (and removes the old directory). If the operation on the left of the **&&** is not successful, the old directory is not removed.

This example is equivalent to:

mv *sourcedir targetdir*

To copy all files and directory subtrees in the current directory to existing *targetdir*, use:

cp -r * *targetdir*

To copy all files and directory subtrees in *sourcedir* to *targetdir*, use:

cp -r *sourcedir*/* *targetdir*

Note that directory pathnames can precede both *sourcedir* and *targetdir*.

DEPENDENCIES

RFA and NFS

Access control lists of networked files are summarized (as returned in *st_mode* by *stat*(2)), but not copied to the new file. When using *mv* or *ln* on such files, a + is not printed after the mode value when asking for permission to overwrite a file.

AUTHOR

Cp was developed by AT&T, the University of California, Berkeley and HP.

SEE ALSO

cpio(1), ln(1), mv(1), rm(1), link(1M), lstat(2), readlink(2), stat(2), symlink(2), symlink(4), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG and LC_CTYPE determine the local language equivalent of y (for yes/no queries).

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cp* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

cp: SVID2, XPG2, XPG3

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio **–o** [**aABcxvCh**]

cpio **–i** [**BdcrtuxvmfPsSb6R**] [*patterns*]

cpio **–p** [**aduxvlmr**] *directory*

DESCRIPTION

Cpio **–o** (copy out) reads the standard input to obtain a list of path names and copies those files to the standard output together with path name and status information. Output is padded to a 512-byte boundary.

Cpio **–i** (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio** **–o**. Only files with names that match *patterns*, according to the rules of Pattern Matching Notation (see *regexp*(5)), are selected. In addition, a leading **!** within a pattern indicates that only those names should be selected that do *not* match the remainder of the pattern. Multiple *patterns* can be specified. If no *patterns* are specified, the default for *patterns* is * (that is, select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files match the permissions of the original files when the archive was created by **cpio** **–o**. The owner and group of the files are that of the current user unless the user is super-user, in which case *cpio* retains the owner and group of the files of the previous **cpio** **–o**.

Cpio **–p** (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below. Destination path names are interpreted relative to the named *directory*.

Options

a Reset access times of input files after they are copied.

A Suppress warning messages regarding optional access control list entries. *Cpio*(1) does not backup optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file having optional access control list entries.

B Block input/output at 5,120 bytes to the record (does not apply to the **cpio** **–p** option); meaningful only with data directed to or from devices that support variable length records such as magnetic tape.

d Create directories as needed.

c Write or read header information in ASCII character form for portability.

r Rename files interactively. If the user types a null line, the file is skipped.

t Print only a table of contents of the input. No files are created, read, or copied.

u Copy unconditionally (normally, an older file will not replace a newer file with the same name).

x Save or restore device special files. Since *mknod*(2) is used to recreate these files on a restore, **–ix** and **–px** can only be used by the super-user. Restoring device files onto a different system can be very dangerous. This is intended for intrasystem (backup) use.

v Verbose: cause a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls** **–l** command (see *ls*(1)).

l Whenever possible, link files rather than copying them. This option does not destroy existing files. Usable only with the **−p** option.

m Retain previous file modification time. This option is ineffective on directories that are being copied.

f Copy in all files except those in *patterns.*

P Read a file written on a PDP−11 or VAX system (with byte swapping) that did not use the **−c** option. Only useful with **−i** (copy in). Files copied in this mode are not changed. Non-ASCII files are likely to need further processing to be readable. This processing often requires knowledge of the content of the file and thus cannot always be done by this program. (PDP−11 and VAX are registered trademarks of Digital Equipment Corporation). The **−s , −S** and **−b** options below can be used when swapping all the bytes on the tape (rather than just the headers) is appropriate. In general, text is best processed with **-P** and binary data with one of the other options.

s Swap all bytes of the file. Use only with the **−i** option.

S Swap all halfwords of the file. Use only with the **−i** option.

b Swap both bytes and halfwords. Use only with the **−i** option.

6 Process a UNIX Sixth Edition format file. Only useful with **−i** (copy in).

R Resynchronize automatically when *cpio* goes "Out of phase," (see DIAGNOSTICS).

C Have *cpio* checkpoint itself at the start of each volume. If *cpio* is writing to a streaming tape drive with immediate report mode enabled and a write error occurs, it normally aborts and exits with return code **2**. With this option specified, *cpio* instead automatically restarts itself from the checkpoint and rewrites the current volume. Alternatively, if *cpio* is not writing to such a device and a write error occurs, *cpio* normally continues with the next volume. With this option specified, however, the user can choose to either ignore the error or rewrite the current volume.

h Follow symbolic links as though they were normal files or directories. Normally, *cpio* archives the link.

Note that *cpio* archives created using a raw device file must be read using a raw device file.

When the end of the tape is reached, *cpio* prompts the user for a new special file and continues.

If you want to pass one or more metacharacters to *cpio* without the shell expanding them, be sure to precede each of them with a backslash (\).

Device files written with the **−ox** option (for example, **/dev/tty03**) do not transport to other implementations of HP-UX.

DIAGNOSTICS

The diagnostic message "Out of phase" indicates that *cpio* could not successfully read its particular "magic number" in the header. Without the **R** option specified, *cpio* fails and returns an exit code of **2**. With the **R** option, *cpio* attempts to resync automatically. (Resyncing means that *cpio* tries to find the next good header in the archive and continues processing from there.) If *cpio* tries to resynchronize from being "Out of phase", it returns an exit code of **1**. If resynchronization fails, try changing header mode (**−c** option) or byte swapping the header (**−P** or **−s** options).

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates

a directory hierarchy:

ls | cpio −o >/dev/rmt/0m

cd olddir
find . −depth −print | cpio −pdi newdir

The trivial case "find . −depth −print | cpio −oB >/dev/rmt/0m" can be handled more efficiently by:

find . −cpio /dev/rmt/0m

WARNINGS

Do not redirect the output of *cpio* to a named *cpio* archive file residing in the same directory as the original files belonging to that *cpio* archive. This can cause loss of data.

Path names are restricted to PATH_MAX characters (see <**limits.h**> and *limits*(5)). If there are too many unique linked files, the program runs out of memory to keep track of them, and thereafter, linking information is lost. Only the super-user can copy special files.

Cpio tapes written on HP machines with the **−ox[c]** options might mislead (non-HP) versions of *cpio* that do not support the **−x** option. If a non-HP (and non-AT&T) version of *cpio* happens to be modified so that (HP) *cpio* recognizes it as a device special file, a spurious device file might be created.

If **/dev/tty** is not accessible, *cpio* issues a complaint and exits.

The **−pd** option does not create the directory typed on the command line.

The **−idr** option does not make empty directories.

The **−plu** option does not link files to existing files.

POSIX defines a file named **TRAILER!!!** as an end-of-archive marker. Consequently, if a file of that name is contained in a group of files being written by *cpio* **−o**, the file is interpreted as end-of-archive, and no remaining files are copied. Recommended practice is to avoid naming files anything that resembles an end-of-archive file name.

DEPENDENCIES

General

The use of *cpio* with cartridge tape units requires additional comments. For an explanation of the constraints on cartridge tapes, see *ct*(7).

Warning: using *cpio* to write directly to a cartridge tape unit can severely damage the tape drive in a short amount of time, and is therefore strongly discouraged. The recommended method of writing to the cartridge tape unit is to use *tcio*(1) in conjunction with *cpio* (note that **−B** must not be used when *tcio*(1) is used) because *tcio*(1) buffers data into larger pieces suitable for cartridge tapes.

The **−B** option must be used when writing directly (that is, without using *tcio*(1)) to a CS/80 cartridge tape unit.

At and before release 2.2 on the Series 300, this system wrote a format which, when crossing media boundaries on some kinds of disks, differs from the format specified by System V.2 (although it matched that written by System III). The program **/etc/ocpio** reads and writes this format and has essentially the same features as *cpio* except that options **−S**, **−b** and **-f** are omitted. However, **/etc/ocpio** is considered obsolescent.

Series 800

The **−A** option is not supported.

SEE ALSO

ar(1), find(1), tar(1), tcio(1), cpio(4), acl(5), environ(5), lang(5), regexp(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in pattern matching notation.

LC_TIME determines the format and content of date and time strings output when listing the contents of an archive with the **−v** option.

LANG determines the language in which messages are displayed.

If LC_COLLATE, LC_CTYPE or LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cpio* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

cpio: SVID2, XPG2, XPG3

NAME

cpp – the C language preprocessor

SYNOPSIS

/lib/cpp [option ... **] [** ifile **[** ofile **]]**

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. Its purpose is to process **include** and conditional compilation instructions, and macros. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc*(1) command, since the functionality of *cpp* may someday be moved elsewhere. See *m4*(1) for a general macro processor.

Cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following options to *cpp* are recognized:

–P Preprocess the input without producing the line control information used by the next pass of the C compiler.

–C By default, *cpp* strips C-style comments. If the **–C** option is specified, all comments (except those found on *cpp* directive lines) are passed along.

–U*name* Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system	unix, __unix
hardware	hp9000s800, hp9000s500, hp9000s300, hp9000s200, hp9000ipc, __hp9000s800 __hp9000s300, hppa, __hppa
UNIX systems variant	PWB, __PWB, hpux, __hpux, _HPUX_SOURCE
lint(1)	lint, __lint

In addition, all symbols that begin with an underscore and either an upper-case letter or another underscore are reserved. All HP–UX systems will have the symbols PWB, hpux, unix, __PWB, __hpux, and __unix defined. Each system will define at least one hardware variant, as appropriate. The lint symbols will be defined when *lint*(1) is running.

–A Remove all predefined symbols that begin with a letter and _HPUX_SOURCE. The user is expected to define _POSIX_SOURCE or _XOPEN_SOURCE when using this option.

–D*name*

–D*name=def* Define *name* as if by a **#define** directive. If no *=def* is given, *name* is defined as 1. The **–D** option has lower precedence than the **–U** option. That is, if the same name is used in both a **–U** option and a **–D** option, the name will be undefined regardless of the order of the options.

–T On HP-UX, preprocessor symbols are no longer restricted to eight characters. The **–T** option forces *cpp* to use only the first eight characters for distinguishing different preprocessor names. This behavior is the same as preprocessors on some other systems with respect to the length of names and is included for backward compatability.

−I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in will be searched for first in the directory of the file containing the **#include** line, then in directories named in **−I** options in left-to-right order, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the file containing the **#include** line is not searched. However, the directory *dir* is still searched.

−H*nnn* Change the internal macro definition table to be *nnn* bytes in size. The macro symbol table will be increased proportionately. The default table size is at least 128,000 bytes. This option serves to eliminate "too many defines" and "too much defining" errors.

Two special names are understood by *cpp*. The name **__LINE__** is defined as the current line number (as a decimal integer) as known by *cpp*, and **__FILE__** is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines begun by **#**. Any number of blanks and tabs are allowed between the # and the directive. The directives are:

#define *name token-string*
Replace subsequent instances of *name* with *token-string*. (*token-string* can be null).

#define *name*(*arg*, ..., *arg*) *token-string*
Notice that there can be no space between *name* and the **(**. Replace subsequent instances of *name* followed by a **(**, a list of comma-separated set of tokens, and a **)** by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of newly created *token-string*.

#undef *name* Cause the definition of *name* (if any) to be forgotten from now on.

#include
#include < *filename* >
Include at this point the contents of *filename* (which will then be run through *cpp*). See the **−I** option above for more detail.

#line *integer-constant* "*filename*"
Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If "*filename*" is not given, the current file name is unchanged.

#endif <*text*>
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. Any *text* occurring on the same line as the **#endif** is ignored and thus may be used to mark matching **#if-#endif** pairs. This makes it easier, when reading the source, to match **#if**, **#ifdef**, and **#ifndef** directives with their associated **#endif** directive.

#ifdef *name* The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name* The lines following will not appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*
Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary −, **!**, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined (** *name* **)** or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else Reverses the notion of the test directive which matches this directive. Thus if lines previous to this directive are ignored, the following lines will appear in the output, and vice versa.

The test directives and the possible **#else** directives can be nested. *Cpp* supports names up to 255 characters in length.

DIAGNOSTICS

The error messages produced by *cpp* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

DEPENDENCIES

Series 300

In the hardware *name* definition associated with predefined symbols (see **−U** option), two hardware variants are defined instead of one. Both hp9000s200 and hp9000s300 are present, and they are treated synonymously because of the similarity between the two series.

FILES

/usr/include standard directory for **#include** files

SEE ALSO

cc(1), m4(1).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of comments and string literals as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cpp* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NOTES

When new-line characters were found in argument lists for macros to be expanded, previous versions of *cpp* put out the new-lines as they were found and expanded. The current version of *cpp* replaces these new-lines with blanks to alleviate problems that the previous versions had when this occurred.

STANDARDS CONFORMANCE
cpp: SVID2, XPG2

NAME

crontab – user crontab file

SYNOPSIS

crontab [*file* **]**
crontab -r
crontab -l

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontab files (see *cron*(1M)). The **–r** option removes a user's crontab from the crontab directory. *Crontab* –l will list the crontab file for the invoking user.

Users are permitted to use *crontab* if their names appear in the file **/usr/lib/cron/cron.allow**. If that file does not exist, the file **/usr/lib/cron/cron.deny** is checked to determine if the user should be denied access to *crontab*. If neither file exists, only root is allowed to submit a job. If only **cron.deny** exists and is empty, global usage is permitted. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0–59),
hour (0–23),
day of the month (1–31),
month of the year (1–12),
day of the week (0–6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values), or a list of elements separated by commas. An element is either a number, or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, **0 0 1,15 * 1** would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, **0 0 * * 1** would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an initial argument of **sh**. Users who desire to have their *.profile* executed must explicitly do so in the crontab file. *Cron* supplies a default environment for every shell, defining **HOME**, **LOGNAME**, **SHELL(=/bin/sh)**, and **PATH(=:/bin:/usr/bin)**.

NOTE

Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users
/usr/spool/cron/crontabs	directory containing the crontab files
/usr/lib/cron/log	accounting information

SEE ALSO

sh(1), cron(1M), queuedefs(4).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

crontab: SVID2, XPG2, XPG3

NAME

crypt – encode/decode files

SYNOPSIS

crypt [*password*]

REMARKS

The decryption facilities provided by this software are under control of the United States Government and cannot be exported without special licenses. These capabilities are only available by special arrangement through HP.

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

The latter command decrypts the file and prints the clear version.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; "sneak paths" by which keys or clear text can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e., to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lowercase letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. The choice of keys and key security are the most vulnerable aspect of *crypt*.

EXAMPLES

The following example demonstrates the use of *crypt* to edit a file that the user wishes to keep strictly confidential:

```
$ crypt <plans >plans.x
key: violet
$ rm plans
...
$ vi -x plans.x
key: violet
...
:wq
$
...
$ crypt <plans.x | pr
key: violet
```

Note that the **–x** option is the encryption mode of *vi*, and prompts the user for the same key with which the file was encrypted.

WARNINGS

If output is piped to *nroff*(1) and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty*(1)).

If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(1), stty(1).

NAME

csh – a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**–cefinstvxTVX**] [*command file*] [*argument list ...*]

DESCRIPTION

Csh is a command language interpreter incorporating a command history buffer, a C-like syntax, and job control facilities.

The command options are interpreted as follows:

–c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

–e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

–f Suppress execution of the *.cshrc* file in your home directory, thus speeding up shell start-up time.

–i Forces *csh* to respond interactively when called from a device other than a computer terminal, such as another computer. *Csh* normally responds non-interactively. If *csh* is called from a computer terminal, it always responds interactively, no matter which options are selected.

–n This option causes commands to be parsed, but **not** executed. This may be used in syntactic checking of shell scripts. All substitutions are performed (history, command, alias, etc.).

–s Command input is taken from the standard input.

–t A single line of input is read and executed.

–v This option causes the *verbose* shell variable to be set. This causes command input to be echoed to your standard output device after history substitutions are made.

–x This option causes the *echo* shell variable to be set. This causes all commands to be echoed to the standard output immediately before execution.

–T This option disables the tenex features, which use the ESCAPE key for command/file name completion and control-D for listing available files (see the CSH UTILITIES section below)

–V This option causes the *verbose* variable to be set before *.cshrc* is executed. This means all *.cshrc* commands are also echoed to the standard output.

–X This option causes the *echo* variable to be set before *.cshrc* is executed. This means all *.cshrc* commands are also echoed to the standard output.

After processing the command options, if arguments remain in the argument list, and the **–c**, **–i**, **–s**, or **–t** options were not specified, the first remaining argument is taken as the name of a file of commands to be executed.

COMMANDS

A simple command is a sequence of words, the first of which specifies the command to be executed. A sequence of simple commands separated by vertical bar (|) characters forms a pipeline. The output of each command in a pipeline is made the input of the next command in the pipeline. Sequences of pipelines may be separated by semicolons (;), and are then executed sequentially. A sequence of pipelines may be executed in background mode by following the last entry with an ampersand (**&**) character.

Any pipeline may be placed in parenthesis to form a simple command which in turn may be a component of another pipeline. It is also possible to separate pipelines with "| |" or "&&" indicating, as in the C language, that the second pipeline is to be executed only if the first fails or succeeds, respectively.

Jobs

The shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the *jobs* command, and assigns them small integer numbers. When a job is started asynchronously with '&', the shell prints a line which looks like:

[1] 1234

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may type the currently defined *suspend* character (see *termio*(7)) which sends a stop signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the *bg* command, or run some other commands and then eventually bring the job back into the foreground with the foreground command *fg*. A *suspend* takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is a delayed *suspend* character which does not generate a stop signal until a program attempts to *read*(2) it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command "stty tostop" (see *stty*(1)). If you set this tty option, background jobs will stop when they try to produce output, as they do when they try to read input. Keyboard signals and line hangup signals from the terminal interface are not sent to background jobs on such systems. This means that background jobs are immune to the effects of logging out or typing the interrupt, quit, suspend, and delayed suspend characters (see *termio*(7)).

There are several ways to refer to jobs in the shell. The character '%' introduces a job name. If you wish to refer to job number 1, you can name it as '%1'. Just naming a job brings it to the foreground; thus '%1' is a synonym for 'fg %1', bringing job 1 back into the foreground. Similarly saying '%1 &' resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous, thus '%ex' would normally restart a suspended *ex*(1) job, if there were only one suspended job whose name began with the string 'ex'. It is also possible to say '%?string' which specifies a job whose text contains *string*, if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a '+' and the previous job with a '−'. The abbreviation '%+' refers to the current job and '%−' refers to the previous job. For close analogy with the syntax of the *history* mechanism (described below), '%%' is also a synonym for the current job.

Csh learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before printing a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, *csh* will notify you immediately of changes in status of background jobs. There is also a *csh* built-in command called *notify* which marks a single process so that its status changes will be immediately reported. By default, *notify* marks the current process. You can just say 'notify' after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that 'You have stopped jobs.' You may use the *jobs* command to see what they are. If you do this or

immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated (see *exit*(2)).

Built-In Commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell. The built-in commands are:

alias
alias name
alias name wordlist
The first form prints all aliases. The second form prints the alias for *name*. The final form assigns the specified *wordlist* as the alias of *name*. Command and file name substitution are performed on *wordlist*. *Name* cannot be **alias** or **unalias**.

bg [%*job* ...]
Puts the current (no argument) or specified jobs into the background, continuing them if they were stopped.

break Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a *switch*, resuming after the *endsw*.

case *label*:
A label in a *switch* statement as discussed below.

cd
cd *directory_name*
chdir
chdir *directory_name*
Change the shell's current working directory to *directory_name*. If no argument is given, *directory_name* defaults to your home directory.

If *directory_name* is not found as a subdirectory of the current working directory (and does not begin with "/", "./" or "../"), each component of the variable *cdpath* is checked to see if it has a subdirectory *directory_name*. Finally, if all else fails, *csh* treats *directory_name* as a shell variable. If its value begins with "/", this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:
Labels the default case in a *switch* statement. The default should come after all other *case* labels.

dirs Prints the directory stack; the top of the stack is at the left; the first directory in the stack is the current directory.

echo wordlist
echo −**n** wordlist
The specified words are written to the shell's standard output, separated by spaces, and terminated with a new-line unless the −**n** option is specified.

else
end

endif

endsw See the description of the *foreach*, *if*, *switch*, and *while* statements below.

eval arguments ...

(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions.

exec command

The specified command is executed in place of the current shell.

exit

exit (*expression*)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expression* (second form).

fg [*%job* ...]

Brings the current (no argument) or specified jobs into the foreground, continuing them if they were stopped.

foreach *name* (*wordlist*)

...

end The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The built-in command *continue* may be used to continue the loop prematurely and the built-in command *break* terminates it prematurely. When this command is read from the terminal, the loop is read once, prompting with '?' before any statements in the loop are executed. If you make a mistake while typing in a loop at the terminal you can then rub it out.

glob *wordlist*

Like *echo* but no '\' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to perform file name expansion on a list of words.

goto *word*

The specified *word* is file name and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

hashstat

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding *exec*'s). An *exec* is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/'.

history [**−h**] [**−r**] [*n*]

Displays the history event list. If *n* is given, only the *n* most recent events are printed. The **−r** option reverses the order of printout to be most recent first rather than oldest first. The **−h** option prints the history list without leading numbers for producing files suitalbe for the **source** command.

if (*expression*) *command*

If the specified expression evaluates true, the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the

rest of the **if** command. *Command* must be a simple command, not a pipeline, a command list, a parenthesized command list, or an aliased command. Input/output redirection occurs even if *expression* is false, when *command* is **not** executed (this is a bug).

if (*expression1*) **then**
...
else if (*expression2*) **then**
...
else
...
endif If the specified *expression1* is true, the commands to the first *else* are executed; otherwise if *expression2* is true, the commands to the second else are executed, etc. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** part is likewise optional. (The words **else** and **endif** must appear at the beginning of input lines; the **if** must appear alone on its input line or after an **else**.)

jobs [**−l**]
Lists the active jobs; the **−l** option lists process id's in addition to the normal information.

kill % *job*
kill − *sig* %*job* ...
kill *pid*
kill −*sig pid* ...
kill −l Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the "SIG" prefix − see *signal*(2)). The signal names are listed by **kill −l**. There is no default, so saying just **kill** does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), the job or process is sent a CONT (continue) signal as well.

login Terminates a login shell, replacing it with an instance of */bin/login*. This is one way to log off, included for compatibility with *sh*(1).

logout Terminates a login shell. Especially useful if *ignoreeof* is set.

newgrp
Changes the group identification of the caller; for details see *newgrp*(1). A new shell is executed by *newgrp* so that the current shell environment is lost.

nice
nice +*number*
nice *command*
nice +*number command*
The first form sets the *nice* (run command priority) for this shell to 4 (the default). The second form sets the priority to the given *number*. The final two forms run *command* at priority 4 and *number* respectively. The super-user may raise the priority by specifying negative niceness using **nice** −*number* .... *Command* is always executed in a sub-shell, and the restrictions place on commands in simple **if** statements apply.

nohup [*command*]
Without an argument, *nohup* can be used in shell scripts to cause hangups to be ignored for the remainder of the script. With an argument, causes the specified *command* to be run with hangups ignored. All processes executed in the background with **&** are effectively *nohup*'ed as described under Jobs in the COMMANDS section.

notify [*%job* ...]
Causes the shell to notify the user asynchronously when the status of the current (no argument) or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

onintr [−] [*label*]
Controls the action of the shell on interrupts. With no arguments, *onintr* restores the default action of the shell on interrupts, which is to terminate shell scripts or to return to the terminal command input level. If − is specified, causes all interrupts to be ignored. If a *label* is given, causes the shell to execute a **goto label** when an interrupt is received or a child process terminates because it was interrupted.

If the shell is running in the background and interrupts are being ignored, *onintr* has no effect; interrupts continue to be ignored by the shell and all invoked commands.

popd [+*n*]
Pops the directory stack, returning to the new top directory. With an argument, discards the *n*th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd [*name*] [+*n*]
With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (using *cd*) and pushes the old current working directory (as in *csw*) onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat *count command*
The specified *command* (which is subject to the same restrictions as the *command* in the one line **if** statement above) is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set *name*
set *name=word*
set *name*[*index*]=*word*
set *name*=(*wordlist*)
The first form of **set** shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index*'th component of *name* to *word*; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the *value* is command and file name expanded.

These arguments may be repeated to set multiple values in a single *set* command. Note, however, that variable expansion happens for all arguments before any setting occurs.

setenv *name value*
Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variables USER, TERM, and PATH are automatically

imported to and exported from the *csh* variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

shift [*variable*]
With no argument, the members of *argv* are shifted to the left, discarding *argv*[1]. An error occurs if *argv* is not set or has less than two strings assigned to it. With an argument, *shift* performs the same function on the specified *variable*.

source [**−h**] *name*
The shell reads commands from *name*. *Source* commands can be nested, but if nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Normally, input during *source* commands is not placed on the history list. The **−h** option can be used to place commands in the history list without being executed.

stop [*%job* ...]
Stops the current (no argument) or specified jobs executing in the background.

suspend
Causes *csh* to stop as if it had been sent a *suspend* signal. Since *csh* normally ignores *suspend* signals, this is the only way to suspend the shell. This command gives an error message if attempted from a login shell.

switch (*string*)
case *str1*:
...
breaksw
...
default:
...
breaksw
endsw Each *case* label (*str1*) is successively matched against the specified *string* which is first command and file name expanded. The form of the *case* labels is the Pattern Matching Notation with the exception that non-matching lists in bracket expressions are not supported (see *regexp*(5)). If none of the labels match before a **default** label is found, the execution begins after the **default** label. Each *case* label and the **default** label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise, control may fall through *case* labels and **default** labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time [*command*]
With no argument, a summary of time used by this shell and its children is printed. If an argument is given, the specified simple *command* is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask [*value*]
The current file creation mask is displayed (no argument) or set to the specified *value*. The mask is given in octal. Common values for the mask are 002, which gives all permissions to the owner and group, and read and execute permissions to all others, or 022, which gives all permissions to the owner, and only read and execute permission to the group and all others.

unalias *pattern*
All aliases whose names match the specified *pattern* are discarded. Thus, all aliases are removed by **unalias** *. No error occurs if *pattern* matches nothing.

unhash
Use of the internal hash table to speed location of executed programs is disabled.

unset *pattern*
All variables whose names match the specified *pattern* are removed. Thus, all variables are removed by **unset** *; this has noticeably distasteful side-effects. No error occurs if *pattern* matches nothing.

unsetenv *pattern*
Removes all variables whose names match the specified *pattern* from the environment. See also the *setenv* command above and *printenv*(1).

wait All background jobs are waited for. If the shell is interactive, an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (*expression*)
...
end While the specified *expression* evaluates non-zero, the commands between the *while* and the matching *end* are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) If the input is a terminal (i.e. not a script), prompting occurs the first time through the loop as for the *foreach* statement.

%job Brings the specified job into the foreground.

%job **&** Continues the specified job in the background.

specified *name* to the value of *expression*. If the expression contains "<", ">", "&" or "|", at least this part of the expression must be placed within parentheses. The third form assigns the value of *expression* to the *index*'th argument of *name*. Both *name* and its *index*'th component must already exist.

The operators "*=", "+=", etc., are available as in C. White space may optionally separate the *name* from the assignment operator. However, spaces are mandatory in separating components of *expression* which would otherwise be single words.

Special postfix "++" and "--" operators increment and decrement *name*, respectively (i.e. **Di**++).

Non-Built-In Command Execution

When a command to be executed is not a built-in command, the shell attempts to execute the command via *exec*(2). Each word in the variable *path* names a directory in which the shell attempts to find the command (if the command does not begin with "/"). If neither **−c** nor **−t** is given, the shell hashes the names in these directories into an internal table so that an *exec* is attempted only in those directories where the command might possibly reside. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if **−c** or **−t** was given, or if any directory component of *path* does not begin with a '/', the shell concatenates the directory name and the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus

(cd ; pwd)

prints the *home* directory but leaves you where you were.

cd ; pwd

does the same thing, but leaves you in the *home* directory.

Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary file, it is assumed to be a script file, which is a file of data for an interpreter that is executed as a separate process.

Csh first attempts to load and execute the script file (see *exec*(2)). If the first two characters of the script file are "#!", *exec*(2) expects an interpreter path name to follow and attempts to execute the specified interpreter as a separate process to read the entire script file.

If no "#! interpreter" is named, and there is an *alias* for the shell, the words of the *alias* are inserted at the beginning of the argument list to form the shell command. The first word of the *alias* should be the full path name of the command to be used. Note that this is a special, late-occurring case of *alias* substitution, which inserts words into the argument list without modification.

If no "#! interpreter" is named and there is no shell *alias*, but the first character of the file is #, the interpreter named by the **$shell** variable is executed (note that this normally would be **/bin/csh**, unless the user has reset **$shell**). If **$shell** is not set, **/bin/csh** is executed.

If no "!# interpreter" is named, and there is no shell alias and the first character of the file is not #, **/bin/sh** is executed to interpret the script file.

History Substitutions

History substitutions enable you to use words from previous commands as portions of new commands, repeat commands, repeat arguments of a previous command in the current command, and fix spelling mistakes in the previous command.

History substitutions begin with an exclamation point (!). Substitutions may begin anywhere in the input stream, but may **not** be nested. The exclamation point can be preceded by a backslash to prevent its special meaning. For convenience, an exclamation point is passed to the parser unchanged when it is followed by a blank, tab, newline, equal sign or left parenthesis. Any input line which contains history substitution is echoed on the terminal before it is executed for verification.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The number of previous commands saved is controlled by the *history* variable. The previous command is always saved, regardless of its value. Commands are numbered sequentially from 1.

You can refer to previous events by event number (such as **!10** for event 10), relative event location (such as **!−2** for the second previous event), full or partial command name (such as **!d** for the last event using a command with initial character d), and string expression (such as **!?mic?** referring to an event containing the characters **mic**).

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, **!!** is a re-do; it refers to the previous command.

To select words from a command you can follow the event specification by a colon (:) and a designator for the desired words. The words of an input line are numbered from zero. The basic word designators are:

0 selects the first word (i.e. the command name itself).

n selects the *n*th word.

^ selects the first argument. (This is equivalent to '1'.)

$ selects the last word.

a-b selects the range of words from *a* to *b*. Special cases are **−y**, which is an abbreviation for "word 0 through word *y*", and **x−**, which stands for "word *x* up to, but not including, word **$**".

* indicates the range from the second word to the last word.

% used with a search sequence to substitute the immediately preceding matching word.

The colon separating the command specification from the word designator can be omitted if the argument selector begins with a ^, $, *, −, or %.

After each word designator, you can place a sequence of modifiers, each preceded by a colon. The following modifiers are defined:

h Use only the first component of a path name by removing all following components.

r Use the root file name by removing any trailing suffix (.xxx).

e Use the file name's trailing suffix (.xxx) by removing the root name.

s/*l*/*r* substitute the value of *r* for the value *l* in the indicated command.

t Use only the final file name of a path name by removing all leading path name components.

& Repeat the previous substitution.

p Print the new command but do not execute it.

q Quote the substituted words, preventing further substitutions.

x Like **q**, but break into words at blanks, tabs and newlines.

g Use a **g**lobal command as a prefix to another modifier to cause the specified change to be made globally. All words in the command are changed, one change per word, and each string enclosed in single quotes (') or double quotes (") is considered one word.

Unless preceded by a **g**, the modification is applied only to the first modifiable word. You get an error if a substitution is attempted and cannot be completed (i.e. if you have a history buffer of 10 commands and ask for a substitution of **!11**).

The left hand side of substitutions are not regular expressions in the sense of the HP-UX editors, but rather strings. Any character may be used as the delimiter in place of a slash (/); a backslash quotes the delimiter into the *l* and *r* strings. The character & in the right hand side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in **!?***s*?. The trailing delimiter in the substitution may be omitted if a newline follows immediately, as may the trailing **?** in a contextual scan.

A history reference may be given without an event specification (e.g. **!$**). In this case the reference is to the previous command unless a previous history reference occurred on the same line, in which case this form repeats the previous reference. Thus

!?foo?^ !$

gives the first and last arguments from the command matching "?foo?".

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a caret (^). This is equivalent to "!:s^", providing a convenient shorthand for substitutions on the text of the previous line. Thus "^lb^lib" fixes the spelling of "lib" in the previous command.

Finally, a history substitution may be surrounded with curly braces { } if necessary to insulate it from the characters which follow. Thus, after

ls −ld ~paul

we might execute **!{l}a** to do

ls −ld ~paula

while **!la** would look for a command starting with "la".

Quoting with Single and Double Quotes

The quotation of strings by single quotes (') and double quotes (") can be used to prevent all or some of the remaining substitutions. Strings enclosed in single quotes are protected from any further interpretation. Strings enclosed in double quotes are still variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitution* below) does a double-quoted string yield parts of more than one word; single-quoted strings never do.

Alias Substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, the argument list is left unchanged.

Thus, if the alias for **ls** is **ls −l**, the command **ls /usr** maps to **ls −l /usr**, leaving the argument list undisturbed. Similarly, if the alias for **lookup** was **grep !^ /etc/passwd**, **lookup bill** maps to **grep bill /etc/passwd**.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the re-formed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can execute

alias print 'pr \!* | lp'

to make a command which uses *pr*(1) to print its arguments on the line printer.

Expressions

A number of the built-in commands take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the **exit**, **if**, and **while** commands. The following operators are available (shown in order of increasing precedence):

|| && | ^ & == != =~ !~ <= >= < > << >> + − * / % ! ~ ( )

The following list shows the grouping of these operators. The precedence decreases from top to bottom in the list:

```
* / %
+ −
<< >>
<= >= < >
== != =~ !~
```

The ==, !=, =~, and !~ operators compare their arguments as strings; all others operate on numbers. The operators =~ and !~ are like != and ==, except that the right hand side is a

pattern (containing *, ?'s, and instances of [...]) against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with **0** are considered octal numbers. Null or missing arguments are considered **0**. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word. These components should be surrounded by spaces except when adjacent to components of expressions which are syntactically significant to the parser – &, |, <, >, (, and).

Also available in expressions as primitive operands are command executions enclosed in curly braces { } and file enquiries of the form "*–l filename*", where *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified *filename* is command and file name expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, all inquiries return false (0). Command executions succeed, returning true, if the command exits with status 0; otherwise they fail, returning false. If more detailed status information is required, the command should be executed outside of an expression and the variable *status* examined.

Control of the Flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip parts of its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach*, *switch*, and *while* statements, as well as the *if–then–else* form of the *if* statement, require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's succeed on non-seekable inputs.)

Signal Handling

The shell normally ignores *quit* signals. Jobs running in background mode are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file **.logout**.

Command Line Parsing

Csh splits input lines into words at blanks and tabs. The following exceptions (parser metacharacters) are considered separate words:

&	ampersand;
|	vertical bar;
;	semicolon;
<	less-than sign;

> greater-than sign;
(left parenthesis;
) right parenthesis;
&& double ampersand;
|| double vertical bar;
<< double less-than sign;
>> double greater-than sign;

The backslash (\) removes the special meaning of these parser metacharacters. A parser metacharacter preceded by a backslash is interpreted as its ASCII value. A newline character (ASCII 10) preceded by a backslash is equivalent to a blank.

Strings enclosed in single or double quotes form parts of a word. Metacharacters in these strings, including blanks and tabs, do not form separate words. Within pairs of backslashs or quotes, a newline preceded by a backslash gives a true newline character.

When the shell's input is not a terminal, the pound sign (#) introduces a comment terminated by a newline.

CSH VARIABLES

Csh maintains a set of variables. Each variable has a value equal to zero or more strings (words). Variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter. The value of a variable may be displayed and changed by using the *set* and *unset* commands. Some of the variables are Boolean, that is, the shell does not care what their value is, only whether they are set or not.

Some operations treat variables numerically. The at sign (@) command permits numeric calculations to be performed and the result assigned to a variable. The null string is considered to be zero, and any subsequent words of multi-word values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable expansion is performed keyed by the dollar sign ($) character. Variable expansion can be prevented by preceding the dollar sign with a backslash character (\) except within double quotes (") where substitution **always** occurs. Variables are never expanded if enclosed in single quotes. Strings quoted by single quotes are interpreted later (see *Command Substitution*) so variable substitution does not occur there until later, if at all. A dollar sign is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together.

Unless enclosed in double quotes or given the **:q** modifier, the results of variable substitution may eventually be command and file name substituted. Within double quotes, a variable whose value consists of multiple words expands to a portion of a single word, with the words of the variable's value separated by blanks. When the **:q** modifier is applied to a substitution, the variable expands to multiple words with each word separated by a blank and quoted to prevent later command or file name substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$variable_name
${variable_name}
When interpreted, this sequence is replaced by the words of the value of the variable *variable_name*, each separated by a blank. Braces insulate *variable_name* from following characters which would otherwise be interpreted to be part of the variable name itself.

If *variable_name* is not a *csh* variable, but is set in the environment, that value is used. Non-*csh* variables cannot be modified as shown below.

$variable_name[selector]
${variable_name[selector]}
: This modification allows you to select only some of the words from the value of *variable_name*. The selector is subjected to variable substitution and may consist of a single number or two numbers separated by a dash. The first word of a variable's value is numbered **1**. If the first number of a range is omitted it defaults to **1**. If the last member of a range is omitted it defaults to the total number of words in the variable ($#variable_name). An asterisk metacharacter used as a selector selects all words.

$#variable_name
${#variable_name}
: This form gives the number of words in the variable. This is useful for forms using a [*selector*] option.

$0 : This form substitutes the name of the file from which command input is being read. An error occurs if the file name is not known.

$number
${number}
: This form is equivalent to an indexed selection from the variable *argv* ($argv[number]).

$* : This is equivalent to selecting all of *argv* ($argv[*]).

The modifiers **:h**, **:t**, **:r**, **:q** and **:x** may be applied to the substitutions above, as may **:gh**, **:gt** and **:gr**. If curly braces { } appear in the command form, the modifiers must appear within the braces. *The current implementation allows only one : modifier on each $ expansion.*

The following substitutions may not be modified with **:** modifiers.

$?variable_name
${?variable_name}
: Substitutes the string **1** if *variable_name* is set, **0** if it is not.

$?0 : Substitutes **1** if the current input file name is known, **0** if it is not.

$$: Substitutes the (decimal) process number of the (parent) shell.

$< : Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Pre-Defined and Environment Variables

The following variables have special meaning to the shell. Of these *autologout*, *argv*, *cwd*, *home*, *path*, *prompt*, *shell*, and *status* are always set by the shell. Except for *cwd* and *status*, this setting occurs only at initialization (initial execution of *csh*). These variables are not modified unless modified explicitly by the user.

Csh copies the HP-UX environment variable USER into the shell variable *user*, the environment variable TERM into *term*, the environment variable HOME into *home*, and PATH into *path*. *Csh* copies these values back into the environment whenever the *csh* variables are reset.

argv : This variable is set to the arguments of the *csh* command statement. It is from this variable that positional parameters are substituted, i.e. **$1** is replaced by **$argv[1]**, etc.

cdpath This variable gives a list of alternate directories searched to find subdirectories in *chdir* commands.

cwd This variable contains the absolute path name of your current working directory. Whenever you change directories (using *cd*), this variable is updated.

echo This variable is set by the **−x** command line option. If set, all built-in commands and their arguments are echoed to your standard output device just before being executed. Built-in commands are echoed before command and file name substitution, since these substitutions are then done selectively. For non-built-in commands, all expansions occur before echoing.

hidden If set, the shell will include unmatched CDF's in expansions (see *cdf*(4)). Also, unmatched CDF's will be displayed with a trailing + when using the **control-D** command to display matching file names. (See the CSH UTILITIES section.) By default, **hidden** is not set.

history This variable is used to create your command history buffer and to set its size. If this variable is not set, you have no command history and can do no history substitutions. Very large values of **history** may run your shell out of memory. Values of 10 or 20 are normal. All commands, executable or not, are saved in your command history buffer.

home This variable contains the absolute path name to your home directory. The variable **home** is initialized from the HP-UX environment. The file name expansion of tilde (~) refers to this variable.

ignoreeof If set, *csh* ignores end-of-file characters from input devices that are terminals. *Csh* will exit normally when it encounters the end-of-file condition, which is control-D typed as the first character on a command line. Setting *ignoreeof* prevents your current shell from being killed by an accidental control-D.

mail This variable contains a list of the files where *csh* checks for your mail. *Csh* periodically (default is 10 minutes) checks this variable after a command completion which results in a prompt. If the variable contains a file name that has been modified since the last check (resulting from mail being put in the file), *csh* prints **You have new mail**.

If the first word of the value of *mail* is numeric, that number specifies a different mail checking interval in seconds.

If multiple mail files are specified, the shell says **New mail in** *file_name*, where *file_name* is the file containing the mail.

noclobber This variable places restrictions on output redirection to insure that files are not accidentally destroyed, and that commands using append redirection (>>) refer to existing files.

noglob If set, file name expansion is inhibited. This is most useful in shell scripts which are not dealing with file names, or after a list of file names has been obtained and further expansions are not desirable.

nonomatch If set, it is no longer an error for a file name expansion to not match any existing files. If there is no match, the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. ´**echo [**´ still gives an error.

notify If set, *csh* notifies you immediately (through your standard output device) of background job completions. The default is **unset** (indicate job completions just before printing a prompt).

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies your current working directory. If there is no *path* variable, only full path names can be executed. When *path* is not set and when users do not specify full path names, *csh* searches for the command through the directories . (your current directory), */bin*, and */usr/bin*. A *csh* which is given neither the **−c** nor the **−t** option normally hashes the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it is necessary to execute *rehash* for *csh* to access these new commands.

prompt This variable lets you select your own prompt character string. The prompt is printed before each command is read from an interactive terminal input. If a **!** appears in the string it is replaced by the current command history buffer event number unless a preceding \ is given. The default prompt is the percent sign (%) for users and the pound sign (#) for the super-user.

shell This variable contains the name of the file in which the *csh* program resides. This variable is used in forking shells to interpret files which have their execute bits set, but which are not executable by the system. (See the description of **Non-built-In Command Execution**).

status This variable contains the status value returned by the last command. If the command terminated abnormally, 0200 is added to the status variable's value. Built-in commands which terminated abnormally return exit status **1**, and all other built-in commands set status to **0**.

time This variable contains a numeric value which controls the automatic timing of commands. If set, *csh* prints, for any command which takes more than the specified number of cpu seconds, a line of information to your standard output device giving user, system, and real execution times plus a utilization percentage. The utilization percentage is the ratio of user plus system times to real time. This message is printed after the command finishes execution.

verbose This variable is set by the **−v** command line option. If set, the words of each command are printed on the standard output device after history substitutions have been made.

Command and File name Substitution

The remaining substitutions, command and file name substitution, are applied selectively to the arguments of built-in commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command Substitution

Command substitution is indicated by a command enclosed in grave accents (` ...`). The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within double quotes, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

File name Substitution

Each command *word* is processed as a pattern for file name substitution, also known as *globbing*, and replaced with a sorted list of file names which match the pattern. The form of the patterns is the Pattern Matching Notation defined by *regexp*(5) with the following exceptions:

Non-matching lists in bracket expressions are not supported.

In a list of words specifying file name substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match.

The metanotation **a{b,c,d}e** is a shorthand for "abe ace ade". Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus

~source/s1/{oldls,ls}.c

expands to

/usr/source/s1/oldls.c /usr/source/s1/ls.c

whether or not these files exist, without any chance of error if the home directory for **source** is */usr/source*. Similarly,

../{memo,*box}

might expand to

../memo ../box ../mbox

(Note that "memo" was not sorted with the results of matching ***box**.) As a special case, {, }, and { } are passed undisturbed.

Input/Output

The standard input and standard output of a command may be redirected with the following syntax:

< *name*

Open file *name* (which is first variable, command and file name expanded) as the standard input.

<< *word*

Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, file name or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, ", ', or ` appears in *word*, variable and command substitution is performed on the intervening lines, allowing \ to quote $, \ and `. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> *name*
>**!** *name*
>**&** *name*
>**&!** *name*

The file *name* is used as standard output. If the file does not exist, it is created; if the file exists, it is truncated, and its previous contents are lost.

If the variable *noclobber* is set, the file must not exist or be a character special file (e.g. a terminal or **/dev/null**) or an error results. This helps prevent

accidental destruction of files. In this case the exclamation point (!) forms can be used to suppress this check.
The forms involving the ampersand character (&) route the standard error into the specified file as well as the standard output. *Name* is expanded in the same way as < input file names are.

>> *name*
>>& *name*
>>! *name*
>>&! *name*
Uses file *name* as standard output like >, but appends output to the end of the file. If the variable *noclobber* is set, it is an error for the file not to exist unless one of the ! forms is given. Otherwise, it is similar to >.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands executed from a shell script have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form "|&" rather than just "|".

CSH UTILITIES

File Name Completion

In typing file names as arguments to commands, it is no longer necessary to type a complete name, only a unique abbreviation is necessary. When you want the system to try to match your abbreviation, press your ESCAPE key. The system then completes the file name for you, echoing the full name on your terminal. If the abbreviation doesn't match an available file name, the terminal's bell is sounded. The file name may be partially completed if the prefix matches several longer file names. In this case, the name is extended up to the ambiguous deviation, and the bell is sounded.

File name completion works equally well when other directories are addressed. In addition, the tilde (~) convention for home directories is understood in this context.

Viewing a File or Directory List

At any point in typing a command, you may request "what files are available" or "what files match my current specification". Thus, when you have typed:

```
% cd ~speech/data/bench/fritz/
```

you may wish to know what files or subdirectories exist (in **~speech/data/bench/fritz**), without aborting the command you are typing. Typing **control-D** at this point lists the files available. The files are listed in multicolumn format, sorted column-wise. Directories and executable files are indicated with a trailing / and *, respectively. Once printed, the command is re-echoed for you to complete. Additionally, you may want to know which files match a prefix, the current file specification so far. If you had typed:

```
% cd ~speech/data/bench/fr
```

followed by a **control-D**, all files and subdirectories whose prefix was "fr" in the directory **~speech/data/bench** would be printed. Notice that the example before was simply a degenerate case of this with a null trailing file name. (The null string is a prefix of all strings.) Notice also that a trailing slash is required to pass to a new sub-directory for both file name completion and listing. Note that the degenerate case

```
% ~^D
```

prints a full list of login names on the current system.

Command Name Recognition

Command name recognition and completion works in the same manner as file name recognition and completion above. The current value of the environment variable PATH is used in searching for the command. For example

% newa [Escape]

might expand to

% newaliases

Also,

% new [Control]-[D]

lists all commands (along PATH) that begin with "new". As an option, if the shell variable *listpathnum* is set, a number indicating the index in PATH is printed next to each command on a [Control]-[D] listing.

Autologout

A new shell variable has been added called *autologout*. If the terminal remains idle (no character input) at the shell's top level for a number of minutes greater than the value assigned to *autologout*, you are automatically logged off. The *autologout* feature is temporarily disabled while a command is executing. The initial value of *autologout* is 60. If unset or set to 0, *autologout* is entirely disabled.

Command Line Control

A ^R will re-print the current command line; ^W will erase the last word entered on the current command line.

Sanity

The shell now restores your terminal to a sane mode if it appears to return from some command in raw, cbreak, or noecho mode.

Saving Your History Buffer

Csh has the facility to save your history list between login sessions. If the shell variable *savehist* is set to a number, that number of command events from your history list are saved. For example, placing the line

set history=10 savehist=10

in your **.cshrc** file maintains a history buffer of length 10 and saves the entire list when you logout. When you log back in, the entire buffer is restored. The commands are saved in the file **.history** in your login directory.

WARNINGS

Csh has certain limitations. Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves file name expansion is limited to one-sixth the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list.

To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

When a command is restarted from a stop, *csh* prints the directory it started in if it is different from the current directory; this can be misleading (i.e. wrong) as the job may have changed directories internally.

Shell built-in functions are not stoppable/restartable. Command sequences of the form "a ; b ; c" are also not handled gracefully when stopping is attempted. If you interrupt b, the shell then immediately executes c. This is especially noticeable if this expansion results from an

alias. It suffices to place the sequence of commands in ()'s to force it into a subshell, i.e. (a ; b ; c).

Because of the signal handling required by csh, interrupts are disabled just before a command is executed and restored as the command begins execution. There may be a few seconds delay between when a command is given and when interrupts are recognized.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by ?, are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

Your terminal type is only examined the first time you attempt recognition.

To list all commands on the system along PATH, enter [SPACE]-[CNTL]-[D].

The csh metasequence !~ does not work.

The **−x** *filename* always tests true for super-user, even if *filename* is not executable.

In an international environment, character ordering is determined by the setting of LC_COLLATE, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particulary when using range expressions in file name generation patterns. For example, the command,

rm [a-z]*

might be expected to match all file names beginning with a lower-case alphabetic character. However, if dictionary ordering is specified by LC_COLLATE, it would also match file names beginning with an upper-case character (as well as those beginning with accented letters). Conversely, it would fail to match letters collated after 'z' in languages such as Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form:

rm [[:lower:]]*

This uses LC_CTYPE to determine character classes and will work predictably for all supported languages and codesets. For shell scripts produced on non-internationalized systems (or without consideration for the above dangers), it is recommended that they be executed in a non-NLS environment. This requires that LANG, LC_COLLATE, etc., be set to "C" or not set at all.

AUTHOR

Csh was developed by the University of California, Berkeley and HP.

FILES

~/.cshrc	a csh script sourced (executed) at the beginning of execution by each shell.
~/.login	a csh script sourced (executed) by login shell, after .cshrc at login.
~/.logout	a csh script sourced (executed) by login shell, at logout.
/etc/passwd	source of home directories for ~name.

/bin/sh	standard shell, for shell scripts not starting with a #.
/etc/csh.login	a csh script sourced (executed) before ~/.cshrc and ~/.login when starting a csh login (analogous to **/etc/profile** in the Bourne shell).
/tmp/sh*	temporary file for <<.

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), umask(2), wait(2), tty(7), a.out(4), environ(5), lang(5), regexp(5).

The C Shell (csh), in *HP-UX Concepts and Tutorials: Shells and Miscellaneous Tools.*

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name substitution.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as letters, and the characters matched by character class expressions in pattern matching notation.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *csh* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

csplit – context split

SYNOPSIS

csplit [**–s**] [**–k**] [**–f** *prefix*] *file arg1* [*... argn*]

DESCRIPTION

Csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1... argn*. By default the sections are placed in xx00 ... xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: >From the start of *file* up to (but not including) the line referenced by *arg1*.
01: >From the line referenced by *arg1* up to the line referenced by *arg2*.
.
.
.
n: >From the line referenced by *argn* to the end of *file*.

If the *file* argument is a – then standard input is used.

Csplit supports the Basic Regular Expression syntax (see *regexp*(5)).

Options

The options to *csplit* are:

–s *Csplit* normally prints the character counts for each file created. If the **–s** option is present, *csplit* suppresses the printing of all character counts.

–k *Csplit* normally removes created files if an error occurs. If the **–k** option is present, *csplit* leaves previously created files intact.

–f *prefix* If the **–f** option is used, the created files are named *prefix***00** ... *prefixn*. The default is **xx00** ... **xx***n*.

The arguments (*arg1 ... argn*) to *csplit* can be a combination of the following:

/rexp/ A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or – some number of lines (e.g., **/Page/–5**).

%rexp%
This argument is the same as */rexp/*, except that no file is created for the section.

lnno A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *Csplit* does not affect the original file; it is the users responsibility to remove it.

DIAGNOSTICS

Self explanatory except for:

arg – out of range

which means that the given argument did not reference a line between the current position and the end of the file. This warning will also occur if the file is exhausted before the repeat count is.

EXAMPLES

```
csplit −f cobol file ´/procedure division/´ /par5./ /par16./
```

This example creates four files, **cobol00 ... cobol03**. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0−3] > file
```

Note that this example overwrites the original file.

```
csplit −k file 100 ´{99}´
```

This example would split the file at every 100 lines, up to 10,000 lines. The **−k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit −k prog.c ´%main(%´ ´/^}/+1´ ´{20}´
```

Assuming that **prog.c** follows the normal **C** coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate **C** routine (up to 21) in **prog.c**.

SEE ALSO

sh(1), environ(5), lang(5), regexp(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the characters matched by character class expressions in regular expressions.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *csplit* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

csplit: SVID2, XPG2, XPG3

NAME

ct – spawn getty to a remote terminal (call terminal)

SYNOPSIS

ct [**−w***n*] [**−x***n*] [**−h**] [**−v**] [**−s***speed*] *telno* ...

DESCRIPTION

Ct dials *telno*, the telephone number of a modem that is attached to a terminal, and spawns a *getty*(1M) process to that terminal.

Ct tries each line listed in the file **/usr/lib/uucp/Devices** until it finds an available line with appropriate attributes or runs out of entries. If no lines are free, *ct* asks whether it should wait for a line, and if so, how many minutes it should wait before giving up. *Ct* searches again for an available line at one-minute intervals until the specified limit is exceeded. Note that normally, *ct* disconnects the current tty line, so that the line can answer the incoming call. This is because *ct* assumes that the current tty line is connected to the terminal to spawn the *getty* process.

The *telno* argument specifies the telephone number, which can be composed of characters 0 thru 9, −, =, *, and #. Use equal signs to signify secondary dial tones and minus signs for delays at appropriate places. The maximum length of *telno* is 31 characters. If more than one telephone number is specified, *ct* tries each in succession until one answers; this is useful for specifying alternate dialing paths.

Options

−w*n* — Instruct *ct* to wait for a line a maximum of *n* number of minutes, if lines are busy. If this option is specified, *ct* does not query the user about whether to wait for a line.

−*x*n — Produce detailed output from the program execution on the standard error output. This option is used for debugging. The debugging level *n* is a single digit; the most useful value is **−x9**.

−h — Prevent *ct* from disconnecting ("hanging up") the current tty line. This option is necessary if the user is using a different tty line than the one used by *ct* to spawn the *getty*.

−v — Verbose mode. The **−v** option is used with the **−h** option and causes *ct* to send a running narrative to the standard error output stream.

−s*speed* — Set the data rate where *speed* is expressed in baud. The default rate is **1200**.

After the user on the destination terminal logs out, *ct* prompts, **Reconnect?** If the response begins with the letter **n** the line is dropped. Otherwise, *getty* is restarted and the **login:** prompt is printed.

Of course, the destination terminal must be attached to a modem that can automatically answer incoming calls.

DEPENDENCIES

HP Clustered Environment

Ct is not supported on cluseter client nodes in an HP Cluster.

FILES

/usr/adm/ctlog
/usr/lib/uucp/Devices

SEE ALSO

cu(1), getty(1M), login(1), uucp(1).

UUCP, tutorial in *HP-UX Concepts and Tutorials.*

NAME

ctags – create a tags file

SYNOPSIS

ctags [**–xvFBatwu**] names...

DESCRIPTION

Ctags makes a tags file for *ex*(1) (or *vi*(1)) from the specified C, Pascal and Fortran sources. A *tags* file gives the locations of specified objects (for C, functions, macros with argments, and typedefs; Pascal, procedures, programs and functions; FORTRAN, subroutines, progams and functions) in a group of files. Each line of the *tags* file contains the object name, the file in which it is defined, and an address specification for the object definition. Output is sorted in ascending collation order (see Environment Variables below). All objects except C *typedefs* are searched with a pattern, *typedefs* with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these objects' definitions.

–x causes *ctags* to print a simple function index. This is done by assembling a list of function names, file names on which each function is defined, the line numbers where each function name occurs, and the text of each line. The list is then printed on the standard output. No *tags* file is created or changed.

–v produces a page index on the standard output. This listing contains the function name, file name, and page number within that file (assuming 56 line pages to match *pr*(1)).

Files whose name ends in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:

–F use forward searching patterns (/.../) (default).

–B use backward searching patterns (?...?).

–a add the information from the files to the *tags* file. Unlike re-building the *tags* file from the original files, this can cause the same symbol to be entered twice in the *tags* file. This option should be used with caution and then only in very special circumstances.

–t create tags for typedefs.

–w suppressing warning diagnostics.

–u causing the specified files to be *updated* in *tags*, that is, all references to those files are deleted, and the new values are added to the file as in **-a** above. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

RETURNS

Too many entries to sort.

An attempt to get additional heap space failed; the sort could not be performed.

Duplicate entry in file *file*, line *line*: *name*.
Second entry ignored.

The same name was detected twice in the same file. A *tags* entry was made only for the first name found.

Duplicate entry in files *file1* and *file2*: *name* (Warning only).

The same name was detected in two different files. A *tags* entry was made only for the first name found.

WARNINGS

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done in a very simple way. No attempt is made to deal with block structure; if there are two Pascal procedures in different blocks with the same name a warning message will be generated.

The method of deciding whether to look for C or Pascal and FORTRAN functions is an approximation and can be fooled by unusual programs.

It does not know about **#ifdefs** and Pascal types.

It relies on the input being well formed to detect *typedefs*.

Use of -tx shows only the last line of typedefs.

Ex(1) is naive about *tags* files with several identical tags; it simply chooses the first entry its (non-linear) search finds with that tag. Such files can be created with either the **-u** or **-a** options or by editing a *tags* file.

If more than one (function) definition appears on a single line, only the first definition will be indexed.

AUTHOR

Ctags was developed by the University of California, Berkeley.

FILES

tags	output tags file
OTAGS	temporary used by -u

SEE ALSO

ex(1), vi(1).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

LC_CTYPE determines the interpretation of the single- and/or multi-byte characters within comments and string literals.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ctags* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

NAME

cu − call another (UNIX) system; terminal emulator

SYNOPSIS

cu [−s*speed*] [−l*line*] [−**h**] [−**q**] [−**t**] [−d*level*] [−**e**|−**o**] [−**m**] [−**n**] { *telno* | *systemname* | **dir** }

DESCRIPTION

Cu calls up another system, which is usually a UNIX operating system, but can be a terminal or a non-UNIX operating system. *Cu* manages an interactive conversation with possible transfers of ASCII files.

Options

−s*speed* Specify the transmission speed (110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200). The default value is 300.

−l*line* Specify a device name to use as the communication line. This can be used to override searching for the first available line having the right speed. When the **-l** option is used without the **-s** option, the speed of a line is taken from the file **/usr/lib/uucp/Devices**. When the **-l** and **-s** options are used simultaneously, *cu* searches **/usr/lib/uucp/Devices** to determine whether the requested speed for the requested line is available. If so, the connection is made at the requested speed; otherwise an error message is printed and the call is not made. The specified device is generally a directly connected asynchronous line (for example, **/dev/ttyab**). In this case, a phone number is not required but the string **dir** can be used to specify that a dialer is not required. If the specified device is associated with an auto-dialer, a phone number must be provided.

−**h** Emulate local echo, supporting calls to other computer systems that expect terminals to be set to half-duplex mode.

−**q** Invoke the use of ENQ/ACK handshake. (Remote sends ENQ, *cu* sends ACK.)

−**t** Used when dialing an ASCII terminal that has been set to auto-answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

−d*level* Cause diagnostic traces to be printed. Level is a number from 0-9. The higher the level, the more detail the diagnostic messages will contain.

−**e**(−**o**) Designate that even (odd) parity should be generated for data sent to the remote.

−**m** Designate a direct line that has modem controls. The modem controls are ignored by *cu*.

−**n** Cause the phone number that *cu* dials to be requested interactively from the user rather than taking it from the command line.

telno When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs for delays, at appropriate places.

systemname A *uucp* system name can be used instead of a phone number (see *uucp*(1)); in this case, *cu* obtains an appropriate direct line or phone number from **/usr/lib/uucp/Systems** (the appropriate baud rate is also read with phone numbers). *Cu* dials each phone number or direct line for *systemname* in the **Systems** file until a connection is made or all the entries are tried.

dir Using **dir** ensures that *cu* uses the line specified by the **-l** option.

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the

receive process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote to ensure that the buffer is not overrun. "Prompt handshaking" can be used to control transfer of ASCII files to systems that have no "type-ahead" capability but require data to be sent only after a prompt is given. This is described in detail below. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following:

~. and ~..
: Terminate the conversation. On a hardwired line (only), ~. sends several EOF characters to log out the session; ~.. suppresses the EOF sequence. In general the remote hardwired machine is unaware of the disconnect if ~.. is used. The ~. and ~.. do not differ for dialup connections.

~!
: Escape to an interactive shell on the local system.

~!*cmd...*
: Run *cmd* on the local system (via **sh -c**).

~&
: Similar to ~! but kill the receive process, restarting it upon return from the shell. This is useful for invoking sub-processes that read from the communication line, where the receive process would otherwise compete for input.

~&*cmd...*
: Run *cmd* on the local system (via **sh -c**) and kill the receive process, restarting it later.

~**$***cmd...*
: Run *cmd* locally and send its output to the remote system.

~**%cd**
: Change the directory on the local system. NOTE: ~**!cd** causes the command to be run by a sub-shell, which has no lasting effect on the current directory.

~**%take** *from* [*to*]
: Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.

~**%put** *from* [*to*]
: Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.

~~...
: Send the line ~... to the remote system. If you use *cu* on the remote system to access a third remote system, send ~~. to cause the second remote *cu* to exit.

~**%break**
: Transmit a **BREAK** to the remote system.

~**%nostop**
: Toggle between DC3/DC1 input control protocol and no input control. This is useful if the remote system does not respond properly to the DC3 and DC1 characters.

~%<*file*
: Send the contents of the local file to the remote system using **prompt handshaking.** The specified file is read a line at a time, and each line is sent to the remote system when the **prompt sequence** is received. If no prompt is received by the time the **prompt timeout** occurs, the line is sent anyway. If the timeout is set to 0 seconds, or if the first character in the prompt sequence is a null character (^@), the handshake always appears to be satisfied immediately, regardless of whether or not the remote system generates a prompt. This capability is intended mainly to facilitate transfer of ASCII files from HP-UX to an HP3000 system running MPE. This is usually accomplished by running the MPE utility *FCOPY*, and giving the command "from=;to=destfile;new" and then running the *cu* input diversion to send the file to *FCOPY*, which saves it in "destfile." This facility might be useful with other systems also, such as an HP1000 running *RTE*.

~**%setpt** *n* Specify the number of seconds to wait for a prompt before giving up. The default is 2 seconds. Specifying a timeout of 0 seconds disables handshaking; that is, handshake appears to complete immediately.

~**%setps** *xy* Set the handshake prompt to the characters *xy*. The default is DC1. The prompt can be any one or two characters. A control character X, that is, Control-X, is specified with a caret (ASCII 94) preceding the character, that is, ^X. A null character can be specified with ^@. (A null first character in the prompt implies a "null" prompt, which always appears to be satisfied.) A caret is specified by ^^.

~**%>**[**>**]*file* Divert output from the remote system to the specified file until another ~**%>** command is given. When an output diversion is active, typing ~**%>** terminates it, and ~**%>** *anotherfile* terminates it and begins a new one. The output diversion remains active through a ~**&** subshell, but unpredictable results can occur if input/output diversions are intermixed with ~**%take** or ~**%put.** The ~**%>>** command appends to the named file. Note that these commands, which are interpreted by the *transmit* process, are unrelated to the ~> commands described below, which are interpreted by the receive process.

~*susp* Suspend the *cu* session. *Susp* is the suspend character set in the terminal when *cu* was invoked (usually ^Z) (see *stty*(1)).

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

~>[>]: *file*
zero or more lines to be written to *file*
~>

Data from the remote is diverted (or appended, if >> is used) to *file*. The trailing ~> terminates the diversion.

The use of ~**%put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of ~**%take** requires the existence of *echo*(1) and *cat*(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are being copied without expansion. When connecting to a machine that uses the eighth bit as a parity bit, **stty istrip** mode should be set on the local system.

When *cu* is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed if ~~ is used. For example, *uname* can be executed on Z, X, and Y as follows:

uname
Z
~!*uname*
X
~~!*uname*
Y

In general, ~ causes the command to be executed on the original machine; ~~ causes the command to be executed on the next machine in the chain.

DIAGNOSTICS

Exit code is zero for normal exit, non-zero (various values) otherwise.

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud:

cu -s1200 9=2015551212

If the speed is not specified, 300 is the default value.

To login to a system connected by a direct line:

cu -l/dev/ttyXX dir

To dial a system with the specific line and a specific speed:

cu -s1200 -l/dev/ttyXX dir

To dial a system using a specific line:

cu -l/dev/culXX 2015551212

To use a system name:

cu YYYZZZ

To connect directly to a modem:

cu -l/dev/culXX -m dir

WARNINGS

Cu buffers input internally.

DEPENDENCIES

HP Clusters

Cu is not supported on client nodes of an HP Cluster.

AUTHOR

Cu was developed by AT&T and HP.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/lib/uucp/Dialers
/usr/spool/uucp/LCK..(tty-device)
/dev/null

SEE ALSO

cat(1), ct(1), echo(1), stty(1), uname(1), uucp(1), uuname(1).

UUCP, tutorial in *HP-UX Concepts and Tutorials.*

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cu* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

cu: SVID2, XPG2, XPG3

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut –**c***list* [*file1 file2* ...]
cut –**f***list* [–**d***char*] [–**s**] [*file1 file2* ...]

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (–**c** option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (–**f** option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

list — A comma-separated list of integer field numbers (in increasing order), with optional – to indicate ranges as in the –**o** option of *nroff/troff* for page ranges; e.g., **1,4,7**; **1–3,8**; **–5,10** (short for **1–5,10**); or **3–** (short for third through last field).

–**c***list* — The *list* following –**c** (no space) specifies character positions (e.g., –**c1–72** would pass the first 72 characters of each line).

–**f***list* — The *list* following –**f** is a list of fields assumed to be separated in the file by a delimiter character (see –**d**); e.g., –**f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless –**s** is specified.

–**d***char* — The character following –**d** is the field delimiter (–**f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted. Adjacent field delimiters delimit null fields.

–**s** — Suppresses lines with no delimiter characters in case of –**f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the –**c** or –**f** option must be specified.

Hints

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

Cut does not expand tabs. Input should be piped through *expand*(1) if tab expansion is required.

EXAMPLES

cut –d: –f1,5 /etc/passwd
mapping of user ID to names

name='who am i | cut –f1 –d" "'
to set **name** to current login name.

DIAGNOSTICS

line too long — A line can have no more than 1023 characters or fields.

bad list for c/f option — Missing –**c** or –**f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

no fields — The *list* is empty.

SEE ALSO

grep(1), paste(1).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cut* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

The delimiter specified with the −d argument just be a single-byte character. Otherwise, single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

cut: SVID2, XPG2, XPG3

NAME

cxref – generate C program cross-reference

SYNOPSIS

cxref [*options*] *files*

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross-reference table. *Cxref* utilizes a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **–c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference. Output is sorted in ascending collation order (see Environment Variables below).

In addition to the **–D**, **–I** and **–U** options (which are identical to their interpretation by *cc*(1)), the following *options* are interpreted by *cxref*:

–c	Print a combined cross-reference of all input files.
–w<*num*>	Width option, which formats output no wider than <*num*> (decimal) columns. This option defaults to 80 if <*num*> is not specified or is less than 51.
–o file	Direct output to the named *file*.
–s	Operate silently; does not print input file names.
–t	Format listing for 80-column width.

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

EXAMPLES

The command:

cxref -c orange.c blue.c color.h

creates a combined cross-reference of the files **orange.c**, **blue.c**, and **color.h**.

The command:

cxref -c -o rainbow.x orange.c blue.c

creates a combined cross-reference of the files **orange.c**, **blue.c**, and **color.h**, and directs the output to the file **rainbow.x**.

WARNINGS

Cxref considers a formal argument in a *#define* macro definition to be a declaration of that symbol. For example, a program that *#include*s **ctype.h** will contain many declarations of the variable **c**.

DEPENDENCIES

Series 300

Cxref uses a special version of the C compiler front end. The size of the internal compiler tables can be adjusted by using the **-W***c* and **-N** options, as described in the manual page for *cc*(1).

FILES

/usr/lib/xcpp special version of C-preprocessor.

/usr/lib/xpass special version of C compiler front end.

SEE ALSO

cc(1).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *cxref* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

STANDARDS CONFORMANCE

cxref: SVID2, XPG2, XPG3

NAME

date – print or set the date and time

SYNOPSIS

date [*mmddhhmm*[*yy*]] [*+format*]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise the current date is set, provided you are superuser. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

date 10080045

sets the date to Oct 8, 12:45 a.m. The current year is the default if no year is mentioned.

The system operates in Coordinated Universal Time (UCT); *date* takes care of conversion to and from local standard and daylight time (see Environment Variables below).

Attempting to set the date backwards generates a warning, and requires an extra confirmation from the (super)user.

When *date* is used to set the date, a pair of date change records is written to the file **/etc/wtmp**.

In the HP Clustered environment, the date and time are automatically set when the system comes up as a cluster node. Setting the date and time from any node in a cluster will set the date and time on all nodes in the cluster.

If the argument begins with +, the output of *date* is under the control of the user as specified by *format*. The *format* string consists of zero or more directives and ordinary characters. A directive consists of a % character, an optional field width and precision specification, and a terminating character that determines the directive's behavior. All ordinary characters are copied unchanged into the output string and the output string is always terminated with a new-line character.

If no argument is given, a default format string of **%c** is used.

Directives

The following directives, shown without the optional field width and precision specification, are replaced by the indicated characters:

%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%c	current date and time representation
%d	day of the month as a decimal number [01,31]
%E	combined Emperor/Era name and year
%H	hour (24-hour clock) as a decimal number [00,23]
%I	hour (12-hour clock) as a decimal number [01,12]
%j	day of the year as a decimal number [001,366]
%m	month as a decimal number [01,12]

%M minute as a decimal number [00,59]

%n new-line character

%N Emperor/Era name

%o Emperor/Era year

%p equivalent of either AM or PM

%S second as a decimal number [00,59]

%t tab character

%U week number of the year (Sunday as the first day of the week) as a decimal number [00,53]

%w weekday as a decimal number [0(Sunday),6]

%W week number of the year (Monday as the first day of the week) as a decimal number [00,53]

%x current date representation

%X current time representation

%y year without century as a decimal number [00,99]

%Y year with century as a decimal number

%Z time zone name (or no characters if time zone cannot be determined)

%% %

The following directives are provided for backward compatibility. It is recommended that the directives above be used in preference to those below.

%D date in usual US format (%m/%d/%y) (use **%x** instead)

%F full month name (use **%B** instead)

%h abbreviated month name (use **%b** instead)

%r time in 12-hour US format (%I:%M:%S [AM|PM]) (use **%X** instead)

%T time in 24-hour US format (%H:%M:%S) (use **%X** instead)

%z time zone name (or no characters if time zone cannot be determined) (use **%Z** instead)

If a directive is not one of the above, the behavior is undefined.

Field Width and Precision

An optional field width and precision specification may immediately follow the initial % of a directive in the following order:

[−|**0**]*w* the decimal digit string *w* specifies a minimum field width in which the result of the conversion is right- or left-justified. It is right-justified (with space padding) by default. If the optional flag '−' is specified, it is left-justified with space padding on the right. If the optional flag '**0**' is specified, it is right-justified and padded with zeros on the left.

.p the decimal digit string *p* specifies the minimum number of digits to appear for the **d**, **H**, **I**, **j**, **m**, **M**, **o**, **S**, **U**, **w**, **W**, **y** and **Y** directives, and the maximum number of characters to be used from the **a**, **A**, **b**, **B**, **c**, **D**, **E**, **F**, **h**, **n**, **N**, **p**, **r**, **t**, **T**, **x**, **X**, **z**, **Z** and **%** directives. In the first case, if a directive supplies fewer digits than specified by the precision, it will be expanded with leading zeros. In the second case, if a directive supplies more characters than specified by the

precision, excess characters will truncated on the right.

If no field width or precision is specified for a **d**, **H**, **I**, **m**, **M**, **S**, **U**, **W**, **y** or **j** directive, a default of ".2" is used for all but **j** for which ".3" is used.

DIAGNOSTICS

No permission if you are not the superuser and you try to change the date;

bad conversion if the date set is syntactically incorrect;

bad format character
if the field directive is not recognizable.

EXAMPLES

For example,

date ´+DATE: %m/%d/%y%nTIME: %H:%M:%S´

might generate the following as output:

DATE: 10/08/87
TIME: 12:45:05

Using the date as set in the example given under DESCRIPTION and LC_TIME set to "german",

date ´%-4.4h %2.1d %H:%M´

might generate the following:

Okt 8 12:45

Here, the month field is four bytes long, flush-left, and space-padded on the right if the month name is shorter than four bytes. The day field is two bytes long, with leading zeros suppressed.

WARNINGS

It is a bad practice to change the date while the system is running multi-user.

The **A** format option was formerly **W** in HP-UX, but was changed for ANSI compatability.

AUTHOR

Date was developed by AT&T and HP.

FILES

/etc/wtmp

SEE ALSO

stime(2), ctime(3C), strftime(3C), tztab(4), environ(5), langinfo(5), lang(5).

EXTERNAL INFLUENCES

Environment Variables

TZ determines the conversion between the system time in UCT and the time in the user's local time zone (see *environ*(5) and *tztab*(4)). TZ also determines the content (that is, the time-zone name produced by the **%z** and **%Z** directives) of date and time strings output by the *date* command.

If TZ is not set in the environment or is set to the empty string, a default of **EST5EDT** is used.

LC_TIME determines the content (for example, weekday names produced by the **%a** directive) and format (for example, current time representation produced by the **%X** directive) of date and time strings output by the *date* command.

LC_CTYPE determines the interpretation of the bytes within the *format* string as single and/or multi-byte characters.

LC_NUMERIC determines the characters used to form numbers for those directives that produce numbers in the output. The characters used are those defined by ALT_DIGITS (see *langinfo*(5)).

LANG determines the language in which messages (other than the date and time strings) are displayed.

If LC_TIME or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *date* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

date: SVID2, XPG2, XPG3

NAME

dc – desk calculator

SYNOPSIS

dc [*file*]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.) The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. An end of file on standard input or the **q** command stop dc. The following constructions are recognized:

number
: The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9 or A-F. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ − / * % ^
: The top two values on the stack are added (+), subtracted (−), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored and a warning generated. The remainder is calculated according to the current scale factor; it is not the integer modulus function. **7 % 3** yields .1 (one tenth) if scale is 1 because **7 / 3** is 2.3 with .1 as the remainder.

s*x*
: The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x*
: The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d
: The top value on the stack is duplicated.

p
: The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ASCII string, removes it, and prints it.

f
: All values on the stack are printed.

q
: exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x
: treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X
: replaces the number on the top of the stack with its scale factor.

[...]
: puts the bracketed ASCII string onto the top of the stack. Strings may be nested by using nested pairs of brackets.

<*x* **>***x* **=***x* **!<***x* **!>***x* **!=***x*
: The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v
: replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! interprets the rest of the line as an HP-UX system command. (unless the next character is <, >, or =, in which case appropriate relational operator above is used).

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input.

I pushes the input base on the top of the stack.

o The top value on the stack is popped and used as the number radix for further output. See below for notes on output base.

O pushes the output base on the top of the stack.

k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

K pushes the scale factor on the top of the stack.

z The stack level is pushed onto the stack.

Z replaces the number on the top of the stack with its length.

? A line of input is taken from the input source (usually the terminal) and executed.

; : are used by *bc* for array operations. **Y** generates debugging output for *dc* itself.

The input base may be any number, but only the digits 0–9 and A–F are available for input, thus limiting the usefulness of bases outside the range 1-16. All 16 possible digits may be used in any base; they always take their conventional values.

The output base may be any number. Bases in the range of 2-16 generate the "usual" results, with the letters A–F representing the values from 10 through 16. Bases 0 and 1 generate a string of **1**'s whose length is the value of the number. Base -1 generates a similar string consisting of **d**'s. Other bases have each "digit" represented as a (multi-digit) decimal number giving the ordinal of that digit. Each "digit" is signed for negative bases. "Digits" are separated by spaces. Given the definition of output base, the command **Op** will always yield "10" (in a representation appropriate to the base); **O1-p** yields useful information about the output base.

EXAMPLES

This example prints the first ten values of n! (n factorial):

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1).

DC: An Interactive Desk Calculator, in *HP-UX Concepts and Tutorials: Shells and Miscellaneous Tools.*

DIAGNOSTICS

x is unimplemented	Where *x* is an octal number.
stack empty	There are insufficient elements on the stack to do what was asked.
Out of space	The free list is exhausted (too many digits).
Out of headers	Too many numbers are being kept around.
Out of pushdown	Too many items are on the stack.
Nesting Depth	There are too many levels of nested execution.

NAME

dd – convert, reblock, translate, and copy a (tape) file

SYNOPSIS

dd [*option=value*] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. Input and output block size can be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	Input file name; default is standard input.
of=*file*	Output file name; default is standard output. The output file will be created using the same owner and group used by creat().
ibs=*n*	Input block size is *n* bytes (default 512).
obs=*n*	Output block size is *n* bytes (default 512).
bs=*n*	Set both input and output block size to the same size, superseding **ibs** and **obs**. This option is particularly efficient if no conversion is specified, because no in-core copy is necessary.
cbs=*n*	Conversion buffer size is *n bytes*.
skip=*n*	Skip *n* input blocks before starting copy.
seek=*n*	Seek *n* blocks from beginning of output file before copying.
count=*n*	Copy only *n* input blocks.
conv=*option*	Data conversion option. Use one of the following:

conv=ascii	Convert EBCDIC to ASCII.
conv=ebcdic	Convert ASCII to EBCDIC
conv=ibm	Convert ASCII to EBCDIC using an alternate conversion table
conv=lcase	Map US ASCII alphabetics to lowercase
conv=ucase	Map US ASCII alphabetics to uppercase
conv=swab	Swap every pair of bytes
conv=noerror	Do not stop processing on an error
conv=sync	Pad every input block to input block size (ibs)
conv=...,...	Multiple comma-separated conversions

Where sizes are specified, a number of bytes is expected. Numbers can end with **k**, **b**, or **w** which specify multiplication by 1024, 512, or 2, respectively. To indicate a product, separate a pair of numbers with **x**

The **cbs** option is used only if **ascii** or **ebcdic** conversion is specified. In the former case, *cbs* characters are placed into the conversion buffer, converted to ASCII, trailing blanks are trimmed, and a new-line is added before sending the line to the output. In the latter case, ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks are added to make up an output block of size *cbs*.

Upon completion, *dd* reports the number of whole and partial input and output blocks.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occured.

DIAGNOSTICS

f+p blocks in(out) numbers of full and partial blocks read(written)

EXAMPLES

This command reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the ASCII file **x**:

```
dd if=/dev/rmt/0m of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magnetic tape. *Dd* is especially suited to I/O on raw physical devices because it allows reading and writing in arbitrary block sizes.

WARNINGS

You may experience trouble writing directly to or reading directly from a cartridge tape. For best results, use *tcio*(1) as an input or output filter. For example, use

```
... | dd ... | tcio -ovVS 256 /dev/rct/c0
```

for output to a cartridge tape, and

```
tcio -ivS 256 /dev/rct/c0 | dd ... | ...
```

for input from a cartridge tape.

Some devices, such as 1/2-inch magnetic tapes, are incapable of seeking. Such devices must be positioned prior to running *dd* by using *mt*(1) or some other appropriate command.

ASCII and EBCDIC conversion tables are taken from the 256-character ACM standard, Nov, 1968. The *ibm* conversion, while less widely accepted as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-line characterss are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

SEE ALSO

cp(1), mt(1), tr(1).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

dd: SVID2, XPG2, XPG3

NAME

delta − make a delta (change) to an SCCS file

SYNOPSIS

delta [−r*SID*] [−**s**] [−**n**] [−g*list*] [−**m**[*mrlist*]] [−**y**[*comment*]] [−**p**] *files*

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of − is given, the standard input is read (see WARNINGS). Each line of the standard input is taken to be the name of an SCCS file to be processed.

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see −**m** and −**y** keyletters below).

Keyletter arguments apply independently to each named file.

−r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *get*s for editing (**get** −**e**) on the same SCCS file were done by the same person (login name). The *SID* value specified with the −**r** keyletter can be either the *SID* specified on the *get* command line or the *SID* to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified *SID* is ambiguous, or, if necessary and omitted on the command line.

−**s** Suppresses the issue, on the standard output, of the created delta's *SID* as well as the number of lines inserted, deleted and unchanged in the SCCS file.

−**n** Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

−g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (*SID*) created by this delta.

−**m**[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

If −**m** is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see −**y** keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that the MR numbers were not all valid).

−**y**[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If −**y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-

line character terminates the comment text.

−p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*-file are explained in the *SCCS User's Guide*. The naming convention for these files is also described there. All files below except the g-file are created in the same directory as the s-file. The g-file is created in the user's working directory.

g-file	Existed before the execution of *delta*; removed after completion of *delta* (unless **−n** was specified).
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

Lines beginning with an **SOH** ASCII character (octal 001) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS (see *sccsfile*(4)) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get*/*delta* sequences should be used.

If the standard input (−) is specified on the *delta* command line, the **−m** (if necessary) and **−y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4).

SCCS User's Guide in *HP−UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *delta* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

STANDARDS CONFORMANCE

delta: SVID2, XPG2, XPG3

NAME

deroff − remove nroff, tbl, and neqn constructs

SYNOPSIS

deroff [−**m**x] [−**w**] [−**i**] [*file* ...]

DESCRIPTION

Deroff reads each *file* in sequence and removes all *nroff* requests, macro calls, backslash constructs, *neqn*(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl*(1) descriptions, replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *Deroff* follows chains of included files (**.so** and **.nx** *nroff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The −**m** option can be followed by an **m**, **s**, or **l**. The −**mm** option causes the macros be interpreted so that only running text is output (that is, no text from macro lines.) The −**ml** option forces the −**mm** option and also causes deletion of lists associated with the **mm** macros.

If the −**w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that contains at least two letters and is composed of letters, digits, ampersands (**&**), and apostrophes ('); in a macro call, however, a "word" is a string that begins with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

If the −**i** option is given, *deroff* ignores the **.so** and **.nx** *nroff* commands.

WARNINGS

Deroff is not a complete *nroff* interpreter; thus it can be confused by subtle constructs. Most such errors result in too much rather than too little output.
The −**ml** option does not handle nested lists correctly.

AUTHOR

Deroff was developed by the University of California, Berkeley.

SEE ALSO

neqn(1), nroff(1), tbl(1).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text and filenames as single and/or multi-byte characters. Note that multi-byte punctuation characters are not recognized when using the -w option.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *deroff* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported except in the case of the −w option which supports only single-byte code sets.

NAME

diff, diffh – differential file comparator

SYNOPSIS

diff [**–befh**] *file1 file2*

/usr/lib/diffh [**–b**] *file1 file2*

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is –, the standard input is used. If *file1* (*file2*) is a directory, a file in that directory named *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed*(1) commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs (where *n1* = *n2* or *n3* = *n4*) are abbreviated as a single number.

By default, each line affected in *file1* is flagged by <; each line affected in *file2* is flagged by >.

Options

–b Cause trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equally.

–e Produce a script of **a**, **c**, and **d** commands for the editor *ed*(1) to recreate *file2* from *file1*.

–f Produce a script similar to that of **–e**, but in the opposite order. The **–f** option is not useful with *ed*(1).

–h Do a fast, half-hearted job. The **–h** option works only when changed segments of text are short and well-separated, although it does work on files of unlimited length. Options **–e** and **–f** are unavailable with **–h**.

Diffh is equivalent to *diff* **–h**. It must be invoked as shown above in the SYNOPSIS, unless the PATH variable in your environment includes the directory **/usr/lib**.

The following shell program might help maintain multiple versions of a file when using the **–e** option. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be available. A "latest version" appears as the standard output.

```
(shift; cat $*; echo '1,$p') | ed – $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

DIAGNOSTICS

Exit status is **0** for no differences, **1** for some differences, **2** for trouble.

EXAMPLES

The following command produces a list of editor commands that when interpreted by *ed*(1), recreate **new_site** from **site** if **new_site** is an updated copy of **site**.

```
diff -e site new_site
```

WARNINGS

Editing scripts produced under the **–e** or **–f** option are naive about creating lines consisting of a single period (.).

Missing newline at end of file X indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output, although the output will seem to

indicate they are the same.

AUTHOR

Diff was developed by AT&T, the University of California, Berkeley, and HP.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

bdiff(1), cmp(1), comm(1), diff3(1), diffmk(1), dircmp(1), ed(1), more(1), nroff(1), rcsdiff(1), sccsdiff(1), sdiff(1), terminfo(4).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the space characters for the *diff* command.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *diff* and *diffh* behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that *diff* and *diffh* do not recognize multi-byte alternative space characters.

STANDARDS CONFORMANCE

diff: SVID2, XPG2, XPG3

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**–ex3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f : *n1* **a**	Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.
f : *n1* , *n2* **c**	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the **–e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3. Option **–x** (**–3**) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to *file1*.

```
(cat script; echo '1,$p') | ed – file1
```

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat **–e**.
Files longer than 64K bytes will not work.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

diffmk – mark differences between files

SYNOPSIS

diffmk name1 name2 name3

DESCRIPTION

Diffmk compares two versions of a file and creates a third file that includes "change mark" commands for *nroff*(1) or *troff*. *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file **macs** contains:

```
.pl 1
.ll 77
.nf
.eo
```

The **.ll** request might specify a different line length, depending on the nature of the program being printed. The **.eo** request is probably needed only for C programs.

If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them (*diffmk* is a shell procedure).

SEE ALSO

diff(1), nroff(1).

BUGS

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, i.e., replacing **.sp** by **.sp 2** will produce a "change mark" on the preceding or following line of output.

Although unlikely, certain combinations of formatting requests may cause change marks to either disappear or to mark too much. Manual intervention may be required as the subtleties of all the various formatting macro packages and preprocessors is beyond the scope of *diffmk*. The input to *tbl*(1) cannot tolerate **.mc** commands. Any **.mc** that would appear inside a **.TS** range will be silently deleted. The script can be changed if this action is inappropriate or *diffmk* can be run on the output from *tbl*(1).

Diffmk uses *diff*(1) and thus has whatever limitations on file size and performance that *diff* may impose. In particular the performance is non-linear with the size of the file, and very large files (well over 1000 lines) may take extremely long to process. Breaking the file into smaller pieces may be advisable.

Diffmk also uses *ed*(1), and if the file is too large for *ed*, *ed* error messages may be imbedded in the file. Again, breaking the file into smaller pieces may be advisable.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

dircmp – directory comparison

SYNOPSIS

dircmp [**−d**] [**−s**] [**−w***n*] *dir1 dir2*

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Sorted listings of files that are unique to each directory are generated for all the options. If no option is entered, a sorted list is output indicating whether the filenames common to both directories have the same contents.

−d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

−s Suppress messages about identical files.

−w*n* Change the width of the output line to *n* characters. The default width is 72.

EXAMPLES

The command:

dircmp -d slate sleet

compares the two directories **slate** and **sleet** and produces a list of changes that would make the directories identical.

SEE ALSO

cmp(1), diff(1).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *dircmp* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

STANDARDS CONFORMANCE

dircmp: SVID2, XPG2, XPG3

NAME

dos2ux, ux2dos – convert ASCII file format

SYNOPSIS

dos2ux *file* ...
ux2dos *file* ...

DESCRIPTION

Dos2ux and *ux2dos* read each specified *file* in sequence and write it to standard output, converting to HP-UX format or to DOS format, respectively. Each *file* can be either DOS format or HP-UX format for either command.

A DOS file name is recognized by the presence of an embedded colon (:) delimiter; see *dosif*(4) for DOS file naming conventions.

If no input file is given or if the argument – is encountered, *dos2ux* and *ux2dos* read from standard input. Standard input can be combined with other files.

EXAMPLES

The following prints file **myfile** on the display:

dos2ux myfile

The following converts **file1** and **file2** to DOS format, then concatenates them together, placing them in **file3**.

ux2dos file1 file2 > file3

RETURN VALUE

Both commands return **0** if successful or **2** if the command failed. The only possible failure is the inability to open a specified file, in which case the commands print a warning.

WARNINGS

Command formats resembling:

dos2ux file1 file2 > file1

overwrite the data in **file1** before the concatenation begins, causing a loss of the contents of **file1**. Therefore, be careful when using shell special characters.

SEE ALSO

doschmod(1), doscp(1), dosdf(1), dosls(1), dosmkdir(1), dosrm(1), dosif(4).

NAME

doschmod − change attributes of a DOS file

SYNOPSIS

doschmod [−**u**] *mode device*:*file* ...

DESCRIPTION

Doschmod is the DOS counterpart of *chmod*(1).

There is one option:

−**u** Disable argument case conversion. In the absence of this option, all DOS file names are converted to upper case.

A DOS file name is recognized by the presence of an embedded colon (:) delimiter; see *dosif*(4) for DOS file naming conventions.

The attributes of each named file are changed according to *mode*, which is an octal number in the range 000 to 0377. *Mode* is constructed from the logical OR of the following modes:

200	Reserved. Do not use.
100	Reserved. Do not use.
040	Archive. Set whenever the file has been written to and closed.
020	Directory. Do not modify.
010	Volume Label. Do not modify.
004	System file. Marks files that are part of the DOS operating system.
002	Hidden file. Marks files that do not appear in a DOS directory listing using the DOS DIR command.
001	Read-Only file. Marks files as read-only.

SPECIAL WARNING

Specifying inappropriate *mode* values can make files and/or directories inaccessible, and in certain cases can damage the file system. To prevent such problems, do not change the mode of directories and volume labels.

Normal users should have no need to use *mode* bits other than 001, 002, and 040.

EXAMPLES

The following marks file **/dev/rfd9122:memo.txt** as a hidden file:

doschmod 002 /dev/rfd9122:memo.txt

The following marks file **driveC:autoexec.bat** read-only:

doschmod 001 driveC:autoexec.bat

SEE ALSO

chmod(1), dos2ux(1), doscp(1), dosdf(1), dosls(1), dosmkdir(1), dosrm(1), chmod(2), dosif(4).

NAME

doscp – copy to or from DOS files

SYNOPSIS

doscp [–**fvu**] *file1 file2*
doscp [–**fvu**] *file1* [*file2* ...] *directory*

DESCRIPTION

Doscp is the DOS counterpart of *cp*(1). *Doscp* copies a DOS file to a DOS or HP-UX file, an HP-UX file to an HP-UX or DOS file, or HP-UX or DOS files to an HP-UX or DOS directory. The last name in the argument list is the destination file or directory.

A DOS file name is recognized by the presence of an embedded colon (**:**) delimiter; see *dosif*(4) for DOS file naming conventions.

The file name – (dash) is interpreted to mean standard input or standard output depending upon its position in the argument list.

Options

There are several options:

–f Unconditionally write over an existing file. In the absence of this option, *doscp* asks permission to overwrite an existing HP-UX file.

–v Verbose mode. *Doscp* prints the source name.

–u Disable argument case conversion. In the absence of this option, all DOS file names are converted to upper case.

Note: Shell metacharacters (*, **?**, and **[...]**) can be used when specifying HP-UX file names, but cannot be used when specifying a DOS file name, because file name expansion is done by the shell and the DOS utilities do not recognize metacharacters.

RETURN VALUE

Doscp returns 0 if all files are copied successfully. Otherwise, it prints a message to standard error and returns with a non-zero value.

EXAMPLES

Copy the files in the HP-UX directory **abc** to the DOS volume stored as HP-UX file **hard_disk**:

doscp abc/* hard_disk:

Copy DOS file **/backup/log** through the HP-UX special file **/dev/rfd9127** to HP-UX file **logcopy** located in the current directory:

doscp /dev/rfd9127:/backup/log logcopy

Copy DOS file **zulu** on the volume stored as HP-UX file **bb** to standard output:

doscp bb:zulu –

SEE ALSO

cp(1), dos2ux(1), doschmod(1), dosdf(1), dosls(1), dosmkdir(1), dosrm(1), dosif(4).

NAME

dosdf – report number of free disk clusters

SYNOPSIS

dosdf *device*[**:**]

DESCRIPTION

Dosdf is the DOS counterpart of *df*(1). It prints the cluster size in bytes and the number of free clusters on the specified DOS volume.

SEE ALSO

df(1), dos2ux(1), doschmod(1), doscp(1), dosls(1), dosmkdir(1), dosrm(1), dosif(4).

NAME

dosls, dosll – list contents of DOS directories

SYNOPSIS

dosls [**–aAudl**] *device*:[*file*]
dosll [**–aAudl**] *device*:[*file*]

DESCRIPTION

Dosls is the DOS counterpart of *ls*(1).

For each directory named, *dosls* lists the contents of that directory. For each file named, *dosls* repeats its name and any other information requested. If invoked by the name *dosll*, the **–l** option is implied.

Options

There are several options:

–a List all directory entries. In the absence of this option, hidden files, system files, and files whose names begin with a dot (**.**) are not listed.

–A Same as **–a**, except the current directory and the parent directory are not listed. For the superuser, this option defaults to being set, and is disabled by **–A**.

–u Disable argument case conversion. In the absence of this option, all DOS file names are converted to uppercase.

–d If an argument is a directory, list only its name. Often used with **–l** to get the status of a directory.

–l List in long format, giving file attribute, size in bytes, and the date and time of last modification for each file, as well as listing the DOS volume label. Long listing is disabled if *dosll* is invoked with the **–l** option.

A DOS file name is recognized by the presence of an embedded colon (**:**) delimiter; see *dosif*(4) for DOS file naming conventions.

EXAMPLES

These examples assume that a DOS directory structure exists on the device accessed through HP-UX special file **/dev/rdsk/0s1**.

The following example lists all of the files in the root directory of the DOS directory structure:

dosls –a /dev/rdsk/0s1:

The following example produces a long-format listing of all the information about the DOS directory **/dos/math**, but does not list the files in the directory:

dosls –ld /dev/rdsk/0s1:/dos/math

SEE ALSO

dos2ux(1), doschmod(1), doscp(1), dosdf(1), dosmkdir(1), dosrm(1), ls(1), dosif(4).

NAME

dosmkdir – make a DOS directory

SYNOPSIS

dosmkdir [**–u**] *device*:*directory* ...

DESCRIPTION

Dosmkdir is the DOS counterpart of *mkdir*(1). It creates specified directories. The standard entries, **.** for the directory itself and **..** for its parent, are made automatically.

There is one option:

–u Disable argument case conversion. In the absence of this option, all DOS file names are converted to uppercase.

A DOS file name is recognized by the presence of an embedded colon (:) delimiter; see *dosif*(4) for DOS file naming conventions.

DIAGNOSTICS

Dosmkdir returns 0 if all directories were successfully created. Otherwise, it prints a message to standard error and returns non-zero.

EXAMPLES

To create an empty subdirectory named **numbers** under the directory **/math/lib** on the device accessed through HP-UX special file **/dev/rfd9122**, use:

dosmkdir /dev/rfd9122:/math/lib/numbers

SEE ALSO

dos2ux(1), doschmod(1), doscp(1), dosdf(1), dosls(1), dosrm(1), mkdir(1), dosif(4).

NAME

dosrm, dosrmdir – remove DOS files or directories

SYNOPSIS

dosrm [–**friu**] *device*:*file* ...
dosrmdir [–**u**] *device*:*file* ...

DESCRIPTION

Dosrm and *dosrmdir* are DOS counterparts of *rm*(1) and *rmdir*(1), respectively.

Dosrm removes the entries for one or more files from a directory. If a specified file is a directory, an error message is printed unless the optional argument –**r** is specified (see below).

Dosrmdir removes entries for the named directories, provided they are empty.

Options

The options are:

–**f** (force) Unconditionally remove the specified file, even if the file is marked read-only.

–**r** Cause *dosrm* to recursively delete the entire contents of a directory, followed by the directory itself. *Dosrm* can recursively delete up to 17 levels of directories.

–**i** (interactive) Cause *dosrm* to ask whether or not to delete each file. If –**r** is also specified, *dosrm* asks whether to examine each directory encountered.

–**u** Disable argument case conversion. In the absence of this option, all DOS file names are converted to uppercase.

A DOS file name is recognized by the presence of an embedded colon (:) delimiter; see *dosif*(4) for DOS file naming conventions.

EXAMPLES

These examples assume that a DOS directory structure exists on the device accessed through the HP-UX special file **/dev/rfd9122**.

This example recursively combs through the DOS directory **/tmp** and asks if each DOS file should be removed (forced, with no file mode checks):

dosrm –irf /dev/rfd9122:/tmp

The following example removes the DOS directory **doug** from the DOS volume stored as HP-UX file **hard_disk**:

dosrmdir hard_disk:doug

SEE ALSO

dos2ux(1), doschmod(1), doscp(1), dosdf(1), dosls(1), dosmkdir(1), rm(1), rmdir(1), dosif(4).

NAME

du – summarize disk usage

SYNOPSIS

du [**–sabr**] [**–t** *type*] [*names*]

DESCRIPTION

Du gives the number of 512-byte blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, **.** is used.

The optional argument **–s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **–a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

The **–b** argument prints out the number of blocks the swap system is currently using and the name of the directory the swap system is associated with.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. **–r** causes *du* to generate messages in such instances.

Du normally reports disk usage for the entire directory hierarchy below the given *names*. The **–t** *type* argument can be used to restrict reporting to file systems of the specified type. The accepted types are **hfs**, **cdfs**, and **nfs** (see *checklist*(4)). Multiple **–t** *type* options can be specified.

A file with two or more links is only counted once.

EXAMPLES

The command:

du -r

displays disk usage for the current working directory and all directories below it, generating error messages for unreadable directories.

The command:

du -t hfs /

displays disk usage for the entire file system excluding any **cdfs** or **nfs** mounted file systems.

WARNINGS

If the **–a** option is not used, non-directories given as arguments are not listed.

Files with holes in them will get an incorrect block count.

SEE ALSO

df(1), bdf(1), checklist(4).

STANDARDS CONFORMANCE

du: SVID2, XPG2, XPG3

NAME

echo – echo (print) arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without appending a new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
n	the 8-bit character whose ASCII code is the 1-, 2-, 3- or 4-digit octal number *n*, whose first character must be a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

NOTES

Berkeley **echo** differs from this implementation. The former does not implement the backslash escapes. However, the semantics of the \c escape can be obtained by using the **-n** option. The echo command implemented as a built-in function of *csh*(1) follows the Berkeley semantics.

BUGS

No characters are printed after the first \c. This is not normally a problem.

EXTERNAL INFLUENCES

Environment Variables LC_CTYPE determines the interpretation of *arg* as single and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *echo* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

echo: SVID2, XPG2, XPG3

NAME

ed, red – text editor

SYNOPSIS

ed [–] [**–p** string] [**–x**] [file]

red [–] [**–p** string] [**–x**] [file]

REMARKS

The decryption facilities provided by this software are under control by the United States Government and cannot be exported without special licenses. The capabilities are only available by special arrangement through HP.

DESCRIPTION

Ed is the standard (line-oriented) text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited.

The optional – suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the **!** prompt after a **!***shell command*.

The **–p** option allows the user to specify a prompt string. If **–x** is present, an *x* command is simulated first to handle an encrypted file.

Ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via **!***shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty –tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (**.**)**alone**at**the** beginning of a line.

Ed supports the Basic Regular Expression (RE) syntax (see *regexp*(5)) with the following additions:

The null RE (e.g., //) is equivalent to the last RE encountered.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2
s/s1/s2/p

```
g/s1
g/s1/p
?s1
?s1?
```

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. A *'x* addresses the line marked with the mark name character *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (**/**) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before **FILES** below.
6. A RE enclosed in question marks (**?**) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before **FILES** below.
7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or −, the addition or subtraction is taken with respect to the current line; e.g, **−5** is understood to mean **.−5**.
9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so − − refers to the current line less 2.
10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the current line (**.**) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5 and 6 above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is respectively either listed, numbered, or printed, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that can be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also **DIAGNOSTICS** below.

E *file*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *f*ile-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE***/***command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All

lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before **FILES** below.

(1,$)G/*RE***/**

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an **&** causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lowercase letter. The address *'x* then addresses this line; . is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab*, *backspace*) are represented by (hopefully) mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see **DIAGNOSTICS** below).

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

($)r *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE***/***replacement***/**
or
(.,.)s/*RE***/***replacement***/g**
or
(.,.)s/*RE***/***replacement***/n**
n = 1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number n appears after the command, only the n th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character

other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before **FILES** below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*RE***/***command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE***/**

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

($)=

The line number of the addressed line is typed; . is unchanged by this command.

!*shell command*

The remainder of the line after the **!** is sent to the HP-UX shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line.

DIAGNOSTICS

? for command errors. **?***file* for an inaccessible file.
(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands. It prints **?** and allows one to continue editing. A second *e* or *q* command at this point will take effect. The – command-line option inhibits this feature.

FILES

/tmp/e#	temporary; # is the process number.
ed.hup	work is saved here if the terminal is hung up.

SEE ALSO

awk(1), crypt(1), edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), vi(1), fspec(4), lang(5), regexp(5).

The ed Editor, in *HP-UX Concepts and Tutorials.*

WARNINGS

A **!** command cannot be subject to a *g* or a *v* command.

The **!** command and the **!** escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).

The sequence **\n** in a RE does not match a new-line character.

The *l* command mishandles DEL.

Files encrypted directly with the *crypt*(1) command with the null key cannot be edited.

If the editor input is coming from a command file (i.e., **ed file < ed-cmd-file)**, the editor will exit at the first failure of a command that is in the command file.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as non-printing, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ed* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

ed: SVID2, XPG2, XPG3

red: SVID2, XPG2, XPG3

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [**–r**] name ...

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor. The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

BRIEF INTRODUCTION

To edit the contents of an existing file you begin with the command "edit name" to the shell. *Edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

Edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say "s/*old*/*new*/" where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No **write** since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "''". The **z** command can also be given other following characters "z−" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form **/text/** to search forward for *text* or **?text?** to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form **/^text/** which searches for *text* at the beginning of a line. Similarly **/text$/** searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$−5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. *Edit* has 26 such buffers named *a* through z. You can later get these lines back by doing "put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

AUTHOR

Edit was developed by AT&T. The 16-bit extensions to *edit* are based in part on software of the Toshiba Corporation.

SEE ALSO

ex(1), vi(1).

The *edit Tutorial* in the *Text Editors and Processors* volume of *HP-UX Concepts and Tutorials*.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

efl – Extended Fortran Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the *efl* language into clean Fortran on the standard output. *Efl* provides the C-like control constructs of *ratfor*(1):

statement grouping with braces.
decision-making:
if, **if-else**, and **select-case** (also known as **switch-case**);
while, **for**, Fortran **do**, **repeat**, and **repeat** ... **until** loops;
multi-level **break** and **next**.

Efl has C-like data structures, e.g.:

```
struct
        {
        integer flags(3)
        character(8) name
        long real coords(2)
        } table(100)
```

The language offers generic functions, assignment operators (+=, **&**=, etc.), and sequentially evaluated logical operators (**&&** and ||). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) } )
```

Efl also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers).
comments:
this is a comment.
translation of relational and logical operators:
>, >=, **&**, etc., become **.GT.**, **.GE.**, **.AND.**, etc.
return expression to caller from function:
return (*expression*)
defines:
define *name replacement*
includes:
include *file*

Efl understands several option arguments: **−w** suppresses warning messages, **−#** suppresses comments in the generated program, and the default option **−C** causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an *efl* option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on.

Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

Efl is best used with *f77*(1).

SEE ALSO

cc(1), f77(1), ratfor(1).

NAME

elm – process mail through screen-oriented interface

SYNOPSIS

elm [**–akKmz**] [**–f** *folder*]
elm [**–s** *subject*] *address-list*
elm **–h**

DESCRIPTION

Elm is screen-oriented electric mail processing system. In interactive use, the main header index and mini-menu of commands are displayed upon initial invocation and at any point when the program is waiting for input.

There are three main ways to use *elm*: explicitly to send a single message by invoking the program with a list of mail addresses, with the program then prompting for the subject, message body, and so on; as a convenient way to send files or the output of commands via command line redirection; or as an interactive mail interface program(see EXAMPLES).

The following options are recognized:

–a Arrow – force the arrow cursor (instead of the inverse bar).

–f *folder* File – read specified file rather than the incoming mailbox.

–h Help – give a list of starting options.

–k Softkeys off – disable use of softkeys(function keys).

–K Keypad + softkeys off – disable use of softkeys and arrow cursor keys. If your terminal doesn't have HP2622 function key protocols, you must use this option.

–m Menu off – Use the extra lines for more message headers.

–s *subject* Subject – specify subject for a message to mail.

–z Zero – don't enter *elm* if no mail is pending.

You can get screen-oriented interface when you invoke *elm* without address-list. Invoking *elm* as a interactive mailer, you get into command mode. In command mode, every command(see also COMMANDS) entered is executed for the message that has current message pointer(inverse bar or arrow) and some commands can operate on 'tagged' messages, too. So when you execute a command, you set the pointer to the message and input the character for the command.

On the header index page, status of the message is always indicated. The status field composed of three different character fields, with the first one indicating temporary status:

D For a *deleted* message.

E For an *expired* message. This flag is set according to the header field "*Expires:*". If the date of this field is older than the day, this flag will appear. The following date formats are examples of what is understood:
1). Mon, 11 Jun 88
2). Jun 11, 88
3). 11 Jun, 88
4). 880611HHMMZ <- X.400 format

N For a *new* message.

The second character denotes permanent status:

C For *confidential* mail. If "*Sensitivity: 3*" of the user defined header field is attached, this can appear and the message is considered company confidential,

as per the ISO X.400 standard.

U For *urgent* mail. This flag is set if the message contains a "*Priority:*".

P For *private* mail. This flag is associated with the header field "*Sensitivity:*" also. This is set for "*Sensitivity: 2*" of the user defined header field only.

A For messages that have an explicit action associated with them through inclusion of the "*Action: *" header field.

F For a *form* letter.

When a message has more than one status flag set of a particular type, the highest precedent indicator will be displayed on the index page. For example, of a *form* letter is also marked as *company confidential*, the 'C' will be displayed, rather than the 'F' status character.
The third character of the status field can be a "+" to indicated that the message is *tagged* too.(See also COMMANDS)

When sending a message, *elm* will use the editor specified in your *.elm/elmrc* file(see CUSTOMIZATION), the editor listed as $EDITOR in your environment, or the default of "**vi(1)**". If *'builtin'* is specified as your editor, you will also have a large set of commands available while composing your message (see TILDE ESCAPES).

If you have a **$HOME/.elm/elmheaders** file, *elm* will automatically read in the contents of the file and add it to the headers of all outbound mail(good for having an 'Organization:' field, or 'Phone:' field, etc). *Elm* also supports the use of backquotes in the *elmheaders* file, so an entry like:

Operating System: 'uname −srv'

will work fine.

Elm has its own alias system, and supports two classes of aliases; personal and system-wide. Personal aliases are specific to a single user and system aliases are good for everyone on a specific machine(see also **elmalias(1)**). To enter the elm alias mode, use the **A**)lias command at the main command prompt. From there, you will be able to create and save an alias for the current message, check personal and system-wide aliases, and other options(see ALIAS COMMANDS).

At invoking time, *elm* can read customized variables from file **$HOME/.elm/elmrc** to initialize parameters. This file can be saved from within *elm* and some of these variables can be modified also by command **O**)ption. (See also CUSTOMIZATION)

FORMS MODE

One feature that is unique to *elm* is the ability to compose and reply to *form* letters and other types of forms.

Creating a form message is quite simple: first you must enable forms mode by adding "**forms=ON**" to your **$HOME/.elm/elmrc**(see CUSTOMIZATION). Then, when you compose the message, each field to be filled in by recipient should have a colon(':') followed by either the number of spaces allowed for the field or a newline, which indicates fields through the end of the line. Additionally, if a colon appears on a line by itself, the recipient will be prompted for multi-line input.

Upon reception of a form message, the user is allowed to reply (but not group reply), at which time *elm* will prompt the user for each field, with any text present between the fields displayed as appropriate.

A simple form message is:

On-Line Phone and Address Database

Please fill out and return as soon as possible.

Name:

Manager:

Department: Division:

Your home address
:

home phone number:

Thank you for your cooperation.

COMMANDS

? Help. This command used once puts you in the *explain key* mode, where any key you press will result in a one-line description of the key. Pressed again at this point will produce a summary listing for each command available. <escape> or ' . ' will leave the help mode and return you to the main menu level.

! Shell. This allows you to send a command to the shell without leaving *elm*.

| Pipe. This command allows you to pipe the current message or the set of *tagged* messages through other filters as you desire. The shell used for the entire command will be either the one specified in your *.elm/elmrc* file, or, if none, */bin/sh*.

+ Next index page. This displays next header index page, if possible.

– Previous index page. This displays previous header index page, if possible.

= Set current to 1. This set current message pointer to first message.

***** Set current to last. This set current message pointer to last message.

/ Pattern match. This command, on the top level, allows you to search through all the *from* and *subject* lines of the current mailbox. If the first character of the pattern is a '/', then the program will try to match the specified pattern against *any* line in the mailbox. Both searches are case insensitive.

<n> Specify new current message. Typing in the number of the message will result in the *elm* program producing the prompt "Set current to : n", where 'n' is the number entered. Note that changing the current message to a message not on the current page of headers will result in a new page being displayed.

<return> Read current message. After entering, screen is cleared and the current message is displayed by the pager specified in system or **pager**(see also CUSTOMIZATION).

< Scan message for calendar entries. A rather novel feature of the *elm* mailer is the ability to automatically incorporate calendar/agenda information from a mail message into the user's calendar file. This is done quite simply; any message that has either the pattern;
-> calendar entry
or
– multi-line
– calendar entry
will be automatically added to the user's **calendar** file with stripping '->' or '–'

if the '<' command is used. (See also CUSTOMIZATION)

a Alias. The *alias* command is a way by which more complex mail addresses can be shortened for the mail user. In alias mode, it provides command **a** for making alias for current message, **p** for checking a user alias, **s** for checking system aliases, **m** for making a new alias and **d** for deleting alias (see also ALIAS COMMANDS).

b Bounce mail. This "remails" mail to someone else in such a way as to make the return address the original sender rather than you (as opposed to the *forward* command, which makes the return address *you* rather than the original sender).

c Change mailbox. Specifying this command allows you to change the mailbox file that is currently being read. As with the *save* command, this command expands filenames with '' being your home directory and '=' being your **maildir** directory, if defined.(See also CUSTOMIZATION) This command also allows the special character '!' to be used to allow you to change to the default incoming mailbox.

d, u Delete and Undelete. Neither of these two commands have any prompts and indicate their action by either adding a 'D' to the current message index entry (indicating deletion pending) or removing the 'D' (indicating that the message isn't queued for deletion).

<control>-D This command allows you to easily mark for deletion all messages that have a specific pattern. After <control>-D is pressed, the program will prompt for the string to match (it matches either the *from* or *subject* lines of the message).

<control>-U This is the direct opposite command to the previous. All messages that match the specified pattern can have any mark for deletion removed by using this command.

f Forward. Allows you to forward the current message to another person. This copies the message into the edit buffer and allows you to add their own message too. (see *bounce* above, too)

g Group reply. Identical to *reply* below, except that the response is mailed to *all recipients* of the original message (see also CUSTOMIZATION *alternatives*)

h Same as **<return>**, but message is displayed with all headers.

j, k These two keys work similarly to what they would do in **vi(1)** or any of the other screen oriented programs. The **j** key moves the current message pointer down to the next message and the **k** key moves back to the previous message.

l Limit. This specifies a subset of the existing messages to be dealt with. It's valid for subject, from and to field. To set the criteria, enter or To clear all the criteria and get back to the 'regular' display, simply enter "*all*" as the limiting criteria.

<control>-L Rewrite the screen. If the screen is confused, you can redraw screen with this command.

m Mail. Send mail to a specified user.

n Next message. See also **<return>**.

o Options. This allows you to alter the settings of a number of option values. (See also CUSTOMIZATION)

p Print. This allows you to print out the current message or the tagged messages to a previously defined printing method **print**.(See also CUSTOMIZATION)

q Quit. Quit *elm*, processing messages as you like. When reading your incoming mail, you can choose to keep the undeleted mail in the incoming mailbox or move it to an 'mbox' file specified by **maildir** in **$HOME/.elm/elmrc** file. You can also decide whether or not to actually delete messages previously marked for deletion. (See also CUSTOMIZATION)

<control>-Q ,DEL Exit. Same as '**x**'. This allows you to leave *elm* in the quickest possible manner without any changing the *mailbox*.

r Reply. Reply to the author of the current message. If the **autocopy**(see also CUSTOMIZATION) is not specified, you can specify whether a copy of the source message is to be copied into the edit buffer, or not. If copied in, all lines from the message are with the prefix character sequence defined as **prefix** (see also CUSTOMIZATION).

s Save to file. This command allows the current message or set of tagged messages to be copied into a user specified file or folder. After saving a file, each message is marked for deletion and, if saving just one message, the current message pointer is incremented.

t Tag. Tag the current message for operation '**|**', '**p**' or '**s**'.

<control>-T Tag all messages containing the specified pattern. Since tagging messages can occur on screens other than the one being viewed, the *elm* will first check to see if any messages are currently *tagged* and ask you if you'd like to remove those tags. After that, it will, similar to the **<control>-D** function to set criteria(see **<control>-D**).

x Exit. This allows you to leave *elm* in the quickest possible manner without any changing the *mailbox*.

TILDE ESCAPE

~? Print a brief help menu.

~b Change the Blind-Carbon-Copy list.

~c Change the Carbon-Copy list.

~e Invoke $EDITOR in your environment on the message, if possible.

~f *options* Add the specified list of messages, or current. This uses **readmail(1)** and all options for **readmail(1)** are available.

~h Change all the available headers (To, Cc, Bcc, and Subject)

~m *options* same as **~f**, but with the current 'prefix'.

~o Invoke a user specified editor on the message.

~p print out the message as typed in so far.

~r *filename* Include (read in) the contents of the specified file.

~s Change the Subject line.

~t Change the To list.

~v Invoke $VISUAL in your environment on the message if possible.

~< *command* execute the specified shell command, entering the output of the command into the editor buffer upon completion. (for example "**~<** who" will include the

output of the *who(1)* command in your message)

~! *command* execute a shell command if one is given (as in "~!ls") or give you a shell (either your shell setting as a **shell**(see also CUSTOMIZATION) or $SHELL in your environment).

~~ Add a line prefixed by a single "~" character.

ALIAS COMMANDS

a Alias current message. This allows you to create an alias that has the return address of the current message as the address field of the alias. It prompts for a unique alias name.

d Delete user alias. This prompts for alias name to be deleted. The alias is deleted from your alias_text file (**$HOME/.elm/aliases.text**).

m Make user alias. This will prompt for a unique alias name and then for an address. The information provided will be added to your individual alias_text file (**$HOME/.elm/aliases.text**) and then added to the database.

p Check personal alias. This is a simple way of checking what is in the alias database. It prompts for an alias name, and returns the address or the list of addresses associated with that name or the error message "alias not found" as appropriate.

s Check system alias. This is for checking what aliases are installed yet as system aliases. All the system aliases are listed.

r Return. Return to the main level of *elm* program.

CUSTOMIZATION

Like many other programs, *elm* also has the ability to read in a 'rc' file automatically for personal configuration. The file should be called **$HOME/.elm/elmrc** and can contain any combination of the following.

String Variables

alternatives This is a list of other machine/username combinations that you receive mail from (forwarded). This is used when the *group reply* feature is invoked to ensure that you don't send yourself a copy of the outbound message. No default.

calendar Name of calendar file. This is used in conjunction with the scan message for calendar entries command '<'. Default is **$HOME/calendar**.

editor The editor to use when typing in new mail. You can also select **none** or **builtin** for builtin editor. Builtin editor is available for all mail that doesn't already have text in the buffer(in replying, mailing with a **signature**, etc). Default is the **$EDITOR** in your current environment, or **vi(1)** if not.

escape Escape character used in **builtin** editor. Default is tilde('~').

fullname This is the name the mailer will use when sending mail from you. Default is the "gecos" field from the */etc/passwd* file.

mailbox This is where to put incoming mail after you've read it. When you answer *no* ('n') to the "keep messages in incoming mailbox?" prompt, this is where the messages go. Default is **$HOME/mbox**.

maildir This is the default mail directory, and is used to expand filenames in *elm* when specified using the '=' metacharacter. That is, if you save to file *=/archive* then the '=' will be expanded to the current value of **maildir**. Default is **$HOME/Mail**.

pager This defines the program to be used to display each message. This can be changed while within the *elm* program by selecting the appropriate entry in the **O**)ption Menu. Default is *'builtin'*.

prefix Value of prefix for included line. When you *reply* to a message or *forward* a message to another person, you can optionally include the original message. This prefix indicates the included line. Default is "> ".

print The command to run when **p**)rint command is executed. This indicates how to print out a message. There are two possible formats for this string, either a command that can have a filename affixed to (as a suffix) and then sent to the system for execution, or a string that contains the meta-sequence '%s' which will be replaced by the name of the message file and then also sent to the shell. Default is "pr %s | lp".

savemail This is where outgoing mail will have a copy silently saved. This will only be used if the **copy** flag is turned on. Also note that if the **savename** feature is enabled then this filename may be ignored since the program first looks for a mailbox that has the same name as the login of the person you're sending to, using that instead if found. Default is **$HOME/mbox**.

shell This defines the shell to use when doing '!' escapes and such. Default is **$SHELL** in your current environment.

signature This file, if defined, will be automatically appended to all outbound mail before the editor is invoked. Furthermore, if you'd like a different "signature" file for *local* mail and *remote* mail (remote being via other hosts), you can alternatively define two variables, *localsignature* and *remotesignature*, to have the same functionality. No default.

sortby When reading mailboxes, either incoming or specified, you can have them sorted by any number of different ways. This can be changed without leaving the *elm* by changing the *S)orting criteria field* in **o**)ption mode, but it can also be predefined to any of *'from'*, *'sent'*, *'received'*, *'subject'*, *'lines'* or *'status'*. Each of these fields can also optionally be prefixed with the sequence *'reverse–'* to reverse the order of the sort. Default is **received**.

weedout When specifying this option, you can then list headers that you don't want to see when you are reading mail. This is effective with **weed** is ON. This list can continue for as many lines as desired, as long as the continued lines all have leading indentation. Default is "*>From*" "*In-Reply-To:*" "*References:*" "*Newsgroups:*" "*Received:*" "*Apparently-To:*" "*Message-Id:*" "*Content-Type:*" "*From*" "*Mailer:*".

Numeric Variables

timeout This is the interval between resynchronizing. *Elm* internally resynchronize every this interval seconds. Default is 600 seconds (10 minutes).

userlevel This is what the program uses to determine the relative level of sophistication of the user. The values are 0 for a new user (the default), 1 for someone familiar with *elm* user, and 2 for experts.

Boolean Variables

alwaysdelete When set, this changes the default answer of the prompt "Delete messages?" to the indicated value. Default is ON for YES.

alwaysleave This changes the default answer on the "keep mail in incoming mailbox?" prompt to the value indicated. Default is ON for YES.

arrow This is identical in function to the '−a' command line option. Default is OFF.

ask This allows you to tell the *elm* that you would rather not be asked "Delete message?" and such each time you leave the program, and instead it should just use the values of **alwaysdelete** and **alwaysleave** without prompting. Default is ON.

askbcc If turned on, it will appear the prompt "Blind-Copies-To:" for each message. If this is off, user can add "bcc" list by ~**b** in the builtin editor or using header editor. Default is OFF.

askcc If turned off, this allows you to send mail without being presented the "Copies-To:" prompt for each message. This still allows the user to explicitly include addresses in the "cc" list via either ~**c** in the builtin editor, or via using the screen-oriented header editor. Default is ON.

autocopy This is a boolean flag, and if set automatically copies the text of the message you are replying to into the edit buffer. Default is OFF.

copy This, in combination with the **savemail** option, will allow you to have silent copies of all outgoing mail made on the outbound step. Default is OFF.

expand If this flag is on, tabs in your message written are expanded to spaces. This will enable that your message is displayed in the same layout as you write on the screen having defferent tab setting. Default is ON.

forms This allows you to mail forms. Default is OFF.

keep The mail system has a habit of deleting mailboxes when you have removed everything from them. With this option turned on, it will instead preserve them as zero-byte files. Default is OFF.

keypad If on, this tells *elm* that you have an HP terminal and enables the <NEXT>, <PREV>, <HOME> and <SHIFT-HOME> keys. Default is ON.

menus If turned off, this will inhibit the menu display on all of the screen displays within the *elm* program. Default is ON.

movepage If this is enabled then commands that move through the mailbox by pages (the + and − keys) will also move the current message pointer to the top of that page of messages. If this is turned off then moving through the pages doesn't alter the current message pointer location. Default is OFF.

names Show only the user names when expanding aliases, rather than the name and electronic mail address on the "To:" field when you send mail. Default is OFF.

noheader This tells the mailer not to include the headers of messages when copying a message into a file buffer for replying to or forwarding. Default is ON.

pointnew If this is turned on, the mailer will be automatically pointing to the first new message in your mailbox when started, instead of at message #1. This is only used for the incoming mailbox since other mailboxes are assumed not to have 'new' and 'old' mail. Default is ON.

resolve With this option enabled, as soon as mail is 'dealt with' *elm* moves you to the next message in the mailbox, with deletion, undeletion, saving a message and forwarding a message. Default is ON.

savename When the user *saves* the messages, *elm* uses filename to save to that is the *login name* of the person who sent you the message rather than *savemail* value. Similarly, when sending mail out, instead of just blindly saving it to the *savemail* file, *elm* will first try to save it to a file that is the *login name* of the

person who is to receive the mail. If the file doesn't already exist on outbound mail, then it will save the mail in the *savemail* file. Default is ON.

skipdeleted If this flag is on, current message pointer skips the message with deleted flag "D" by **j** or **k** command. Default is OFF.

softkeys If on, this tells *elm* that you have an HP terminal with the HP 2622 function key protocol and that you would like to have them available while in the program. Default is ON.

titles Using with the flag **weed**, this flag allows you to have the first line of a message titled with:
Message *N/M* from *username"date* at *time*
where all the information has been previously extracted from the message. Default is ON.

warnings The mailer likes to warn you when you send mail to a machine that cannot be directly accessed. Setting this flag will allow you to effectively shut off all the warning messages. Default is ON.

weed This is a boolean flag that, in combination with the **weedout** list, allows you to custom define the set of headers you would like to not have displayed while reading messages. Default is ON.

EXAMPLES

If you invoke *elm* with a list of addresses to send to, you can send a message to them without any overhead of the main system being loaded. *Elm* will then prompt for Subject, Copies, Blind-Copies, and then drop you in your editor to compose the message:

$ elm taylor

To: taylor(Dave Taylor)

Subject: **this is test**

Copies To: <**return**>

Blind-Copies To: <**return**>

...invokes editor, message composed, then...

Your options now are:

S)end the message, E)dit it again, change/add H)eaders or F)orget it

What is your choice? **s**

mail sent!

Elm can send files or the output of commands via command line redirection:

$ elm taylor < help.c

This will read in the file and transmit it to the specified people. A subject may be specified with "−s *subject*" as an option to *elm*:

$ elm −s "File help.c transmission" taylor < help.c

AUTHOR

elm was developed by Hewlett-Packard Company.

FILES

/usr/mail Directory for incoming mail
(mode "755", group ID "mail")

/usr/mail/*username*.lock
Lock for mail directory

/usr/mail/*username*
Incoming mailbox for the *user*.
(mode "660", group ID "mail")

/usr/lib/nls/C/elm.cat
Location of the message catalogue

$HOME/.elm Directory for *elm*

$HOME/.elm/elmrc
Personal customized file

$HOME/.elm/elmheaders
Contents of additional headers.

/usr/lib/elm/elm_help.0
Help file for main screen

/usr/lib/elm/elm_help.1
Help file for alias screen

/usr/lib/elm/elm_help.2
Help file for option screen

/usr/lib/elm/elmrc-info
Comment file for .elm/elmrc file

/usr/mail/.elm Directory for system alias of *elm*

/usr/mail/.elm/aliases.hash
System alias hash table

/usr/mail/.elm/aliases.data
System alias data table

/usr/mail/.elm/aliases.text
System alias text file

$HOME/.elm/aliases.hash
User alias hash table

$HOME/.elm/aliases.data
User alias data table

$HOME/.elm/aliases.text
User alias text file

/tmp/snd.*pid* and sndh.*pid*
Outgoing mail edit buffer

/tmp/form.*pid* Editor buffer for form message

/tmp/print.*pid* Temporary file for printing message

/tmp/alias.*pid* Temporary file for deleting alias

$HOME/.elm/readmail
Used by **readmail(1)** program

/tmp/mbox.*logname*
Temporary mbox for *user*

$HOME/Cancelled.mail
Cancelled message of non-interactive use.

SEE ALSO

elmalias(1), mailfrom(1), newmail(1), readmail(1), vi(1).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

elmalias – create and verify elm user and system aliases

SYNOPSIS

elmalias [**–q**]
elmalias **–c** *alias-list*
elmalias **–l** [*regular–expression*]

DESCRIPTION

Elmalias is the utility for setting, checking and listing aliases. If no options are given to the program it will attempt to install a new set of personal aliases.
Possible starting options are:

–c *alias-list* Check alias – check aliases in personal or system alias file.

–l *regular-expression*
List alias – list aliases available for the user.

–q Set system aliases – install the system-level alias set for all elm users on the machine. This command is only available for system administrators.

To install aliases, the program will either look for a file called **$HOME/.elm/aliases.text** (for personal aliases) or a file **/usr/mail/.elm/aliases.text** (if invoked by a system administrator). Installation is complete when the program has created two other elm data files: **aliases.hash** and **aliases.data**, both in the same directory as the **aliases.text** file.

The format of alias-text file is:

alias [, *alias*, ...] = [*fullname* =] *address* [, *address*, ...]

Addresses can be explicit routing or addressing information for a specific mailbox (eg. "user@host.domain") or another alias. If more than one alias is listed as the first part of the line, all will have the same value. If more than one address is listed, the specific alias will be assumed to be a group rather than individual alias. (See EXAMPLES)

Invoking with option **–c** checks the alias indicated by *alias-list*. This checks the user file first, and then the system alias file to try to find the specified alias or aliases. If found, the value of the alias is printed out, otherwise an error message is generated.

Finally, invoking with option **–l** outputs a line per alias in both the user and the system aliases. Each line is of the form:

alias *address* (*fullname*)

If an optional *regular-expression* is used, just the aliases that match the specified expression are listed. If not, they are all listed. *regular expression* is the same as **egrep(1)** of the shell.
The output is always sorted alphabetically.

EXAMPLES

A simple example **aliases.text** file:

```
# sample alias file
mom                = my_mother@a.computer
dad,father,pop = Father = host!otherhost!dad
parents            = mom, dad
```

listing this file would result in:

```
dad    host!otherhost!dad (Father)
father host!otherhost!dad (Father)
```

```
mom     my_mother@a.computer
parents mom,dad
pop     host!otherhost!dad (Father)
```

AUTHOR

elmalias was developed by Hewlett-Packard Company.

FILES

$HOME/.elm/aliases.text
Alias source for user

$HOME/.elm/aliases.hash
Alias hash table for user

$HOME/.elm/aliases.data
Alias data file for user

/usr/mail/.elm/aliases.text
Alias source for system

/usr/mail/.elm/aliases.hash
Alias hash table for system

/usr/mail/.elm/aliases.data
Alias data file for system

SEE ALSO

elm(1), egrep(1).

WARNING

Note that the precedence of aliases is user file then system file. This means that a user can 'overload' an alias by having one defined in the system file also defined in theirs. This shouldn't turn out to be a problem, but is something for the system administrator to keep in mind when creating the system alias file.

NAME

enable, disable − enable/disable LP printers

SYNOPSIS

enable *printers*
disable [−**c**] [−**r**[*reason*]] *printers*

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers are reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

−**c** Cancel any requests that are currently printing on any of the designated printers.

−**r**[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next −**r** option. If the −**r** option is not present or the −**r** option is given without a reason, a default *reason* is used. *Reason* is reported by *lpstat*(1). The maximum length of the *reason* message is 80 bytes.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

EXAMPLES

The command:

enable snowwhite

enables the printer **snowwhite** to accept requests.

The command:

disable -c snowwhite

deactivates the printer **snowwhite** and cancels any logged jobs.

WARNINGS

If the restrict cancel feature (selected by the −**orc** option in *lpadmin*(1M)) is enabled, *disable* ignores the −**c** option.

Enable and *disable* perform their operation on the local system (or HP cluster) only.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), lp(1), lpadmin(1M), lpsched(1M), lpstat(1), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *enable* and *disable* behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support
Single- and multi-byte character code sets are supported.

NAME

env − set environment for command execution

SYNOPSIS

env [−] [−**i**] [*name=value*] ... [*command* [*arguments* ...]]

DESCRIPTION

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The −**i** flag causes the inherited environment to be ignored completely so that the command is executed with exactly the environment specified by the arguments. The − flag has the same effect as the −**i** flag.

If no command is specified, the resulting environment is printed; one name-value pair per line.

WARNING

The − flag is an obsolete option to env. Use −**i** instead.

SEE ALSO

sh(1), exec(2), profile(4), environ(5).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

env: SVID2, XPG2, XPG3

NAME

ex − text editor

SYNOPSIS

ex [−] [**−v**] [**−r**] [**−R**] [*+command*] [**−l**] [**−t** *tag*] [**−V**] [**−w***size*] [**−x**] [*file ...*]

REMARKS

The decryption facilities provided by this software are under the control of the United States Government and cannot be exported without special licenses. These capabilities are only available by special arrangement through HP.

DESCRIPTION

The command *ex* is a line oriented text editor, which supports both command and display editing (see *vi*(1)). The command line options are:

−	Suppress all interactive-user feedback. This is useful in processing editor scripts.
−v	Invoke *vi*.
−r	Recover *file* or files after an editor or system crash. If no *file* is specified a list of all saved files is printed.
−R	Set "read-only" mode to prevent overwriting a file inadvertently.
+command	Begin editing by executing the specified editor search or positioning *command*.
−l	*Lisp* mode; indents appropriately for *lisp* code; the **()** {} **[[** and **]]** commands in *vi* are modified to be meaningful for *lisp*.
−t *tag*	Edit the file containing the *tag* and position the editor at its definition (see the **tag** command below).
−V	Verbose mode; editor commands are displayed as they are executed when input from a user's **.exrc** file or a source file (see the **source** command below).
−w*size*	Set the value of the **window** editor option to *size*.
−x	Encryption mode; the user is prompted for a key to initiate creation or editing of an encrypted file (see the **crypt** command below).

The *file* argument(s) indicates files to be edited, in the order specified.

The name of the file being edited by *ex* is the current file. The text of the file is read into a buffer and all editing changes are performed in this buffer; changes have no effect on the file until the buffer is written out explicitly.

The *alternate* file name is the name of the last file mentioned in an editor command, or the previous current file name if the last file mentioned becomes the current file. The character % in file names is replaced by the current file name, and the character **#** by the alternate file name.

The named buffers ASCII **a** through **z** can be used for saving blocks of text during the edit. If the buffer name is specified in upper case, the buffer is appended to rather than being overwritten.

The read-only mode can be cleared from within the editor by setting the **noreadonly** edit option (see Edit Options below). Writing to a different file is allowed in read-only mode; in addition, the write can be forced by using **!** (see the **write** command below).

If an interrupt signal is received, *ex* returns to the command level. If the editor input comes from a file, *ex* exits at the interrupt.

If the system crashes or *ex* aborts due to an internal error or unexpected signal, *ex* attempts to preserve the buffer if any unwritten changes were made. The command line option **−r** is used

to retrieve the saved changes.

At the beginning, *ex* is in the command mode, which is indicated by the colon (:) prompt. The input mode is entered by **append**, **insert**, or **change** commands; it is left (and command mode reentered) by typing a period (.) alone at the beginning of a line.

Command lines beginning with the double quote character (") are ignored. (This may be used for comments in an editor script.)

Multiple commands can be combined on a single line by separating them with a vertical bar character (|). However, global commands, comments, and the shell escape command must be the last command on a line because they are not terminated by the | character.

Addressing

.	Dot (.) refers to the current line. There is always a current line; the positioning might be the result of an explicit movement by the user, or the result of a command that affected multiple lines (in which case it is usually the last line affected).
n	The *n*th line in the buffer, with lines numbered sequentially from 1.
$	The last line in the buffer.
%	Abbreviation for **1,$**; the entire buffer.
+*n* −*n*	An offset relative to the current line. (For example, the forms .+3, +3, and +++ are equivalent.)
/re/ ?*re*?	Line containing the pattern *re*, scanning forward (*/re/*) or backward (?*re*?). The trailing / or ? can be omitted if the line is only being printed. If the pattern is omitted, *ex* uses the more recently set of either the substitution string or scanning string (see Regular Expressions below).
´*x*	Lines can be marked using single ASCII lowercase letters (see the **mark** command below); ´*x* refers to line marked *x*. In addition, the previous current line is marked before each non-relative motion; this line may be referred to by using ´ for *x*.

Addresses to commands consist of a series of line addresses (specified as above), separated by , or ;. Such address lists are evaluated left-to-right. When ; is the separator, the current line is set to the value of the previous address before the next address is interpreted. If more addresses are given than the command requires, then all but the last one or two are ignored. Where a command requires two addresses, the first line must precede the second one in the buffer. A null address in a list defaults to the current line.

Command Names and Abbreviations

The command names enclosed within parenthesis in the table below are available only in their abbreviated forms.

abbreviate	**ab**	open	**o**	unabbreviate	**una**
append	**a**	next	**n**	undo	**u**
args	**ar**	number	**# nu**	unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
chdir	**chd cd**	print	**p**	visual	**vi**
copy	**co t**	put	**pu**	write	**w**
crypt	**cr X**	quit	**q**	xit	**x**
delete	**d**	read	**r**	yank	**ya**
edit	**e ex**	recover	**rec**	(window)	**z**

file	**f**	rewind	**rew**	(escape)	**!**
global	**g v**	set	**se**	(lshift)	<
insert	**i**	shell	**sh**	(rshift)	>
join	**j**	source	**so**	(scroll)	^**D**
list	**l**	stop	**st** ^**Z**	(line no)	=
map		substitute	**s & ~**	(exec buffer)	* **@**
mark	**k ma**	suspend	**su** ^**Z**		
move	**m**	tag	**ta**		

Command Descriptions

In the following, *line* is a single-line address, given in any of the forms described in the Addressing section above; *range* is a pair of line addresses, separated by a comma or semicolon (see the Addressing section for the difference between the two); *count* is a positive integer, specifying the number of lines to be affected by the command; *flags* is one or more of the characters **#**, **p**, and **l**; the corresponding command to print the line is executed after the command completes. Any number of + or − characters can also be given with these flags.

When *count* is used, *range* is ineffective; instead, only a line number should be specified to indicate the first line affected by the command. (If a range is given, the last line of the range is interpreted as the starting line for the command.)

These modifiers are all optional; the defaults are as follows, unless otherwise stated: the default for *line* is the current line; the default for *range* is the current line only (.,.); the default for *count* is 1; the default for *flags* is null.

When only a *line* or a *range* is specified (with a null command), the implied command is **print**; if a null line is entered, the next line is printed (equivalent to **.+1p**)

abbreviate **ab[breviate]** *word rhs*

Add the named abbreviation to the current list. In visual mode, if *word* is typed as a complete word during input, it is replaced by the string *rhs*.

append *line* **a[ppend][!]**

Enter input mode; the input text is placed after the specified line. If line 0 is specified, the text is placed at the beginning of the buffer. The last input line becomes the current line, or the target line if no lines are input.

Appending the command with a **!** causes the **autoindent** edit option setting to be toggled for this insert only.

args **ar[gs]**

The argument list is printed, with the current argument inside **[** and **]**.

change *range* **c[hange][!]** *count*

Enters input mode; the input text replaces the specified lines. The last input line becomes the current line; if no lines are input, the effect is the same as a delete.

Appending the command with a **!** causes the **autoindent** edit option setting to be toggled for this insert only.

chdir **chd[ir][!]** [*directory*]
cd[!] [*directory*]

Change the working directory to *directory*. If *directory* is omitted, the value of the HOME environment variable is used. If the buffer has been modified since the last write and the name of the file being edited does not begin with a slash

(/), a warning is issued and the working directory is not changed. To force a change of directory in this case, append the character **!** to the command.

copy *range* **co[py]** *line flags*
range **t** *line flags*

A copy of the specified lines (*range*) is placed after the specified destination *line*; line 0 specifies that the lines are to be placed at the beginning of the buffer. (The letter **t** is an alternative abbreviation for the **copy** command.)

crypt **cr[ypt]**
X

The user is prompted for a key with which to enter encryption mode. This command can also be used to change the key entered with a previous **crypt** command or the **-x** command line option. If no key is supplied in response to the prompt (that is, only RETURN is pressed), encryption mode is canceled and the buffer is written out in plain-text form by subsequent write commands.

While in encryption mode, all file input is decrypted using the current key. However, while an input file is being processed, if a block of text (approximately 1024 bytes) is encountered that contains only 7-bit ASCII characters, that block of text is assumed to be plain-text and is not decrypted. All file output, except that piped via a **!** shell escape to another command, is encrypted using the current key.

The temporary file used by the editor to manage the buffer is not encrypted until the current buffer is discarded (or written out) and editing begins on a new file. When creating a new file that requires encryption protection, ensure that the buffer file is also encrypted by specifying the **-x** option when invoking the editor.

delete *range* **d[elete]** *buffer count*

The specified lines are deleted from the buffer. If a named *buffer* is specified, the deleted text is saved in it. If no buffer is specified, the unnamed buffer is used (that is, the buffer where the most recently deleted or yanked text is placed by default). The line after the deleted lines becomes the current line, or the last line of the file if the deleted lines were at the end of the file.

edit **e[dit][!]** [*+line*] *file*
ex[!] [*+line*] *file*

Begin editing a new file (**ex** is an alternative name for the **edit** command). If the current buffer has been modified since the last write, a warning is printed and the command is aborted. This action can be overridden by appending the character **!** to the command (**e!** *file*). The current line is the last line of the buffer; however, if this command is executed from within *visual*, the current line is the first line of the buffer. If the *+line* option is specified, the current line is set to the specified position, where *line* may be a number (or **$**) or specified as */re* or **?***re*.

file **f[ile]**

Print the current file name and other information, including the number of lines and the current position.

global *range* **g[lobal][!]** */re/ command ...*

range **v** */re/ command ...*

Perform *command* on lines within *range* (or on the entire buffer if no *range* is given) that contain *re*. First mark the lines within the given *range* that match the pattern *re*. If the pattern is omitted, the more recently set of either the substitution string or the scanning string is used (see Regular Expressions below). Then the given *command*(s) is executed with **.** set to each marked line. Any character other than a letter or a digit can be used to delimit the pattern instead of the **/**.

The *command*(s) can be specified on multiple lines by hiding new-lines with a backslash. If *command*(s) is omitted, each line is printed. **Append**, **change**, and **insert** commands are allowed; the terminating dot can be omitted if it ends *command*(s). The **visual** command is also permitted (unless the **global** command itself has been issued from visual mode), and takes input from the terminal. (If *command* contains a visual mode command (that is, **open** or **visual**), the visual mode command must be terminated by the visual mode command **Q** in order to proceed to the next marked line.)

The **global** command itself and the **undo** command are not allowed in *command*(s). The edit options **autoprint**, **autoindent**, and **report** are inhibited.

Appending a **!** to the **global** command (that is, **g!** ...) or using the alternate name **v** causes the *command*(s) to be run on the lines within *range* that do not match the pattern.

insert *line* **i[nsert][!]**

Enter input mode; the input text is placed before the specified line. The last line input becomes the current line, or the line before the target line, if no lines are input.

Appending the command with a **!** causes the **autoindent** edit option to be toggled for this insert only.

join *range* **j[oin][!]** *count flags*

Join together the text from the specified lines into one line. White space is adjusted to provide at least one blank character (two if a period appears at the end of a line, or none if the first character of a line is a closing parenthesis (**")"**)). Extra white space at the beginning of a line is discarded.

Appending the command with a **!** causes a simpler join with no white space processing.

list *range* **l[ist]** *count flags*

Print the specified lines with tabs displayed as **^I** and the end of each line marked with a trailing **$**. (The only useful flag is **#** for line numbers.) The last line printed becomes the current line.

map **map[!]** [*x* | **#***n*] *rhs*

The **map** command is used to define macros for use in visual mode. The first argument, *x*, can be a single character or a multicharacter sequence. In the special sequence, **#***n*, *n* is a digit referring to the function key **n**. When *x* or the function key corresponding to **#***n* is typed in visual mode, **map** interprets the action as though *rhs* were typed. If **!** is appended to the command **map**, the mapping is effective during input mode rather than command mode. Special characters, white space, and new-line must be escaped with a **^V** to be entered in the arguments. The first argument cannot contain the colon (**:**) as the first

character, nor can multicharacter sequences begin with an alphabetic character. See also the Edit Options **timeout**, **timeoutlen**, **keyboardedit**, and **keyboardedit!** below.

mark *line* **ma[rk]** *x*
line **k** *x*

(The letter **k** is an alternative abbreviation for the **mark** command.) The specified line is given the specified mark *x*, which must be a single ASCII lowercase letter (a-z). (The *x* must be preceded by a space or tab.) The current line position is not affected.

move *range* **m[ove]** *line*

Move the specified lines (*range*) to follow the target *line*. The first line moved becomes the current line.

next **n[ext][!]** [*file ...*]

The next file from the command line argument list is edited. Appending a **!** to the command overrides the warning about the buffer having been modified since the last write (and discards any changes unless the **autowrite** edit option is set). The argument list can be replaced by specifying a new one on this command line.

number *range* **nu[mber]** *count flags*
range **#** *count flags*

(The character **#** is an alternative abbreviation for the **number** command.) Print the lines, each preceded by its line number. (The only useful flag is **l**.) The last line printed becomes the current line.

open *line* **o[pen]** */re/ flags*

Enter open mode, which is similar to visual mode with a one-line window. All the visual mode commands are available. If a match is found for the optional regular expression in *line*, the cursor is placed at the start of the matching pattern. The visual mode command **Q** exits open mode. For more information, see *vi*(1).

preserve **pre[serve]**

The current editor buffer is saved as though the system had just crashed. This command is used in emergencies, for example when a write does not work and the buffer cannot be saved in any other way.

print *range* **p[rint]** *count*

Print the specified lines, with non-printing characters printed as control characters in the form ^**x**; DEL is represented as ^?. The last line printed becomes the current line.

put *line* **pu[t]** *buffer*

Place deleted or "yanked" lines after *line*. A buffer can be specified; otherwise, the text in the unnamed buffer (that is, the buffer in which deleted or yanked text is placed by default) is restored.

quit **q[uit]**

Terminate the edit. If the buffer has been modified since the last write, a warning is printed and the command fails. This warning can be overridden by appending a **!** to the command (discarding changes).

read *line* **r[ead]** *file*

Place a copy of the specified **file** in the buffer after the target line (which can be line 0 to place text at the beginning). If no **file** is named the current file is the default. If no current file exists, **file** becomes the current file. The last line read becomes the current line; in visual mode, the first line read becomes the current line.

If **file** is given as **!string**, **string** is interpreted as a system command and passed to the command interpreter; the resultant output is read into the buffer. A blank or tab must precede the **!**.

recover **rec[over][!]** *file*

Recover **file** from the save area, after an accidental hangup or a system crash. If the current buffer has been modified since the last write, a warning is printed and the command is aborted. This action can be overridden by appending the character **!** to the command (**rec!** *file*).

rewind **rew[ind][!]**

The argument list is rewound, and the first file in the list is edited. Any warnings may be overridden by appending a **!**.

set **se[t]** *parameter*

With no arguments, the **set** command prints those options whose values have been changed from the default settings; with the parameter **all** it prints all option values.

Giving an option name followed by a **?** causes the current value of that option to be printed. The **?** is necessary only for Boolean-valued options. Boolean options are given values by the form **se** *option* to turn them on, or **se no***option* to turn them off; string and numeric options are assigned by the form **se** *option=value*. More than one parameter can be given; they are interpreted left to right.

See Edit Options below for further details about options.

shell **sh[ell]**

The user is put into the command interpreter specified by the **shell** edit option (see below). Editing is resumed on exit from the command interpreter.

source **so[urce]** *file*

Read and execute commands from the specified **file**. The **so** commands can be nested.

substitute
range **s[ubstitute]** */re/repl/ options count flags*
range **s** *options count flags*
range **&** *options count flags*
range **sr** *options count flags*
range **~** *options count flags*
range **s\?***repl*
range **s\&***repl*

On each specified line, the first instance of the pattern *re* is replaced by the string *repl*. (See Regular Expressions and Replacement Strings below.) Any character other than a letter or a digit can be used to delimit the pattern instead of the **/**. If *options* includes the letter **g** (global), all instances of the pattern in the line are substituted. If the option **c** (confirm) is included, the

user is queried about whether to perform each individual substitution, as follows: Before each substitution the line is typed with the pattern to be replaced marked with carat characters (^); a response of **y** causes the substitution to be performed, while any other input aborts it. The last line substituted becomes the current line.

If the substitution pattern *re* is omitted (**s**//*repl*/), the more recently set of either the substitution string or the scanning string is used (see Regular Expressions below).

If the alternative forms **s** or **&** of the command are used, the substitution pattern defaults to the previous substitution string and the replacement string defaults to the previous replacement string used.

If the alternative forms **sr** or ~ of the command are used, the substitution pattern defaults to the more recently set of either the substitution string or the scanning string and the replacement string defaults to the previous replacement string used.

The alternative form **s**\?*repl* is equivalent to **s**/*scan-re*/*repl*/, where *scan-re* is the previous scanning string.

The alternative form **s**\&*repl* is equivalent to **s**/*subs-re*/*repl*/, where *subs-re* is the previous substitution string.

suspend **su[spend][!]**
stop **st[op][!]**
^Z

Suspend the editor job and return to the calling shell. ^Z and the **stop** command are alternative names for the **suspend** command. ^Z is the user process control suspend character which is typically the character control-Z (ASCII SUB) (see *stty*(1)). This command is disabled if the calling shell does not support job control or has disabled it.

The buffer is written to the current file before the editor is suspended if the **autowrite** editor option is set, the **readonly** editor option is not set, and the buffer has been modified since the last write. This action can be overridden by appending the character **!** to the **suspend** or **stop** command.

tag **ta[g][!]** *tag*

The files specified by the **tags** edit option (see below) are searched sequentially until a tag definition with the name of *tag* is found. If found, edit the file and position the editor at the address specified by the tag definition.

The buffer is written to the current file before the new file is edited if a file different from the current file is being edited, the **autowrite** editor option is set, the **readonly** editor option not set, and the buffer has been modified since the last write. This action can be overridden by appending the character **!** to the command.

unabbreviate **una[bbreviate]** *word*

Delete *word* from the list of abbreviations (see the **abbreviate** command above).

undo **u[ndo]**

Reverse the changes made by the previous editing command. For this purpose, **global** and **visual** are considered single commands. Commands that affect the external environment, such as **write**, **edit** and **next**, cannot be undone. An

undo can itself be reversed.

unmap **unm[ap][!]** *x*

The macro definition for *x* is removed (see the **map** command above).

version **ve[rsion]**

Print the current version of the editor.

visual *line* **vi[sual]** *type count flags*

Enter visual mode at the specified *line*. The *type* is optional, and can be one of the designated characters +, −, ., or ^, as in the **z** command, to specify the position of the specified line on the screen window. (The default is to place the line at the top of the screen window.) A *count* specifies an initial window size; the default is the value of the edit option **window**. The flags **#** and **l** cause lines in the visual window to be displayed in the corresponding mode (see the **number** and **list** commands). The command **Q** exits visual mode. For more information, see *vi*(1).

write *range* **w[rite][!][>>]** *file*
range **wq[!][>>]** *file*

Write the specified lines (or the entire buffer, if no *range* is given) out to *file*, printing the number of lines and characters written. If *file* is not specified, the default is the current file. (The command fails with an error message if there is no current file and no file is specified.)

If an alternate file is specified and the file exists, the write fails, but can be forced by appending a **!** to the command. An existing file can be appended to by appending >> to the command. If the file does not exist, an error is reported.

If the file is specified as **!string**, **string** is interpreted as a system command; the command interpreter is invoked, and the specified lines are passed as standard input to the command.

The command **wq** is equivalent to a **w** followed by a **q**; **wq!** is equivalent to **w!** followed by **q**; and **wq>>** is equivalent to **w>>** followed by **q**.

xit **x[it][!][>>] file**

If changes have been made to the buffer, a **write** command is executed with any options (such as **!**, >>, or *file*) used by the **write** command. Then (in any case) the **quit** command is executed.

yank *range* **ya[nk]** *buffer count*

Place the specified lines in the named *buffer*. If no buffer is specified, the unnamed buffer is used (that is, the buffer where the most recently deleted or yanked text is placed by default).

(window) *line* **z** *type count flags*

The number of lines specified by *count* are displayed. The default for *count* is the value of the edit option **window**.

If *type* is omitted, *count* lines following the specified *line* (default current line) are printed.

If *type* is specified, it must be one of the following designated characters: +, −, ., ^, or =. A + displays a window of lines following the addressed line, a − causes the addressed line to be placed at the bottom of the window of

displayed lines, a . causes the addressed line to be placed at the center of the window, a ^ displays a window of lines that are two windows prior to the addressed line, and = displays the addressed line at the center of the window with a line of dashes above and below the addressed line.

The last line printed becomes the current line, except for the = type, in which case the addressed line becomes the current line.

(escape) ! *command*

The remainder of the line after the **!** is passed to the system command interpreter for execution. A warning is issued if the buffer has been changed since the last write. A single **!** is printed when the command completes. The current line position is not affected.

Within the text of *command*, **%** and **#** are expanded as filenames, and **!** is replaced with the text of the previous **!** command. (Thus **!!** repeats the previous **!** command.) When such expansion is performed, the expanded line is echoed.

(escape) *range* **!** *command*

In this form of the **!** command, the specified lines (there is no default; see previous paragraph) are passed to the command interpreter as standard input; the resulting output replaces the specified lines.

(lshift) *range* < *count flags*

Shift the specified lines to the left; the number of spaces to be shifted is determined by the edit option **shiftwidth**. Only white space (blanks and tabs) is lost in shifting; other characters are not affected. The last line changed becomes the current line.

(rshift) *range* > *count flags*

Shift the specified lines to the right by inserting white space (see previous paragraph for further details).

(scroll) **^D**

Control-D (ASCII EOT) prints the next *n* lines, where *n* is the value of the edit option **scroll**.

(line no) *line* = *flags*

Print the line number of the specified **line** (default last line). The current line position is not affected.

(exec buffer) * [*buffer*]

Execute the contents of the named buffer as an editor command. There is no difference between the * and the **@** forms of this command. If a buffer is not specified or *buffer* is * or **@**, the buffer last named in the **exec buffer** command is executed.

Regular Expressions

The editor maintains copies of two regular expression strings at all times: the substitution string and the scanning string. The substitute command sets the substitution string to the regular expression used. Both the global command and the regular-expression form of line addressing (see Addressing above) for all commands set the scanning string to the regular expression used. These strings are used as default regular expressions as described under Addressing, the **global** command, and the **substitute** command.

The editor supports Basic Regular Expressions (see *regexp*(5)) with the following modifications:

\< The \< matches the beginning of a "word"; that is, the matched string must begin in a letter, digit, or underline, and be preceded by the beginning of the line or a character other than the above.

\> The \> matches the end of a "word" (see previous paragraph).

~ Match the replacement part of the last **substitute** command.

[*string***]** The positional quoting within bracket expressions defined by Basic Regular Expressions is replaced by the use of the backslash (\) to quote bracket expression special characters.

\{...\} The constructs \{*m*\}, \{*m*,\}, and \{*m*,*n*\} are not supported.

nomagic When the editor option **nomagic** is set, the only characters with special meanings are ^ at the beginning of a pattern, **$** at the end of a pattern, and \. The characters ., *, **[**, and ~ lose their special meanings, unless escaped by a \.

Replacement Strings

The character **&** (**&** if **nomagic** is set) in the replacement string stands for the text matched by the pattern to be replaced. The character ~ (\~ if **nomagic** is set) is replaced by the replacement part of the previous **substitute** command. The sequence *n*, where *n* is an integer, is replaced by the text matched by the pattern enclosed in the *n*th set of parentheses \(and \). The sequence **u** (**l**) causes the immediately following character in the replacement to be converted to uppercase (lowercase), if this character is a letter. The sequence **U** (**L**) turns such conversion on, until the sequence **E** or **e** is encountered, or the end of the replacement string is reached.

Edit Options

The command **ex** has a number of options that modify its behavior. These options have default settings, which can be changed using the **set** command (see above). Options can also be set at startup by putting a **set** command string in the environment variable EXINIT, or in the file **.exrc** in the HOME directory, or in **.exrc** in the current directory. If EXINIT exists, the **.exrc** file in the HOME directory is not executed. If the current directory is not the HOME directory and the **exrc** editor option (see below) is set, the **.exrc** file in the current directory is executed after EXINIT or the HOME directory **.exrc**.

The editor obtains the horizontal and vertical size of the terminal screen from the *terminfo*(4) database. These values can be overridden by setting the COLUMNS and LINES environment variables. See the **window** option below for more information.

Options are Boolean unless otherwise specified.

autoindent, ai If autoindent is set, each line in insert mode is indented (using blanks and tabs) to align with the previous line. (Indentation begins after the line appended, or before the line inserted or the first line changed.) Additional indentation can be provided as usual; succeeding lines are automatically indented to the new alignment. Reducing the indent is achieved by typing ^**D** one or more times; the cursor is moved back **shiftwidth** spaces for each ^**D**. (A ^ followed by a ^**D** removes all indentation temporarily for the current line; a **0** followed by a ^**D** removes all indentation.)

autoprint, ap The current line is printed after each command that changes buffer text. (Autoprint is suppressed in **global** commands.)

autowrite, aw The buffer is written to the current file if the buffer has been modified and a **next**, **rewind**, or **!** command is given.

beautify, bf Cause all control characters other than tab, new-line and formfeed to be discarded from the input text.

directory, dir Specify the directory to which the editor buffer should be placed. This option only takes effect when a new buffer is created. It should be set in EXINIT or **.exrc** to affect the location of the buffer file for the edit file specified on the command line.

If the specified directory is not writable by the user, the editor quits if set from EXINIT or a **.exrc** file and issues an error message if set interactively by the user.

doubleescape When set, two consecutive ESC (escape) characters are required to leave input mode. In input mode, a single ESC character followed by a different character causes *vi*(1) to issue an audible or visual warning (see the **flash** edit option) and insert both characters into the buffer. This option is not set by default.

The character sequences transmitted by the keyboard editing keys of some terminals are identical to some sequences of *vi*(1) user commands. If the mapping of these keys is enabled (see the **keyboardedit** and **keyboardedit!** options), *vi*(1) might not be able to reliably distinguish between the character sequence transmitted by an editing key and the same character sequence typed by a user. This problem is most likely to occur when the user is typing ESC to terminate input mode immediately followed by another *vi*(1) command. By setting the **doubleescape** option, the ambiguity of this case is removed.

edcompatible, ed Cause the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled by repeating the suffixes.

errorbells, eb When set, error messages are preceded with a bell only on terminals that do not support a standout or highlighting mode such as inverse video. If the terminal supports highlighting, the bell is never used prior to error messages and this option has no effect. Note that visual mode errors are signaled by the bell (regardless of the setting of this option) without an accompanying error message.

exrc When set, the **.exrc** file in the current directory is processed during editor initialization if the current directory is not the HOME directory. This option is not set by default and must be set in the EXINIT environment variable or the HOME directory **.exrc** file to have any effect. See the Edit Options introductory text above.

flash, fl When set, the screen flashes instead of beeping, provided an appropriate **flash_screen** entry is present in the **/usr/lib/terminfo/*** database for the terminal being used.

hardtabs, ht Define the spacing between hardware tab settings and the number of spaces used by the system when expanding tab characters. Tab stops are placed in each column number (starting at the left edge of the screen) that corresponds to an integer multiple of the numeric value of this option.

ignorecase, ic All uppercase characters in the text are mapped to lowercase in regular expression matching. Also, all uppercase characters in regular expressions are mapped to lowercase, except in character class specifications.

keyboardedit When set, the keyboard editing key mappings that are loaded automatically at initialization for command mode use are enabled. If not set, these mappings are disabled (but not deleted). Use the **map** command to get a list of the

currently enabled command mode mappings. This option is set by default.

keyboardedit! When set, the keyboard editing key mappings automatically loaded at initialization for insert mode use are enabled. If not set, these mappings are disabled (but not deleted). Use the **map!** command to list the currently enabled insert mode mappings. This option is not set by default for terminals whose keyboard editing keys send HP-style escape sequences (an ESCAPE followed by a single letter). This option is set by default for all other terminals.

lisp **Autoindent** mode, and the **( )** { } **[[ ]]** commands in visual mode are suitably modified for **lisp** code.

list All printed lines are displayed with tabs shown as ^**I**, and the end of line marked by a **$**.

magic Change the interpretation of characters in regular expressions and substitution replacement strings (see the relevant sections above).

mesg If set, other users can use the *write*(1) command to send messages to your terminal, possibly disrupting the screen display. Unsetting this option blocks write permission to your terminal from other system users while you are using the editor.

modelines, ml If set when the editor reads in a file, any *ex* commands embedded in the first five and last five lines of the file are executed after **.exrc** and EXINIT commands are processed but before editing control is given to the user. The *ex* commands must be prefixed by **ex:** or **vi:** and terminated by **:** in a single line. Any number of other characters with the exception of the colon (:) can precede or follow the embedded command.

novice Use the version of the editor available for novices, known as *edit*(1) or *vedit*(1).

number, nu Cause lines to be printed with line numbers.

optimize, opt Suppress automatic carriage returns on terminals that do not support direct cursor addressing. This streamlines text output in certain situations such as when printing multiple lines that contain leading white space.

paragraphs, para

The value of this option is a string whose successive pairs of characters specify the names of text-processing macros that begin paragraphs. (A macro appears in the text in the form **.XX**, where the **.** is the first character in the line.)

If any macros have a single-character name, use a space character to substitute for the missing second character in the name. When typing a space character in such situations, the space must be preceded by a backslash (\) to prevent the editor from interpreting it as a delimiter.

prompt When set, command mode input is prompted for with a colon (**:**); when unset, no prompt is displayed.

readonly, ro Set the read-only flag for the file being edited, thus preventing accidental overwriting at the end of the session. This option is equivalent to invoking *vi* or *ex* with the **-R** option or using the *view*(1) command.

redraw The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.)

remap If set, macro translation allows for macros defined in terms of other macros; translation continues until the final product is obtained. If unset, a one-step translation only is done.

report The value of this option gives the number of lines that must be changed by a command before a report is generated on the number of lines affected.

scroll The value of this option determines the number of lines scrolled by a ^**D** command and the number of lines displayed by the **z** command (twice the value of scroll).

sections The value of this option is a string, in that successive pairs of characters specify the names of text-processing macros that begin sections. (See **paragraphs** option above.)

shell, sh The value of this option specifies the file name of the user shell to be used for the **!** shell escape and **shell** commands. It is set by default to the value of the user's SHELL environment variable, if set, and otherwise to **/bin/sh**.

shiftwidth, sw The value of this option gives the width of a software tab stop, and is used during **autoindent** and by the shift commands.

showmatch, sm
When a **)** or } is typed while in visual mode, the matching **(** or { is shown if it is still on the screen.

showmode, smd
When set, **showmode** displays the current editor mode (such as "INPUT MODE") in the lower right-hand corner of the screen during the **visual** and **open** commands.

slowopen, slow
In visual mode, **slowopen** prevents screen updates during input to improve throughput on unintelligent terminals.

tabstop, ts The value of this option specifies the software tab stops to be used by the editor to expand tabs in the input file.

taglength, tl Specify the maximum number of characters in a tag that should be treated as significant. Characters beyond the limit are ignored. A value of zero (default value) means that all characters in the tag are significant.

tags Specify which files should be used by the **tag** command and the **-t** option. The default is **tags /usr/lib/tags**. Each line of a tags file contains the following three fields separated by one or more space or tab characters: the tag name, the name of the file to be edited, and an address specification associated with the tag. A **tags** file must be sorted in order by tag name (see EXTERNAL INFLUENCES below).

The command *ctags*(1) can be used to create **tags** files from C, Pascal and Fortran source files.

term Define the type of terminal being used with the editor. The value is obtained from the TERM environment variable by default. There is no difference between the **term** and **ttytype** editor options. Setting either one results in both being changed.

terse When set, error messages are shorter.

timeout, to When set, all the characters of a multicharacter macro name (the first argument to the map command) must be received within the amount of time specified by the **timeoutlen** option to be accepted as a match for the macro and mapped to the defined string. If not set, no limit is placed on how long *vi*(1) waits for the completion of a macro name. This option is set by default.

timeoutlen The value of this option specifies in milliseconds (ms) the length of the macro timeout window (see the **timeout** option). The value must be at least 1 ms and the default is 500 ms. The value of this option has no effect unless the **timeout** option is enabled.

ttytype, tty Define the type of terminal being used with the editor. The value is obtained from the TERM environment variable by default. There is no difference between the **term** and **ttytype** editor options. Setting either one results in both being changed.

warn Warn if there has been "no write since last change" before a **!** or **shell** command escape.

window, wi The number of lines in a text window in visual mode. The default value is set to one less than the size of your terminal screen (as defined by the LINES environment variable, if set, or the *terminfo*(4) data base otherwise). However, if the terminal baud rate (see *stty*(1) is set to less than 1200 or 2400, the default value is reduced to a maximum of 8 or 16 lines, respectively. The default value may be overridden by specifying a window size via the **-w** option when invoking the editor.

w300 If the terminal baud rate is less than 1200, set the **window** editor option to the value specified.

w1200 If the terminal baud rate is greater than or equal to 1200 but less than 2400, set the **window** editor option to the value specified.

w9600 If the terminal baud rate is greater than or equal to 2400, set the **window** editor option to the value specified.

wrapscan, ws When set, editor searches using */re/* (or *?re?*) resume from the beginning (or end) of the file upon reaching the end (or beginning) of the file (that is, the scan "wraps around"). When unset, editor searches stop at the beginning or the end of the file, as appropriate.

wrapmargin, wm
In visual mode, if the value of this option is greater than zero (that is, **wrapmargin**=*n*), a newline is automatically added to an input line at a word boundary, so that lines end at least *n* spaces from the right margin of the terminal screen.

writeany, wa Inhibits the checks otherwise made before write commands, allowing a write to any file (provided the system allows it).

WARNINGS

The **undo** command causes all marks to be lost on lines that are changed and then restored.

The **z** command prints a number of logical rather than physical lines. More than a screenful of output might result if long lines are present.

Null characters are discarded in input files and cannot appear in resultant files.

On some systems, recovery of an edit with the **−r** option might be possible only if certain system-dependent actions were taken when the system was restarted.

On HP terminals, the attribute field of any function key specified via a **map #***n* ... command should be set to **normal** rather than to the default of **transmit**.

AUTHOR

Ex was developed by the University of California, Berkeley. The 16-bit extensions to *ex* are based in part on software of the Toshiba Corporation.

FILES

/usr/lib/terminfo/*/*	describes capabilities of terminals
./.exrc	editor initialization file
$HOME/.exrc	editor initialization file
/usr/lib/exstrings	error messages
/usr/lib/exrecover	recover command
/usr/lib/expreserve	preserve command
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

ctags(1), ed(1), edit(1), stty(1), vi(1), write(1), terminfo(4), environ(5), lang(5), regexp(5).

The vi/ex Editor tutorial in the *Text Editors and Processors* volume of *HP-UX Concepts and Tutorials*.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions and in processing the **tags** file.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as uppercase or lowercase letters, the shifting of the case of letters, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, the editor behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

ex: SVID2, XPG2, XPG3

NAME

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [–tabstop] [–tab1,tab2,...,tabn] [file ...]
unexpand [–**a**] [file ...]

DESCRIPTION

Expand processes the named files or the standard input and writes the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given, tabs are set *tabstop* spaces apart instead of the default **8**. If multiple tabstops are given, the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default only leading blanks and tabs are reconverted to maximal strings of tabs. If the –**a** option is given, tabs are inserted whenever they would compress the resultant file by replacing two or more blanks before a tab stop.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr* returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& *expr* returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison (note that = and == are identical, in that both test for equality).

expr { +, − } *expr*
addition or subtraction of integer-valued arguments.

expr { *, /, % } *expr*
multiplication, division, or remainder of the integer-valued arguments.

expr : *expr* The matching operator : compares the first argument with the second argument which must be a regular expression. *Expr* supports the Basic Regular Expression syntax (see *regexp*(5)), except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the **\(...\)** pattern symbols can be used to return a portion of the first argument.

length *expr* The length of *expr*.

substr *expr expr expr*
Takes the substring of the first *expr*, starting at the character specified by the second *expr* for the length given by the third *expr*.

index *expr expr* Returns the position in the first *expr* which contains a character found in the second *expr*.

match Match is a prefix operator equivalent to the infix operator :.

EXAMPLES

1. **a=`expr $a + 1`**
   Adds 1 to the shell variable **a**.

2. **expr $a : '.*/\(.*\)' \| $a**
   For $a equal to either "/usr/abc/file" or just "file", this example returns the last segment of a path name (i.e., **file**). Watch out for / alone as an argument: *expr* will take it as the division operator (see WARNINGS below).

3. **expr //$a : '.*/\(.*\)'**
   This is a better representation of example 2. The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. **expr $VAR : '.*'**
   Returns the number of characters in **$VAR**.

RETURN VALUE

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

SEE ALSO

sh(1), test(1), environ(5), lang(5), regexp(5).

DIAGNOSTICS

syntax error for operator/operand errors

non-numeric argument if arithmetic is attempted on such a string

WARNINGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

```
expr X$a = X=
```

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions and the behavior of the relational operators when comparing string values.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *expr* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

expr: SVID2, XPG2, XPG3

NAME

f77, fc − FORTRAN 77 compiler

SYNOPSIS

f77 [*options*] *file* ...
fc [*options*] *file* ...

Remarks:

The *fc* command conflicts with the built-in *ksh*(1) *fc* command, and is therefore considered obsolescent, but is retained for backward compatibility. Refer to *ksh*(1) for information about the *fc* command history editor.

DESCRIPTION

F77 and *fc* are names for the same HP-UX FORTRAN 77 compiler. The compiler accepts the following types of *file* arguments (see the DEPENDENCIES section for additional *file* types):

(1) Arguments whose names end with **.f** are taken to be FORTRAN 77 source files. They are compiled, and each object file is left in the current directory in a file whose name is that of the source, with **.o** substituted for **.f**. (The **.o** file will not be created for a single source that is compiled and loaded, nor for any source which fails to compile correctly.)

(2) Arguments whose names end with **.o** are passed on to the linker (*ld*(1)) to be linked into the final program.

Arguments can be passed to the compiler through the **FCOPTS** environment variable as well as on the command line. The compiler picks up the value of **FCOPTS** and places its contents before any arguments on the command line. For example,

```
FCOPTS=-v
export FCOPTS
f77 -L prog.f
```

is equivalent to

```
f77 -v -L prog.f
```

The **TMPDIR** environment variable can be set to specify a directory with temporary files, to be used instead of the default directories **/tmp** and **/usr/tmp**.

Options

The following options are recognized:

−A<*secondary*>

Generate errors for features not part of the ANSI 77 standard. As a side effect, the **−K** option is set if neither secondary option **R** nor **r** is specified.

Secondary is one or more of the letters from the set {**aAhHmMnNrRsS**}, and specifies what exceptions to the standard are allowed. *Secondary* is optional. If it is not specified, no exceptions to the ANSI standard are allowed. If *secondary* is specified, only the specified exceptions are allowed and all other exceptions are recognized with an error. If *secondary* is an uppercase letter, the specified exception is completely ignored. If *secondary* is a lowercase letter, the exception specified generates a warning but not an error.

A *secondary* specification causes the following exceptions to be recognized:

H or **h** Recognize Hollerith data as a non-ANSI exception. **H** produces no warnings; **h** produces warnings.

M or **m** Allow MIL-STD-1753 extensions. **M** produces no warnings; **m** produces warnings.

N or **n** Allow NAMELIST as a non-ANSI exception. **N** produces no warnings; **n** produces warnings.

R or **r** Allow recursion. **R** produces no warnings; **r** produces warnings. As a side effect, DATA statements amid executable statements produce errors.

S or **s** Allow selected syntactical features: lowercase letters, tabs, quotation marks, and symbolic names with a length greater than six characters. **S** produces no warnings; **s** produces warnings.

A or **a** Allow recognition of all the exceptions listed above; all other ANSI extensions are disallowed. **A** produces no warnings for all exceptions allowed; **a** produces warnings.

−a Issue warnings for non-ANSI features (same as **$OPTION ANSI ON** and the **+s** option).

−c Suppress linking and produce object (**.o**) files from source files.

−C Enable range checking (same as **$OPTION RANGE ON**).

−D Compile debug lines as source statements.
Source lines with a "D" or "d" in column 1 are treated as comments by default.

−g Cause the compiler to generate additional information needed by a symbolic debugger. (This option may be incompatible with optimization.)

−G Prepare object files for profiling with *gprof* (see *gprof*(1)).

−I2 Make default size of integers and logicals **INTEGER*2** and **LOGICAL*2** (same as **$OPTION SHORT**).

−I4 Make default size of integers and logicals **INTEGER*4** and **LOGICAL*4**. This is the compiler's default.

−I*dir* Add *dir* to the list of directories that are searched for **$INCLUDE** files whose names do not begin with **/**. Initially, only the directory containing the source file is searched.

−K Automatically SAVE all local variables in all subprograms. This option forces static storage for these variables in order to provide a convenient path for importing FORTRAN 66 and FORTRAN 77 programs which were written to depend on static allocation of memory (that is, variables retaining their values between invocations of the respective program units).

−l*x* Cause the linker to search first in the library named by **/lib/lib***x***.a** and then in **/usr/lib/lib***x***.a** (See *ld*(1)).

−L Write a program listing to the standard output during compilation.

−n Cause the output file from the linker to be marked *shared*.

−N Prevent the output file from the linker from being marked *shared*.

−o *outfile* Name the output file from the linker *outfile* instead of **a.out**.

−onetrip Execute any DO loop at least once.

−O Invoke full optimization.

−p Prepare object files for profiling with *prof* (see *prof*(1)).

−q Cause the output file from the linker to be marked *demand load*.

−Q Prevent the output file from the linker from being marked *demand load.*

+Q *dfile* Cause *dfile* to be read before compilation of each source file. *Dfile* can only contain compiler directives.

−R4 Make the default size of floating-point constants **REAL*4**. This is the compiler's default.

−R8 Make the default size of floating-point constants **REAL*8**. For example, the constant **3.5** would be treated as if it was the constant **3.5D0** if the **-R8** option was specified.

−s Cause the output of the linker to be stripped of symbol table information (see *ld*(1) and *strip*(1)). (This option is incompatible with symbolic debugging.)

−S Compile the named source files and leave the assembly language output in corresponding files whose names are suffixed with **.s** (no **.o** files are created).

+s Issue warnings for non-ANSI features (same as **$OPTION ANSI ON**). The **−a** option should be used instead of **+s**.

−t *c,name* Substitute or insert subprocess *c* with *name* where *c* is one or more implementation-dependent set of identifiers indicating the subprocess(es). Works in two modes:

1. If *c* is a single identifier, *name* represents the full path name of the new subprocess.
2. If *c* is a set of identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path names of the new subprocesses.

One or more values that *c* can assume are:

c Compiler body (standard suffix is *f77comp*)
0 Same as *c*
l Linker (standard suffix is *ld*)

For other values that *c* can assume, see the DEPENDENCIES section below.

−u Force types of identifiers to be implicitly undeclared (same as specifying **IMPLICIT NONE**; no other **IMPLICIT** statements are permitted).

−U Use uppercase for external names (default is lowercase).

−v Enable the verbose mode, producing a step-by-step description of the compilation process on the standard error output.

−w Suppress warning messages (same as **$OPTION WARNINGS OFF**).

−w66 Enable warnings about FORTRAN 66 features used.

−W *c,arg1[,arg2,...,argN]*

Cause *arg1* through *argN* to be handed off to subprocess *c*. The *argi* take the form *-argoption[,argvalue]*, where *argoption* is the name of an option recognized by the subprocess and *argvalue* is a separate argument to *argoption* where necessary. The *c* can assume the values listed under the **−t** option, as well as those of **d** (driver program), which has a special meaning explained below.

The **−W d** option specification allows additional implementation-specific options to be recognized and passed through the compiler driver to the appropriate subprocesses (see **−W** above). For example, on the Series 800:

```
- -W d,-Q,dfile,-s
```

sends the options **−Q** *dfile* and **−s** through the compiler driver. Furthermore, a shorthand notation for this mechanism can be used by placing + in front of the option name as in:

+Q dfile +s

which is equivalent to the previous option expression. Note that for simplicity this shorthand is applied to each implementation-specific option individually, and that the *argvalue* is no longer separated from the *argoption* by a comma (see **−W**).

−y Generate additional information needed by static analysis tools, and ensure that the program is linked as required for static analysis. This option is incompatible with optimization.

−Y Enable 8- and 16-bit Native Language Support (NLS) in strings and comments. In the default case, NLS is not enabled.

+R Write cross reference and symbol table information to standard output during compilation.

DIAGNOSTICS

The diagnostics produced by *f77* are intended to be self-explanatory. If a listing is requested (**−L** option), errors are written to the listing file. If no listing is being generated, errors are written to standard error output.

Series 800: Errors are written to both the listing file and the standard error output if the **−L** option is specified and if standard output and standard error output are not directed to the same place.

DEPENDENCIES

Series 300

The **−y** option is not supported on the Series 300.

Arguments whose names end with **.c** or **.s** are assumed to be C or assembly source programs, and are compiled or assembled, producing **.o** files.

Arguments whose names end with **.r** are assumed to be *ratfor*(1) source programs. These are first transformed by the *ratfor* preprocessor, then compiled by *f77* producing **.o** files.

The **−t** and **−W** options allow *c* to assume the following values:

r	*Ratfor* preprocessor (standard suffix is *ratfor*)
c	Compiler front end (standard suffix is *f77pass1*)
0	Same as *c*
p	Procedure integrator (standard suffix is *c0*)
1	Code generator (standard suffix is *f1*)
2	Peephole optimizer (standard suffix is *c2*)
g	Global optimizer (standard suffix is *c1*)
a	Assembler (standard suffix is *as*)
l	Linker (standard suffix is *ld*)

Specifying **−W1, −l** causes source file line numbers to be printed as assembly code comments for debugging purposes.

The **−K** option has two side effects:

(a) All non-initialized variables are initialized to zero, and

(b) The DATA statement can appear among executable statements.

The following implementation-specific options are supported on the Series 300:

+A Cause the compiler to align data using alignment rules where non-character items 4 bytes and larger are aligned on 2-byte boundaries instead of 4-byte boundaries.

+A< *secondary* >
Set the alignment of data items within FORTRAN **STRUCTURE**s. Defined values for *secondary* are:

3 Set the alignment of data items within FORTRAN **STRUCTURE**s to be the same as the alignment produced by the Series 300 C compiler. This is the same as the **$HP9000_300 ALIGNMENT ON** option. This is the compiler default.

8 Set the alignment of data items within FORTRAN **STRUCTURE**s to be the same as the alignment produced by the Series 800 C compiler. This is the same as the **$HP9000_800 ALIGNMENT ON** option.

N Set the alignment of data items within FORTRAN **STRUCTURE**s to be the same as the alignment produced by an industry standard FORTRAN compiler. This is the same as the **$NOSTANDARD ALIGNMENT ON** option.

+bfpa Cause the compiler to generate code that uses the HP 98248A floating-point accelerator card, if it is installed at run-time. If the card is not installed, floating-point operations will be done on the MC68881 math coprocessor.

+B Cause the compiler to treat the backslash character (\) as a C-like escape character.

+e Enable "other vendor" compatibility mode for importing non-HP code. This is equivalent to specifying all the **+E** options and the **-K** option. Some register allocation is turned off and all local variables in subprograms have static storage.

+E< *secondary* >
This option enables specific "other vendor" compatibility modes. Defined values for *secondary* are:

0 Enable extended-range DO loops and jump into IF blocks. This option disables some register allocation.

1 Cause the linker to search the compatibility library **/usr/lib/libFext.a** in addition to the regular FORTRAN libraries.

2 Enable a non-standard interpretation of logical data. The value of a logical is determined by the value of the lowest order bit, rather than by a zero or non-zero value. The logical value of true is represented as -1, rather than +1. This option is the same as the inline **$NOSTANDARD LOGICAL** compiler option.

3 Causes the compiler to pass character item parameters with the length at the end of the parameter list.

+ffpa Cause the compiler to generate code for the HP 98248A floating-point accelerator card. This code does not run unless the card is installed.

+M Prevent the compiler from generating in-line code for the MC68881 math coprocessor. Library routines will be referenced for *matherr* capability.

+N<*secondary*><*n*>

This option lets the user set the initial size of internal compiler tables. However, these tables are expanded automatically by the compiler as needed. *Secondary* is one of the letters from the set {**acenqstx**}, and *n* is an integer value. *Secondary* and *n* are not optional. The table sizes can be re-specified using one of the secondary letters and the number *n* as follows:

- **a** Initial size of external label name storage table (default is 10000 bytes).
- **c** Initial size of control statements table (default is 20 table entries).
- **e** Initial number of expression tree nodes (default is 1000 entries).
- **n** Initial size of the hash table of symbols (default is 401 table entries).
- **q** Initial size of equivalence table (default is 150 table entries).
- **s** Initial size of statement label table (default is 201 table entries).
- **t** Initial size of external symbol storage table (default is 40000 bytes).
- **x** Initial size of external symbol table (default is 200 table entries).

+O<*secondary*>

Cause specific optimizations to be performed. Defined values for *secondary* are:

- **1** Invoke the peephole assembly code optimizer only.
- **2** Invoke the global optimizer and the peephole optimizer. This is the same as the **−O** option.
- **3** Invoke the procedure integrator along with the global optimizer and the peephole optimizer.

+P<*secondary*>

Cause specific optimizations to be performed. Defined values for *secondary* are:

- **1** Invoke the assembly code optimizer (equivalent to **+O1** option).
- **2** Invoke the procedure integrator.

+U Distinguish uppercase and lowercase (case is significant). Keywords are only recognized in lowercase.

Series 800

The **−A**, **−w66**, **−R4**, and **−R8** options are not currently supported.

The **TMPDIR** environment variable is not supported.

The **−Y** option additionally enables NLS support of lexical comparisons of 8- and 16-bit data. This option also causes /usr/lib/libportnls.a to be linked in between libcl.a and libc.a. If the **−Y** option is not specified, the library /usr/lib/libnlsstubs.a is linked in between libcl.a and libc.a.

The **−l** option, when used with the **isam** library, will cause **/usr/lib/libmap.a**, **/usr/lib/libfh.a**, and **/usr/lib/libfhsup.a** to be linked in between **libcl.a** and **libc.a**. If the **−lisam** option is not given, the library **/usr/lib/libisamstubs.a** is linked in between **libcl.a** and **libc.a**.

The following implementation-specific options are supported on the Series 800:

+A Cause the compiler to align data using alignment rules where non-character items 4 bytes and larger are aligned on 2-byte boundaries instead of 4-byte boundaries. This is the same as **$HP1000**

+A< *secondary* >
Set the alignment of data items within FORTRAN **STRUCTURE**s. Defined values for *secondary* are:

3 Set the alignment of data items within FORTRAN **STRUCTURE**s. to be the same as the alignment produced by the Series 300 C compiler. This is the same as the **$HP9000_300 ALIGNMENT ON** option.

N This option is not currently supported.

+e Enable "other vendor" compatibility mode. This is equivalent to specifying the **$NOSTANDARD** compiler directive at the beginning of the program file.

+E< *secondary* >
This option enables specific "other vendor" compatibility modes. Defined values for *secondary* are:

0 This is provided for compatibility reasons only. Extended-range DO loops and jumps into IF blocks are enabled by default on Series 800 systems.

1 This is the same as specifying the **$NOSTANDARD SYSTEM** compiler directive. It tells the compiler to treat calls to the routines DATE, IDATE, SECNDS, TIME, RAN and EXIT as calls to intrinsics.

2 This is the same as specifying the **$NOSTANDARD LOGICALS** compiler directive. It tells the compiler to treat -1 as .TRUE. and zero as .FALSE.

3 This is the same as specifying the **$NOSTANDARD CHARS** compiler directive. It tells the compiler to pass character item parameters with the length at the end of the parameter list.

4 This is the same as specifying the **$NOSTANDARD IO** compiler directive. It allows the output of numeric data types (INTEGER, REAL, DOUBLE and QUAD) with alpha format specifiers (A and R).−

+O< *opt* >
Invoke optimizations selected by *opt*. If *opt* is **1**, only level 1 optimizations are performed, If *opt* is **2**, all optimizations are performed. The option **−O** is the same as **+O2**.

+T Cause the running program to issue a procedure traceback for run-time errors.

AUTHOR

F77 and *fc* were developed by HP.

FILES

a.out	linked executable output file
file.f	input file (FORTRAN source file)
file.o	object file
/lib/libc.a	C library; see Section 3 of this manual and *intro*(3).
/lib/libm.a	math library
/tmp/fc*	compiler temporary files

Series 300

/bin/as20	assembler
/lib/c0	procedure integrator
/lib/c1	global optimizer
/lib/c2	assembly code optimizer
/usr/lib/end.o	symbolic debugger string buffer
/lib/f1	compiler code generator
/usr/lib/f77pass1	compiler front end
file.c	input file (C source file)
file.r	input file (*ratfor* source file)
file.s	input file (assembly source file)
/lib/frt0.o	run-time startoff routine
/lib/libp/libc.a	C library (profiling)
/usr/lib/libFext.a	compatibility function library
/lib/libp/libFext.a	compatibility function library (profiling)
/usr/lib/libF77.a	intrinsic function library
/lib/libp/libF77.a	intrinsic function library (profiling)
/usr/lib/libI77.a	FORTRAN I/O library
/lib/libp/libI77.a	FORTRAN I/O library (profiling)
/lib/libp/libm.a	math library (profiling)
/lib/gfrt0.o	startoff with profiling via *gprof*
/lib/mfrt0.o	startoff with profiling via *prof*
/usr/tmp/fxr*	temporary files for cross referencing

Series 800

/usr/lib/f77comp	compiler
/lib/crt0.o	runtime startup
/usr/lib/libcl.a	FORTRAN math and I/O libraries
/usr/lib/libportnls.a	NLS run-time library
/usr/lib/libnlsstubs.a	stubs for NLS library routines
/usr/lib/libmap.a	ISAM run-time support library
/usr/lib/libfh.a	ISAM run-time support library
/usr/lib/libfhsup.a	ISAM run-time support library
/usr/lib/libisamstubs.a	stubs for ISAM library routines
/usr/lib/nls/$LANG/f77_msgs.cat	error message catalog

SEE ALSO

as(1), asa(1), cc(1), gprof(1), ld(1), prof(1), strip(1), matherr(3M).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported in comments and strings.

NAME

factor, primes – factor a number, generate large primes

SYNOPSIS

factor [*number*]

primes [*start* [*stop*]]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number, it factors the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to sqrt(*n*) and occurs when *n* is prime or the square of a prime.

The largest number that can be dealt with by *factor* is 1.0e14.

Primes prints prime numbers between a lower and upper bound. If *primes* is invoked without any arguments, it waits for two numbers to be typed in. The first number is interpreted as the lower bound, and the second as the upper bound. All prime numbers in the resulting inclusive range are printed.

If *start* is specified, all primes greater than or equal to *start* are printed. If both *start* and *stop* are given, then all primes occurring in the inclusive range "*start – stop*" are printed.

Start and *stop* values must be integers represented as long integers.

If the stop value is omitted in either case, *primes* runs until either overflow occurs or it is stopped by typing *interrupt*.

The largest number that can be dealt with by *primes* is 2,147,483,647.

DIAGNOSTICS

"Ouch" when the input is out of range, for garbage input, or when *start* is greater than *stop*.

EXAMPLES

The command:

factor 12

prints the prime factorization for the number **12**.

The command:

primes 0 20

prints all prime numbers between **0** and **20**.

NAME

file – determine file type

SYNOPSIS

file [**–m** mfile] [**–c**] [**–f** ffile] arg ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable *a.out* file, *file* will print the version stamp, provided it is greater than 0 (see the description of the **–V** option in *ld*(1)).

File uses the file */etc/magic* to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of */etc/magic* explains its format.

The **–m** option instructs *file* to use an alternate magic file.

The **–c** flag causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file classification is done under **–c**.

–f*ffile* specifies that *ffile* is a file containing a list of the files which are to be examined. *File* then classifies each file whose name appears in *ffile*.

SEE ALSO

ld(1).

STANDARDS CONFORMANCE

file: SVID2, XPG2

NAME

find − find files

SYNOPSIS

find *path-name-list* [*expression*]

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, one or more path names) seeking files that match a Boolean *expression* written in the primaries given below. By default, *find* does not follow symbolic links.

The Boolean expression is evaluated using short-circuit evaluation. This means that whenever the result of a Boolean operation (*AND* or *OR*) is known from evaluating the left-hand argument, the right-hand argument is not evaluated.

In the descriptions of the primaries, the argument n is used as a decimal integer; $+n$ means more than n, $-n$ means less than n, and n means exactly n.

The following primaries are recognized:

−depth A position-independent term which causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission. It is also useful when you are using *cpio*(1) and you wish the modification dates of directories to be preserved. Always true.

−follow A position-independent term which causes *find* to follow symbolic links. Always true.

−hidden A position-independent term which causes *find* to include hidden subdirectories (context-dependent files) in the directory hierarchy of each path name in the *path-name-list*. In addition, the normally hidden components of the path name become visible as as when using the **−print** primary. See *cdf*(4). Always true.

−fsonly *type* A position-independent term which causes *find* to stop descending any directory whose file system is not of the type specified by *type*, where *type* is one of **nfs**, **cdfs**, or **hfs**. In this context, mount points inherit the fstype of their parent directory. This means that when **−fsonly hfs** has been specified and *find* encounters an nfs mount point that is mounted on an hfs file system, the mount point will be visited but entries below that mount point will not. It is important to note that when **−fsonly nfs** has been specified, any **hfs** file systems that are beneath the mount point of an **nfs** file system are not traversed. Always true.

−mountstop A position-independent term that causes *find* to avoid crossing any file system mount points that exist below starting points enumerated in *path-name-list*. The mount point itself is visited, but entries below the mount point are not. Always true.

−name *file* True if *file* matches the last component of the current file name. The matching is performed as specified by Pattern Matching Notation (see *regexp*(5)).

−path *file* Like **−name** except the full path (as would be output by **−print**) is used instead of just the basename. Note that '/' characters are not treated as a special case. For example, '***/.profile**' would match '**./users/fred/.profile**'.

−perm *onum* If the minus sign is omitted, the primary is true provided the file permission bits exactly match the octal number *onum* and only the bits corresponding to

the octal mask 0777 are compared (see description of the octal *mode* in *chmod*(1)). Otherwise, if *onum* is prefixed by a minus sign, more flag bits (corresponding to the octal mask 017777) are compared, and the primary is true if all of the bits that are set in *onum* are set in the *st_mode* value of the file.

−fstype *type* True if the file system to which the file belongs is of type *type*, where *type* is one of **hfs, cdfs,** or **nfs**.

−type *c* True if the type of the file is *c*, where *c* is:

f	regular file
d	directory
b	block special file
c	character special file
p	FIFO (named pipe)
l	symbolic link
s	socket
n	network special file
M	mount point
H	hidden directory (see *cdf*(4))

The use of **−type** *H* implies the **−hidden** primary (see above).

−links *n* True if the file has *n* links.

−user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

−group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

−devcid *cname* True if the file is a block or character special file whose *st_rcnode* value is equal to the cnode id of the cnode *cname*. If *cname* is numeric and does not appear as a cnode name in the **/etc/clusterconf** file, it is taken as a cnode ID. See *mknod*(2).

−size *n*[**c**] True if the file is *n* blocks long. If *n* is followed by a **c**, the size is in characters.

−atime *n* True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.

−mtime *n* True if the file has been modified in *n* days.

−ctime *n* True if the file inode has been changed in *n* days.

−newer *file* True if the current file has been modified more recently than the argument *file*.

−inum *n* True if the file has inode number *n*.

−print Causes the current path name to be printed. Always true.

−exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by a semicolon (semicolon is special to the shell and must be escaped). Any command argument {} is replaced by the current path name.

−ok *cmd* Like **−exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

−cpio *device* Write the current file on *device* in *cpio*(4) format (5120-byte records). The use of **−cpio** implies **−depth**. Always true.

−ncpio Same as **−cpio** but adds the −c option to **cpio**. The use of **−ncpio** implies **−depth**. Always true.

−prune If the current entry is a directory, cause *find* to skip that directory. This can be useful to avoid walking certain directories, or to avoid recursive loops when using cpio −p. Note, however, that **−prune** is useless if the **−depth** option has also been given. See the description of **−only** and the EXAMPLES section, below, for more information. Always true.

−only This is a positive-logic version of **−prune.** A **−prune** is performed after every directory, unless **−only** is successfully evaluated for that directory. As an example, the following three commands are equivalent:

find . -fsonly hfs -print
find . -print -fstype hfs -only
find . -print ! -fstype hfs -prune

Note, however, that **−only** is useless if the **−depth** option has also been given. Always true.

(*expression* **)** True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

Primaries can be combined by using the following operators (in order of decreasing precedence):

! *expression* Logical NOT operator. True if the *expression* is not true.

expression expression
Logical AND operator. True if both of the *expressions* are true.

expression **−o** *expression*
Logical OR operator. True if either or both of the *expressions* are true.

If *expression* is omitted, or if none of **−print, −ok, −exec, −cpio,** or **−ncpio** are specified, **−print** is assumed.

Access Control Lists

The following primary enables the user to search for access control list entries:

−acl *aclpatt* True if the file's access control list matches an access control list pattern or contains optional access control list entries (see *acl*(5)). The **−acl** primary has three forms:

−acl *aclpatt* Match all files whose access control list includes all (zero or more) pattern entries specified by the *aclpatt* pattern.

−acl =*aclpatt* Match a file only if its access control list includes all (zero or more) pattern entries specified by the *aclpatt* pattern, and every entry in its access control list is matched by at least one pattern entry specified in the *aclpatt* pattern.

−acl opt Match all files containing optional access control list entries.

The *aclpatt* string of the **−acl** primary can be given as an operator or short form pattern; see *acl*(5).

By default, **−acl** is true for files whose access control lists include all the (zero or more) access control list patterns in *aclpatt*. A file's access control list can also contain unmatched entries.

If *aclpatt* begins with =, the remainder of the string must match all entries in a file's access control list.

The *aclpatt* string (by default, or the part following =) can be either an access control list or an access control list pattern. However, if it is an access control list, *aclpatt* must include at least the three base entries ((*u*.%, mode), (%.*g*, mode), and (%.%, mode)).

As a special case, if *aclpatt* is the word **opt**, the primary is true for files with access control list entries.

EXAMPLES

The following example searches two directories (**/example** and **/new/example**) for files containing the string **Where are you** and prints the names of the files:

find /example /new/example -exec grep −l 'Where are you' {} \;

To remove all files named **a.out** or ∗**.o** that have not been accessed for a week:

find / \(−name a.out −o −name '∗.o' \) −atime +7 −exec rm {} \;

Note that the spaces delimiting the escaped parentheses are required.

Print the names of all files in **/bin** that are context dependent, that is, hidden directories:

find /bin -type H -print

Print the names of all files on this machine. Avoid walking **nfs** directories while still printing the **nfs** mount points:

find / -fsonly hfs -print

Copy the entire file system to a disk mounted on '/Disk', avoiding the recursive copy problem. Both commands are equivalent (note the use of −**path** instead of −**name**):

cd /; find . ! -path ./Disk -only -print | cpio -pdm /Disk

cd /; find . -path ./Disk -prune -o -print | cpio -pdm /Disk

Copy the root disk to a disk mounted on '/Disk', skipping all mounted file systems below '/'. Note that −**mountstop** does not cause '/' to be skipped, even though it is a mount point. This is because '/' is the starting point and −**mountstop** only affects entries **below** starting points.

cd /; find . -mountstop -print | cpio -pdm /Disk

Access Control Lists

The following finds all files not owned by user "karl" that have access control lists with at least one entry associated with "karl", and one entry for no specific user in group "bin" with the read bit on and the write bit off:

find / ! −user karl −acl 'karl.∗, %.bin+r−w' −print

The following finds all files that have a read bit set in any access control list entry:

find / -acl '∗.∗+r' -print

The following finds all files that have the write bit unset and execute bit set in every access control list entry:

find / −acl '=∗.∗−w+x' −print

The following finds all files that have optional access control list entries:

find / −acl opt −print

DEPENDENCIES

RFA and NFS The −**acl** primary is always false for RFA and NFS files.

AUTHOR

Find was developed by AT&T and HP.

FILES

/etc/clusterconf	cnode names
/etc/group	group names
/etc/passwd	user names
/etc/mnttab	mount points

SEE ALSO

chacl(1), cpio(1), sh(1), test(1), mknod(2), stat(2), cdf(4), cpio(4), fs(4), acl(5), environ(5), lang(5), regexp(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name matching.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in pattern matching notation.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *find* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

find: SVID2, XPG2, XPG3

NAME

findmsg, dumpmsg – create message catalog file for modification

SYNOPSIS

findmsg [**−a**] *file* ...
dumpmsg *file* ...

DESCRIPTION

Findmsg extracts messages from a C program source *file* and writes them to the standard output in a format suitable for input to *gencat*(1). If multiple-input files are specified and the **-a** option is not used, the files are processed sequentially with message-catalog comment lines identifying the input *file* written before the output for each input *file*. *Gencat*(1) will not accept the resulting output if any two source files contain messages belonging to the same set number. The **-a** option causes *findmsg* to merge identically numbered sets from multiple input files to enable *gencat*(1) to process the *findmsg* output.

Findmsg scans the source files for uncommented lines with one of the following three formats embedded within it:

catgets(*any_var*,**NL_SETN**,*1*, "*message*")

"*message*" /* **catgets** *1* */

/* **catgets** *1* */ "*message*"

or any combination of these formats wholly contained on a single physical line. Any number of spaces or tabs may separate the *catgets* comment from the *message*. The digit *1*, which may be any valid message number (see *gencat*(1)), is combined with the *message* string to produce a message-catalog source line. The message source line is assigned to the set whose number is the current value of **NL_SETN** as set by the last **#define** directive encountered. If **NL_SETN** has not yet been defined when a message line is found, the message is output without a set number specification. If more than one message is found belonging to the same set and message number, the last message found is output; other(s) are silently discarded. Conditional compilation and **#include** instructions in the C source files are ignored.

Dumpmsg extracts messages from a message catalog *file* created by *gencat*(1). The messages are written to standard output in a format suitable for editing and re-input to *gencat*(1).

WARNINGS

For historical reasons, *findmsg* also recognizes the formats:

nl_msg(*1*,"*message*")

"*message*" /* **nl_msg** *1* */

/* **nl_msg** *1* */ "*message*"

The use of these formats is not recommended.

AUTHOR

Findmsg was developed by the Hewlett-Packard Company.

SEE ALSO

findstr(1), gencat(1), insertmsg(1), catgets(3C).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of messages as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to

the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *findmsg* and *dumpmsg* behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

findstr – find strings for inclusion in message catalogs

SYNOPSIS

findstr *file ...*

DESCRIPTION

Findstr examines files of C source code for uncommented string constants, which it places along with the surrounding quotes on the standard output, preceding each by the file name, start position, and length. This information will be used by *insertmsg*(1). *Findstr* does not output strings that are parameters of the *catgets*(3C) routine.

WARNINGS

Findstr outputs initialization strings of static string variables. Calling *insertmsg*(1) with these strings causes their replacement with a call to *catgets*(3C). Since the initializer must be a string, this assignment results in an invalid C declaration. For example, the following line:

static char *x[] = "*message*"

is modified by *insertmsg*(1) to:

static char *x[] = (catgets(nlmsg_fd,NL_SETN,1,"*message*"))

These strings should be manually removed from *findstr* output before being input to *insertmsg*(1).

SEE ALSO

insertmsg(1)

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of comments and string literals as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *findstr* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

finger – user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default *finger* lists the login name, full name, terminal write status (if write permission is denied), idle time, login time, the user's home directory and login shell, any plan the person has placed in the file **.plan** in their home directory, and the project on which they are working from the file **.project** also in the home directory, and office location and phone number (if they are known) for each current HP-UX user.

Idle time is minutes if it is a single integer, hours and minutes if a ":" is present, or days and hours if a "d" is present. Account names as well as first and last names of users are accepted.

Options

–m	Match arguments only on user name.
–l	Force long output format.
–p	Suppress printing of the **.plan** files
–s	Force short output format.

WARNINGS

Only the first line of the **.project** file is printed.

AUTHOR

Finger was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/etc/utmp	who file
/etc/wtmp	last login file
/etc/passwd	for users names, offices, ...
~/.plan	plans
~/.project	projects

SEE ALSO

who(1).

NAME

fixman – fix manual pages for faster viewing with man(1)

SYNOPSIS

fixman

DESCRIPTION

This shell script processes all ordinary files under /usr/man/cat* to unexpand all possible spaces to tabs and remove all {character, backspace} pairs. Such pairs usually exist to cause overstriking or underscoring for printer output. They only slow down *man*(1), and use up significant amounts of disk space. The script should be run after running *catman*(1M) to rebuild all cat-able manual entries from pre-nroff forms.

The script does not remove duplicate blank lines, so all files remain a multiple of one page (66 lines) long and can still be passed directly to *lp*(1). (Note that *man*(1) normally uses *rmnl*(1) to accomplish this removal.)

To ensure success, the script should be run by the super-user. It can take two to three hours to complete. As a side-effect, file ownerships and permissions may be changed.

FILES

/usr/man/cat*	Directories containing post-nroff versions of manual entries.

AUTHOR

Fixman was developed by the Hewlett-Packard Company.

SEE ALSO

catman(1M), chmod(1), expand(1), lp(1), man(1), mv(1), rmnl(1), sed(1).

NAME

flint – FORTRAN inter-procedural checker

SYNOPSIS

flint [**-p**] [**-u**] [**-V**] [**-l***x*] [**-x** *exclusions*] [**-T**] [**-f** *flistfile*] [**-c**] [**-o** *lib*] [*f77 options*] *file ...*

DESCRIPTION

Flint detects erroneous, non-portable, or wasteful use of the FORTRAN language. It pays particular attention to mismatches between the calls and the definitions of FUNCTIONs and SUBROUTINEs, COMMON block differences, and the use of uninitialized variables.

The following are problems that *flint* detects:

- Unused variables (declared but never referenced).
- Variables uninitialized (undefined) at the time of reference.
- Unreferenced labels.
- Mismatches in the number and types of formal and actual arguments to FUNCTIONs and SUBROUTINEs.
- Mismatches in the expected and actual return value of FUNCTIONs.
- CALLs of FUNCTIONs, and the use of SUBROUTINEs in expressions.
- COMMON block layout differences

Flint can also generate the following information:

- Call graph (which procedures call which procedures).
- List of procedures referenced but not defined in the source made available to *flint*.

Arguments whose names end with **.f** are interpreted as FORTRAN source files. Arguments whose names end with **.lf** are interpreted as the result of an earlier invocation of *flint* with either the **-c** or **-o** option used. In addition, the **-f** *flistfile* option can be used to name a file containing a list of file names (one per line) to check. *Flint* produces a warning if a file with any other suffix is specified.

Flint takes all **.f**, **.lf**, and **fllib-l***x***.lf** files (specified by **-l***x*) and processes them in their command line order. By default, *flint* appends the standard FORTRAN lint library (**fllib-lc.lf**) to the end of the file list. If the **-c** option is not used, the second pass of *flint* checks this list of files for mutual compatibility. When the **-c** option is used, the **.lf** and **fllib-l***x***.lf** files are ignored.

Any number of *flint* options can be used, in any order, intermixed with file name arguments. The following options are used to change the type of output produced by *flint*:

Options

-x *exclusions* Prevent pass 2 from reporting the listed errors and warnings. *Flint* assigns an error number to each error and warning that it generates. To suppress the reporting of an error or warning, list its error number in a **-x** option. For example,

flint -x 1010,5001 file.f

does not report error number 1010 or warning number 5001 during interprocedural checking. Note that the final statistics about the number of errors and warnings found are reported as if the **-x** option were unspecified.

-T Prevent pass 2 from displaying the text of errors and warnings. *Flint* usually displays an error indication on one line followed by a textual explanation of the error message on the next line. The **-T** option suppresses the display of the textual message.

-p Print a call graph of the program. The call graph consists of a list of those procedures called by each procedure and a list of those procedures which call that procedure.

-u Suppress complaints about FUNCTIONs and SUBROUTINEs that are defined but not used, or used but not defined. This option is useful when running *flint* on a subset of files of a larger program.

-V Suppress complaints about uninitialized variables. Detection of uninitialized variables takes considerable time and space. This option is useful for 'quick checks' of FORTRAN programs.

The following arguments alter *flint*'s behavior:

-l*x* Add a predefined *flint* library **fllib-l***x***.lf**. For example, you can include a version of the *flint* math library **fllib-lm.lf** by inserting **-lm** on the command line. This argument does not suppress the default use of **fllib-lc.lf**.

-f *flistfile* Check the files listed in the file named *flistfile*. The file names in *flistfile* must be specified one per line, and can be either **.f** or **.lf** files.

-c Cause *flint* to produce a **.lf** file for every **.f** file on the command line. These **.lf** files are the product of *flint*'s first pass only, and are not checked for inter-procedural compatibility or uninitialized variables.

-o *lib* Cause *flint* to produce a *flint* library named **fllib-l***lib***.lf**. The **-c** option nullifies any use of the **-o** option. The *flint* library produced is the input that would be given to *flint*'s second pass. The **-o** option simply causes this file to be saved in the named *flint* library. Note that, in order to reduce the size of the *flint* libraries, the data required for uninitialized variable detection is not written if the **-o** option is given.

The **-a**, **-A**, **-C**, **-D**, **-g**, **-G**, **-I**, **-K**, **-L**, **-n**, **-N**, **-O**, **-q**, **-Q**, **-R**, **-s**, **-S**, **-v**, **-w**, **-W**, and all *+opt* options of *f77*(1) are also recognized as separate arguments; however, its **-C**, **-g**, **-G**, **-K**, **-n**, **-N**, **-O**, **+O**, **-q**, **-Q**, and **-s** options are ignored. This makes *flint*'s option list similar to that of the *f77*(1) command, facilitating its use in makefiles instead of the *f77* command.

Flint is a two-pass process. The first pass operates on a per-source-file basis. Syntax and other compiler errors are produced at this point. If all source files are processed without error and the **-c** and **-o** options are not specified, *flint* starts the second pass. It prints the message "Beginning inter-procedural checking" and performs the desired inter-procedural checks. The behavior of the **-c** and **-o** options allows for incremental use of *flint* on a set of FORTRAN source files.

Generally, *flint* is invoked once for each source file with the **-c** option. Each invocation produces a **.lf** file corresponding to the **.f** file, and prints diagnostics for that source file. After all source files have been run separately through *flint*, one should invoke *flint* again without the **-c** option, listing all of the **.lf** files. This will detect all of the inter-procedural inconsistencies. This scheme works well with *make*(1); it allows *make* to use *flint* on only those source files that have been modified since the set of source files were last run through *flint*.

WARNINGS

The algorithm that *flint* uses to detect uninitialized variables does not recognize EQUIVALENCE statements, nor can it follow assigned GOTO statements; therefore, *flint* might occasionally make the incorrect claim that a variable is uninitialized. *Flint* also does not attempt to decipher variable FORMAT expressions; uninitialized variables used in variable FORMAT expressions are not detected.

DEPENDENCIES

Currently, *flint* utilizes a special version of the Series 300 FORTRAN compiler front end; therefore, it only runs on the Series 300.

FILES

/usr/lib/fllib-l*x*.lf	*Flint* libraries that may be specified by the **-l***x* option
/usr/lib/flint1	First pass of *flint*
/usr/lib/flint2	Actual inter-procedural checking program
/usr/tmp/*lint*	Temporary files

SEE ALSO

f77(1), lint(1), make(1).

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [–width] [file ...]

DESCRIPTION

Fold is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1) before coming to *fold*.

SEE ALSO

expand(1)

BUGS

If underlining is present it may be incorrectly altered by folding.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *fold* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

forder − convert file data order

SYNOPSIS

forder [−**a**] [−**l**] [−**n**] [*file* ...]

DESCRIPTION

The text orientation (mode) of a file can be right-to-left (non-Latin) or left-to-right (Latin). This text orientation can affect the way data is arranged in the file. The data arrangements that result are called screen order and keyboard order. *Forder* converts the order of characters in the file from screen order to keyboard order or vice versa.

Forder reads the concatenation of input files (or standard input if none are given) and produces on standard output a converted version of its input. If "-" appears as an input file name, *forder* reads standard input at that point. You can use "--" to delimit the end of options.

Forder converts input files for all languages that are read from right-to-left. Unless the −**a** option is used, the command merely copies input files to standard output for languages that are read from left-to-right.

Options

−**a** Convert file data order for languages read from left-to-right.

−**l** Identify the file as having been created in Latin mode.

−**n** Identify the file as having been created in non-Latin mode.

EXAMPLES

The following command begins with *file1*, which exists in screen order, converts it to keyboard order, sorts the keyboard-ordered output, converts it back to screen order, and redirects the output to *file2*. Note that −**n** is given to inform *forder* that *file1* was created in non-Latin mode.

forder −n file1 | sort | forder −n > file2

WARNINGS

It is the user's responsibility to ensure that the **LANGOPTS** environment variable accurately reflects the status of the file.

If present, alternative numbers always have a left-to-right orientation.

AUTHOR

Forder was developed by HP.

SEE ALSO

environ(5), hpnls(5), strord(3C), nljust(1).

EXTERNAL INFLUENCES

Environment Variables

The LANGOPTS environment variable determines the mode and order of the file. The syntax of LANGOPTS is:

[*mode*] [*_order*]

where *mode* describes the mode of a file: **l** represents Latin mode, and **n** represents non-Latin mode. Non-Latin mode is assumed for values other than **l** and **n**. The *order* describes the data order of a file: **k** is keyboard, and **s** is screen. Keyboard order is assumed for values other than **k** and **s**. Mode information in LANGOPTS can be overridden from the command line.

The LC_ALL environment variable determines the direction of a language (left-to-right or right-to-left).

The LC_NUMERIC environment variable determines whether a language has alternative numbers.

The LANG environment variable determines the language in which messages are displayed.

International Code Set Support
Single-byte character code sets are supported.

NAME

from – who is my mail from?

SYNOPSIS

from [**–s** *sender*] [*user*]

DESCRIPTION

From prints out the mail header lines in your mailbox file to show who sent you mail. If *user* is specified, *user*'s mailbox is examined instead of your own. If the **–s** option is given, only headers of mail from *sender* are printed.

EXAMPLES

The following command lists the header lines of all mail from the sender **ken** currently in your mailbox:

from -s ken

FILES

/usr/mail/*

AUTHOR

From was developed by the University of California, Berkeley.

SEE ALSO

biff(1), mail(1), prmail(1).

NAME

fsplit – split f77, ratfor, or efl files

SYNOPSIS

fsplit options files

DESCRIPTION

Fsplit splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata, function, main, program,* and *subroutine* program segments. Procedure X is put in file X.**f,** X.**r**, or X.**e** depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN*.**[efr]** and unnamed *blockdata* segments in the files *blockdataN*.**[efr]** where *N* is a unique integer value for each file.

The following *options* pertain:

–f (default) Input files are *f77*.

–r Input files are *ratfor*.

–e Input files are *Efl*.

–s Strip *f77* input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), efl(1), f77(1), ratfor(1), split(1).

NAME

ftio – faster tape I/O

SYNOPSIS

ftio **−o** | **−O** [**achpvxAEHLM**] [**−B** *blksize*] [**−D** *type*] [**−K** *comment*] [**−L** *filelist*] [**−N** *datefile*] [**−S** *script*] [**−T** *tty*] [**−Z** *nobufs*] *tapedev* [*pathnames*] [**−F** *ignorenames*]

ftio **−i** | **−I** [**cdfmptuvxAEMPR**] [**−B** *blksize*] [**−S** *script*] [**−T** *tty*] [**−Z** *nobufs*] *tapedev* [*patterns*]

ftio **−g** [**v**] *tapedev* [*patterns*]

DESCRIPTION

Ftio is a tool designed specifically for copying files to 9-track magnetic tape drives. It should perform faster than either *cpio*(1) or *tar*(1) in comparable situations. *Ftio* uses multiple processes (to read/write the file system and to write/read the tape device) with large amounts of shared memory between the processes, as well as a large block size for reading and writing to the tape.

Ftio is compatible with *cpio* in that output from *cpio*(1) is always readable by *ftio*, and output from *ftio* is readable by *cpio*(1), except as explained in the Cpio Compatibility section below.

Ftio must be invoked with exactly one of the options **−o**, **−O**, **−i**, **−I**, or **−g**. The option can be followed by modifiers, which must appear immediately after the option with no spaces between the option and the modifier, for example, **ftio −idxE**. See the Modifiers section below.

Tapedev specifies the name of an output file. A device on a remote machine can be specified in the form *machine:device*. *Ftio* creates a server, */etc/rmt*, on the remote machine to access the tape device.

Options

-o Copy out files onto *tapedev* together with path name and status information. If *pathnames* are specified, *ftio* recursively descends *pathnames* looking for files, and copies those files onto *tapedev*. If *pathnames* are not specified, *ftio* reads the standard input to obtain a list of path names to copy. In addition to copying the files onto the tape set, *ftio* generates, for each tape in the tape set, a tape header containing the current tape number, machine node name and type, operating system name, release and version numbers (all from *uname*(2)), username of the backup initiator, time and date of the backup, number of consecutive times the current media has been used, a comment field, and other items used internally by *ftio*. The tape header is separated from the main body of the archive by an end of file mark. The tape header can be read by invoking *cat*(1) with the device file name as the first argument. Note that device files written with the **−o** option (for example, **/dev/tty03**) do not transport to other implementations of HP-UX.

-O Copy out files in the same way as **ftio -ocva**, if no modifiers are used with the **−O**. However, if the file **.ftiorc** exists in the user's home directory, *ftio* opens this file and scans for lines preceded by **O=**. Options defined on matching lines are passed to *ftio* as if they had been passed in the original command. See the EXAMPLES section.

−i Extract, or copy in, files from *tapedev*, which is assumed to be the product of a previous **ftio -o** operation. Only files with names that match *patterns*, according to the rules of Pattern Matching Notation (see *regexp*(5)), are selected. In addition, a leading **!** within a pattern indicates that only those names should be selected that do *not* match the remainder of the pattern. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (that is, select all files). The extracted files are conditionally created and copied into the

current directory tree based upon the options described below. The permissions of the files are those of the previous **-o** operation.

−I Extract, or copy in, files in the same way as for **ftio -icdmv**, if no modifiers are used with the **−I**. However, if the file **.ftiorc** exists in the user's home directory, *ftio* opens this file, and scans for lines preceded by **I=**. Options defined on matching lines are passed to *ftio* as if they had been passed in the original command. See the EXAMPLES section.

−g Read the file list on *tapedev*. If *patterns* is specified, only file names that match are printed. Note that file names are always preceded by the volume that *ftio* expected the file to be on when the file list was created, and, therefore, only the last volume is valid in this respect.

−B *blksize* Specify the size (in bytes) of blocks written to tape. This number may end with **k**, which specifies multiplication by 1024. The use of larger blocks generally improves performance and tape usage. The maximum block size allowed is limited by the tape drive used. A default of 16384 bytes is set because this is the maximum block size on most Hewlett-Packard tape drives.

−D *type* Recursively descend a directory only if the file system to which it belongs is of type *type*, where *type* is either **hfs** or **nfs**.

−F *ignorenames* Arguments following **-F** specify *patterns* that should not be copied to the tape. The same rules apply for *ignorenames* as do for *patterns*, see the previous description for **ftio -i**.

−K *comment* Specify a comment to be placed in the *ftio* tape header.

−L *filelist* If *pathnames* is specified, perform the file search and generate a list of files to back up prior to actually commencing the backup. This list is then appended to the tape header of each tape in the backup as a list of files that *ftio* attempted to fit onto this tape. The last tape in the backup contains a catalog of where the files are in the archive set. If *pathnames* is not specified, the file list is taken from standard input before the backup begins. *Filelist* specifies the output file. In addition to generating file lists, the **-L** option implements tape checkpointing, allowing the backup to restart from a write failure on a bad medium.

−M Do not generate or expect tape headers, and change the default block size to 5120 bytes. This allows for full compatibility with *cpio*(1). See the Cpio Compatibility section below.

−N *datefile* Only files that are newer than the file specified in *datefile* are copied to tape.

−R Automatically resynchronize when *ftio* goes out of phase. This is useful when restoring from a multi-tape backup on tapes other than the first. The default behavior is that *ftio* asks the user if resyncing is required.

−S *script* Specify a command that is invoked every time a tape is completed in a multi-tape backup. The command is invoked by *sh*(1) with the following arguments: *script tape_no user_name*. Script is the string *script* specified by the **-S** option, *tape_no* is the number of the tape required, and *user_name* is the user who invoked *ftio*. Typically, the string *script* specifies a shell script which is used to notify the user that a tape change is required.

−T *tty* Specify alternate to **/dev/tty**. Normally **/dev/tty** is opened by *ftio* when terminal interaction is required.

−Z *nobufs* Specify the number of *blksize* chunks of memory to use as buffer space between the two processes, where *blksize* is the size of blocks written to the tape. The use of more chunks is usually better, but a point is reached where no improvement is gained, and in fact performance may deteriorate as buffer space is swapped out of main memory. A default value of 16 is set for *nobufs*, but the use of 32 or 64 may improve performance if your system is relatively unloaded. We recommend performing a backup with the system in single user mode.

Modifiers

a After files are copied out to tape, reset the access time to appear as though the file was not accessed by *ftio*.

c Write header information in ASCII character form for portability.

d When restoring files, create directories as needed.

f Copy in all files except those that match *patterns*.

h Archive the files to which symbolic links point as if they were normal files or directories. Normally, *ftio* archives the link itself.

m Retain previous file modification time and ownership of file. Restoring modification time is ineffective on directories that are being restored.

p At the end of the backup, print the number of blocks transferred, the total time taken (excluding tape rewind and change reel time), and the effective transfer rate calculated from these figures. These values are printed at the end of each tape if **p** is given twice.

t Print only a table of contents of the input. No files are created, read, or copied.

u Copy unconditionally (normally, an older file will not replace a newer file with the same name).

v Be verbose. Print a list of file names as well as tape headers. When used with the **t** modifier, the table of contents looks the same as the output of **ls -l** command (see *ls*(1)).

x Save or restore device special files. *Ftio* uses *mknod*(2) to recreate these files during a restore operation. Thus, this modifier is restricted to the superuser. This is intended for intrasystem (backup) use. Restoring device files onto a different system can be very dangerous.

A If copying from tape (**−i** or **−I** option), print all file names found on the archive, noting which files have been restored. This is useful when the user restores selected files, but wants to know which (if any) files are on the tape.

If copying to tape (**−o** or **−O** option), suppress warning messages regarding optional access control list entries. *Ftio*(1) does not backup optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file that has optional access control list entries.

E When archiving, store all files with absolute path names with a path name relative to the root directory (that is, all files whose name starts with / have the first / removed). On restoration, any files in the archive that have an absolute path name have the leading / removed from the path name and are restored relative to the current directory.

L Behave like the **−L** option, except that the file list is left in the current directory as the file **ftio.list**, instead of the file named in *filelist*.

H Search hidden subdirectories (context-dependent files or CDFs). Normally, only the CDF element matching the current context is archived, without expanding the path name to show the actual element. For more information on CDFs see *cdf*(4).

P On restoration, use *prealloc*(2) to pre-allocate disk space for the file. This vastly improves the localization of file fragments.

When the end of the tape is reached, *ftio* invokes *script* if the **-S** option has been exercised, rewinds the current tape, and then asks the user to mount the next tape.

To pass one or more metacharacters to *ftio* without having the shell expand them, protect them by either preceding each of them with a backslash (for example, **/usr***), or enclosing them in protective quotes (for example, **'/usr*'**).

Cpio Compatibility

Ftio uses the same archive format as *cpio*(1). However, by default *ftio* creates tape headers and uses a tape block size of 16k bytes. *Cpio*(1) by default uses 512 byte blocks. When used with the **−B** option, *cpio*(1) uses 5120 byte blocks. To achieve full compatibility with *cpio*(1) in either input or output mode, the user should specify the **M** modifier. **Ftio -oM** creates a single or multi-tape archive that has no tape headers, and, by default, the same block size as **cpio -[o|i]B**. An archive created by a **cpio -oB** command may be restored using **ftio -iM**. If the **M** modifier of *ftio* is combined with a **-B 512** block size specification, full compatibility with **cpio -[o | i]** (no **-B**) is achieved.

EXAMPLES

The first example below copies the entire contents of the file system (including special files) onto the tape drive **/dev/rmt/0h**:

```
ftio -ox /dev/rmt/0h /
```

The following example restores all the files on **/dev/rmt/0h**, relative to the current directory:

```
ftio -idxE /dev/rmt/0h
```

The following example demonstrates how to list the contents of a backup set created using **ftio -o**:

```
ftio -itv /dev/rmt/0h
```

Note that use of the **v** modifier gives a more detailed listing, and displays the contents of tape headers.

The next example demonstrates the use of the **.ftiorc** file. A **.ftiorc** file exists in the user's home directory with the following contents:

```
# Example .ftiorc file.
I= cdmuvEpp -B 16k -S /usr/local/bin/ftio.change
O= cavEpp -Z 8 -B 16k -S /usr/local/bin/ftio.change
```

Ftio is invoked with the following command line to backup the users home directory and the system binary directory:

```
ftio -O /dev/rmt/0h /user/my_home /bin
```

Because the **-O** option has been specified, the **.ftiorc** will be checked for additional options. In this case, character headers are generated, access times are reset, a listing of the files copied are printed to standard output, all file names are copied to **/dev/rmt/0h** with path names relative to **/**, performance data are printed when the backup is complete (and at every tape change), and, if the backup goes beyond one media the script, **/usr/local/bin/ftio.change** is invoked by *ftio* after each media is completed.

WARNINGS

Ftio uses System V shared memory and semaphores for its operation. The resources committed to these functions are not automatically freed by the system when the process terminates. *Ftio* does this only when it terminates normally, or when it terminates after receiving one the following signals: SIGHUP, SIGINT, SIGTERM. Any other signal is handled in the default manner described by *signal*(2). Note that the behavior for SIGKILL is to terminate the process without delay. Thus, if *ftio* receives a SIGKILL signal (as might be produced by the indiscriminate use of **kill -9**, see *kill*(1)), system resources used for shared memory and semaphores are not returned to the system. If it becomes necessary to terminate an invocation of *ftio*, use **kill -15**. Current system usage of shared memory and semaphores can be checked using *ipcs*(1). Committed resources can be removed using *ipcrm*(1).

AUTHOR

Ftio was developed by HP.

SEE ALSO

cpio(1), find(1), ipcs(1), ipcrm(1), kill(1), ls(1), rmt(1M), mknod(2), prealloc(2), signal(2), uname(2), cdf(4), acl(5), environ(5), lang(5), regexp(5), mt(7).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the characters matched by character class expressions in pattern matching notation.

LC_TIME determines the format and contents of date and time strings.

LANG determines the language in which messages are displayed.

If LC_COLLATE, LC_CTYPE, or LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ftio* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

NAME

gencat − generate a formatted message catalog file

SYNOPSIS

gencat [−**l**] *catfile msgfile ...*

DESCRIPTION

Message catalogs allow a program to process input and produce output according to local customs and languages. For details, see *Native Language Support* listed in the SEE ALSO section below.

Gencat merges each message source *msgfile* into a formatted message catalog *catfile* that can be accessed by *catgetmsg*(3C) or *catgets*(3C). If *catfile* does not exist, it is created. If *catfile* exists, its messages are included in the new *catfile*. If set and message numbers collide, the new message text in *file* replaces the old message text in *catfile*. A *msgfile* consists of message, directive and comment lines (all without leading spaces or tabs) described below. Except as noted, fields are separated by one or more space or tab characters.

$set *s* [*comment*]
: A **$set** directive specifies the set *s*, of the messages that follow until the next **$set** or end-of-file appears. The set number *s* is an unsigned integer in the range 1 − NL_SETMAX. Any string following the set number is treated as a comment. If a **$set** directive is not specified, messages are put in the default set NL_SETD.

  Set numbers must be in ascending order within a *msgfile* but need not be contiguous.

$delset *s* [*comment*]
: A **$delset** directive deletes the message set identified by the set number *s*, from an existing message catalog. Any string following the set number is treated as a comment.

m message_text
: A message line specifies a message number *m*, and associated message text. The message number *m* is an unsigned integer in the range 1 − NL_MSGMAX. The *message_text* is a C string, including spaces, tabs and \ (backslash) escapes, but by default without surrounding quotes (see **$quote** directive below). The message number *m* is separated from the *message_text* by a single space or tab character. The *message_text* begins with the first character following the separator and ends at new-line. Extra spaces or tabs (including any trailing spaces or tabs) are considered part of the *message_text*.

  The *message_text* of a message line is stored in *catfile* with message number *m* and set number *s* specified by the most recent **$set** directive.

  Message numbers must be in ascending order within a set but need not be contiguous.

  Note that the space or tab separator distinguishes insertion of a null message from deletion of a message. If a message line has a number and separator but no text, the message number and an associated null message string are stored in *catfile*. If a message line has a number but neither separator nor text, the message number and its associated message text are deleted from *catfile*.

−**l**
: If the −**l** option is specified, the length of *message_text* must be no more than MAX_BUFLEN − 1 bytes. If the −**l** option is not specified, the length of *message_text* must be no more than NL_TEXTMAX bytes. See *catgetmsg*(3C), *catgets*(3C), *catread*(3C), and *getmsg*(3C) for message length limits imposed by these routines.

$quote [*q comment*] A **$quote** directive specifies a quote character *q*, used to surround *message_text* and make leading and trailing space visible in a message line. Any string following the specified quote character *q* is treated as a comment. By default, or if a quote character *q* not is supplied, quoting of *message_text* is not recognized.

$ *comment* A **$** followed by a space or tab is treated as a comment and can appear anywhere in a file. A line consisting of zero or more spaces or tabs is treated as a comment line.

NL_TEXTMAX, NL_SETMAX and NL_MSGMAX are defined in <**limits.h**>. NL_SETD is defined in <**nl_types.h**>. MAX_BUFLEN is defined in <**msgcat.h**>.

WARNINGS

The **$quote** directive is not currently supported on MPE or RTE systems. (MPE and RTE are HP operating systems.)

DEPENDENCIES

Series 300

The **−l** option is not currently supported.

AUTHOR

Gencat was developed by HP and the X/Open Company, Ltd.

SEE ALSO

findmsg(1), insertmsg(1), catgetmsg(3C), catgets(3C), catopen(3C).
Native Language Support, in *HP-UX Concepts and Tutorials: Device I/O and User Interfacing.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of messages as single- and/or multi-byte characters.

Messages will be issued in LANG if it is set to a valid language and LANG messages are available. Otherwise "C" locale messages will be issued.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *gencat* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

gencat: XPG2, XPG3

NAME

get − get a version of an SCCS file

SYNOPSIS

get [−r*SID*] [−c*cutoff*] [−**e**] [−**b**] [−i*list*] [−x*list*] [−**k**] [[−**l**[**p**]] [−**p**] [−**s**] [−**m**] [−**n**] [−**g**] [−**t**] [−w*string*] [−a*seq-number*] *file*...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with −. The arguments can be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also **FILES** below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

−r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the −**e** keyletter is also used), as a function of the SID specified.

−c*cutoff* *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters can separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: **"−c77/2/2 9:22:25"**. Note that this implies that one can use the %E% and %U% identification keywords (see below) for nested *gets* within, for example, a *send*(1) command:

~!get "−c%E% %U%" s.file

−**e** Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The −**e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *get*for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get** −**e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an −**e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the −**k** keyletter in place of the −**e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the −**e** keyletter is used.

−**b** Used with the −**e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is

not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

−i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= SID | SID − SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

−x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **−i** keyletter for the *list* format.

−k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **−k** keyletter is implied by the **−e** keyletter.

−l[p] Causes a delta summary to be written into an *l-file*. If **−lp** is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*. The user must have read permission for the s-file in order to use the **−l** option.

−p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 (stderr) instead, unless the **−s** keyletter is used, in which case it disappears.

−s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

−m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** keyletter generated format.

−g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Used to access the most recently created ("top") delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).

−w *string* Substitute *string* for all occurrences of **@%M%** when *get*ing the file.

−a*seq-number* The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile*(4)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter, and users should not use it. If both the **−r** and **−a** keyletters are specified, the **−a** keyletter is used. Care should be taken when using the **−a** keyletter in conjunction with the **−e** keyletter, as the SID of the delta to be created may not be what one expects. The **−r** keyletter can be used with the **−a** and **−e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** keyletter is used included deltas are listed following the notation "Included"; if the **−x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**−b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

\# Successor.

† The **−b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of − means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

Identification Keywords

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see *admin*(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string **@(#)** recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for HP-UX system program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-HP-UX system program files. %A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
   * otherwise.

b. A blank character if the delta was applied or was not applied and ignored;
   * if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was not applied:
   "I": Included.
   "X": Excluded.
   "C": Cut off (by a **−c** keyletter).

d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an **−e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **−e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **−i** keyletter argument if it was present, followed by a blank and the **−x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the **−e** keyletter is used.

An l-file cannot be generated when **−g** is used. In other words, **−g −l** does not work.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(4).

SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *get* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

STANDARDS CONFORMANCE

get: SVID2, XPG2, XPG3

NAME

getaccess – list access rights to file(s)

SYNOPSIS

getaccess [**–u** *user*] [**–g** *group*[,*group*]...] [**–n**] *file* ...
getaccess **–r** [**–n**] *file* ...

Remarks:

To ensure continued conformance with emerging industry standards, features described in this manual entry are likely to change in a future release.

DESCRIPTION

Getaccess lists for the specified files the effective access rights of the caller (that is, for their effective user ID, effective group ID, and supplementary groups list). By default, the command prints a symbolic representation of the user's access rights to the named file: **r** or – for read/no read, **w** or – for write/no write, and **x** or – for execute/no execute (for directories, search/no search), followed by the file name.

Options

–u *user*
: List access for the given user instead of the caller. A *user* can be a known user name, a valid ID number, or @, representing the file's owner ID. If information about more than one file is requested, the value of @ can differ for each.

  This option sets the user ID only. The access check is made with the caller's effective group ID and supplementary group IDs unless **–g** is also specified.

–g *group*[,*group*]...]
: List access for the given group(s) instead of the caller's effective group ID and supplementary groups list. A *group* can be a known group name, a valid ID number, or @, representing the file's group ID. If information about more than one file is requested, the value of @ can differ for each.

–r
: List access using the caller's real user ID, group ID, and supplementary groups list, instead of effective ID values.

–n
: List access rights numerically (octal digits **0**..**7** instead of **rwx**) for each file requested. The bit values R_OK, W_OK, and X_OK are defined in the file <**unistd.h**>.

Checking access using access control lists is described in *acl*(5).

In addition, the write bit is cleared for files on read-only file systems or shared-text programs being executed. The execute bit is not turned off for shared-text programs open for writing because it is not possible to ascertain whether a file open for writing is a shared-text program.

Superuser processes have read and write access to all files. However, write access is denied for files on read-only file systems or shared-text programs being executed. Execute access is allowed if and only if the file is not a regular file or the execute bit is set in any of the file's ACL entries.

To use *getaccess* successfully, the caller must have search access in every directory component of the path name of the *file*. *Getaccess* verifies search access first by using the caller's effective IDs, regardless of the user and group IDs specified. This is distinct from the case in which the caller can search the path but the user for whom access is being checked does not have access to the file.

Note: a file name argument of – has no special meaning (such as standard input) to *getaccess*.

RETURN VALUE

If *getaccess* succeeds, it returns a value of 0. If it is invoked incorrectly or encounters an unknown user or group name, it prints an appropriate message to standard error and returns a

value of 1. If any file is nonexistent or unreachable (by the caller), it prints an appropriate message to standard error, continues, and then returns a value of 2.

EXAMPLES

The following command prints the caller's access rights to *file1* using the file's group ID instead of the caller's effective group ID and groups list.

```
getaccess -g@ file1
```

Here's how to check access by user "ggd" in groups "red" and 19 to all files in the current directory, with access rights expressed as octal values.

```
getaccess -u ggd -g red,19 -n .* *
```

Here's how to list access rights for all files under "mydir".

```
find mydir -print | sort | xargs getaccess
```

AUTHOR

Getaccess was developed by HP.

FILES

/etc/passwd
/etc/group

SEE ALSO

chacl(1), lsacl(1), getaccess(2), glossary(9).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *getaccess* behaves as if all internationalization variables are set to "C". See *environ*(5).

NAME

getcontext – display current context

SYNOPSIS

getcontext

DESCRIPTION

Getcontext displays the context of the invoking process. The context is displayed as a list of words separated by spaces, and is used when matching context dependent files, see *cdf*(4).

AUTHOR

Getcontext was developed by HP.

SEE ALSO

getcontext(2), cdf(4), context(5).

NAME

getopt – parse command options

SYNOPSIS

getopt *optstring args*

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters (see *getopt*(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space.

The positional parameters ($1 $2 ...) of the shell are reset so that each option is preceded by a – and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

Getopt recognizes two hyphens (– –) to delimit the end of the options. If absent, *getopt* places – – at the end of the options.

The most common use of *getopt* is in the shell's **set** command (see the example below). There, *getopt* converts the command line to a more easily parsed form. *Getopt* writes the modified command line to the standard output.

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

EXAMPLES

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- 'getopt abo: $*'
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)        FLAG=$i; shift;;
        -o)             OARG=$2; shift 2;;
        --)             shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

WARNINGS

Getopt option arguments may not be null strings nor contain embedded blanks.

SEE ALSO

sh(1), getopt(3C).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *getopt* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

NAME

getprivgrp – get special attributes for group

SYNOPSIS

getprivgrp [**-g** | *group–name*]

DESCRIPTION

Getprivgrp lists the access privileges of privileged groups set by *setprivgrp*(1M). If *group–name* is supplied, access privileges are listed for that group only. If the caller is not a member of *group–name*, no information is displayed. If **–g** is used, *getprivgrp*(1M) lists access privileges that have been granted to all groups. Otherwise, access privileges are listed for all privileged groups to which the caller belongs.

The superuser is a member of all groups. Access privileges include RTPRIO, MLOCK, CHOWN, LOCKRDONLY, and SETRUGID.

In the HP Clustered environment, privilege groups are maintained separately for each member of the cluster. The CHOWN privilege from a cnode is determined by the privilege groups set up on the root server.

AUTHOR

Getprivgrp was developed by HP.

SEE ALSO

getprivgrp(2), setprivgrp(1M), privgrp(4).

NAME

gprof – display call graph profile data

SYNOPSIS

gprof [*options*] [*a.out* [*gmon.out* ...]]

DESCRIPTION

Gprof produces an execution profile of C, Pascal, and Fortran programs. The effect of called routines is incorporated into the profile of each caller. Profile data is taken from the call graph profile file (**gmon.out** default) that is created by programs compiled with the **–G** option of *cc*(1), *pc*(1), and *f77*(1). That option also links in versions of the library routines that are compiled for profiling. The symbol table in the named object file (**a.out** default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each function in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. *Gprof* discovers all cycles in the call graph. All calls made into the cycle share the time of that cycle. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendants. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how the time of this function and the time of its descendants are propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle, each with their contributions to the time and call counts of the cycle.

Options:

–a Suppress printing statically declared functions. If this option is given, all relevant information about the static function (for example, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

–b Suppress printing a description of each field in the profile.

–c Identify the static call graph of the program using a heuristic that examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

–e *name* Suppress printing the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that are not suppressed). More than one **–e** option can be given. Only one *name* can be given with each **–e** option.

–E *name* Suppress printing the graph profile entry for routine *name* (and its descendants) as **–e** above, and also exclude the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **–E mcount –E mcleanup** is the default.)

–f *name* Print only the graph profile entry of the specified routine *name* and its descendants. More than one **–f** option can be given. Only one *name* can be given with each **–f** option.

–F *name* Print only the graph profile entry of the routine *name* and its descendants (as **–f** above) and also uses only the times of the printed routines in total time and percentage computations. More than one **–F** option can be given. Only one *name* can be given with each **–F** option. The **–F** option overrides the **–E** option.

−s Produce a profile file **gmon.sum** that represents the sum of the profile information in all specified profile files. This summary profile file can be given to subsequent executions of *gprof* (probably also with a **−s**) to accumulate profile data across several runs of an *a.out* file.

−z Display routines that have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **−c** option for discovering which routines were never called.

WARNINGS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. It is assumed that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

Parents that are not profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call *exit*(2) or return normally for the profiling information to be saved in the **gmon.out** file.

DEPENDENCIES

Series 800

The **-c** option is not supported.

AUTHOR

Gprof was developed by The University of California, Berkeley.

FILES

a.out	Default object file.
gmon.out	Default dynamic call graph and profile.
gmon.sum	Summarized dynamic call graph and profile.
/usr/lib/gprof.callg	Call graph description.
/usr/lib/gprof.flat	Flat profile description.

SEE ALSO

cc(1), f77(1), pc(1), prof(1), exit(2), profil(2), monitor(3C).

gprof: A Call Graph Execution Profiler; Graham, S.L., Kessler, P.B., McKusick, M.K.; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*; SIGPLAN Notices; Vol. 17, No. 6, pp. 120-126, June 1982.

NAME

grep, egrep, fgrep – search a file for a pattern

SYNOPSIS

grep [*options*] [*expression*] [*file ...*]

egrep [*options*] [*expression*] [*file ...*]

fgrep [*options*] [*strings*] [*file ...*]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* supports the Basic Regular Expression syntax (see *regexp*(5)). *Egrep* supports the Extended Regular Expression syntax (see *regexp*(5)) with the extension that newline is also a regular expression alternation character. *Fgrep* patterns are fixed *strings*, making it a fast and compact means for finding known text strings.

Options

−v	All lines but those matching are printed.
−x	(Exact) only lines matched in their entirety are printed (*fgrep* only).
−c	Only a count of matching lines is printed.
−i	Ignore uppercase/lowercase distinction during comparisons.
−l	Only the names of files with matching lines are listed (once), separated by newlines.
−n	Each line is preceded by its relative line number in the file.
−b	Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
−s	The error messages produced for nonexistent or unreadable files are suppressed.
−e *expression*	Same as a simple *expression* argument, but useful when the *expression* begins with a hyphen (−).
−f *file*	The regular *expression* (*grep* and *egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters **$**, *, **[**, ^, **|**, **( )**, and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes ´...´.

Fgrep searches for lines that contain one of the *strings*, each of which is separated from the next by a newline.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

EXAMPLES

In the Bourne shell (*sh*(1)) the following example searches two files, finding all lines containing occurrences of any of four strings:

```
fgrep ´if
then
else
fi´ file1 file2
```

Note that the single quotes are necessary to tell *fgrep* when the strings have ended and the file names have begun.

For the C shell (*csh*(1)) the following command can be used:

```
fgrep ´if\
then\
else\
fi´ file1 file2
```

To search an **address** file with the following entries:

Ken 112 Warring St. Apt. A
Judy 387 Bowditch Apt. 12
Ann 429 Sixth St.

the command:

grep Judy address

would print:

Judy 387 Bowditch Apt. 12

WARNINGS

Lines matching the pattern but which are longer than BUFSIZ-1 bytes might not be found as a match or might be only partially copied to the standard output. *Grep*, *fgrep*, and *egrep* differ in their handling of long lines. (BUFSIZ is defined in **/usr/include/stdio.h**.)

If a line has embedded nulls, *grep* and *egrep* search only up to the first null; if a match is found the entire line is output.

SEE ALSO

sed(1), sh(1), environ(5), lang(5), regexp(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as letters, the case information for the *-i* option, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, the commands behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

grep: SVID2, XPG2, XPG3

egrep: SVID2, XPG2, XPG3

fgrep: SVID2, XPG2, XPG3

NAME

groups – show group memberships

SYNOPSIS

groups [**–p**] [**–g**] [**–l**] [*user*]

DESCRIPTION

The *groups* command shows the groups to which you or the optionally specified user belong. If invoked with no arguments, *groups* prints the current access list returned by *getgroups*(2).

Each user belongs to a group specified in the password file **/etc/passwd** and possibly to other groups as specified in the files **/etc/group** and **/etc/logingroup**. A user is granted the permissions of those groups specified in **/etc/passwd** and **/etc/logingroup** at login time. The permissions of the groups specified in **/etc/group** are normally available only with the use of *newgrp*(1). If a user name is specified with no options, *groups* prints the union of all these groups.

The **-p**, **-g**, and **-l** options limit the printed list to those groups specified in **/etc/passwd**, **/etc/group**, and **/etc/logingroup**, respectively. If a user name is not specified with any of these options, *cuserid*(3S) will be called to determine the default user name.

The printed list of groups is sorted in ascending collation order (see Environment Variables below).

EXAMPLES

The command:

groups -l tim

checks the file **/etc/logingroup** and displays all groups to which the user **tim** belongs.

AUTHOR

Groups was developed by the University of California, Berkeley.

FILES

/etc/group
/etc/logingroup
/etc/passwd

SEE ALSO

id(1), newgrp(1), getgroups(2), initgroups(3C), cuserid(3S), group(4).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *groups* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

NAME

head – give first few lines

SYNOPSIS

head [–count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

SEE ALSO

tail(1).

NAME

help – ask for help

SYNOPSIS

help [*args*]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1 — Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **ge6**, for message 6 from the *get* command).

type 2 — Does not contain numerics (as a command, such as **get**)

type 3 — Is all numeric (e.g., **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

DIAGNOSTICS

Use *help*(1) for explanations.

WARNINGS

Only SCCS and a very few other commands currently use *help*.

FILES

/usr/lib/help — directory containing files of message text.

/usr/lib/help/helploc — file containing locations of help files not in **/usr/lib/help**.

SEE ALSO

SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *help* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [*nameofhost*]

DESCRIPTION

The *hostname* command prints the name of the current host, as given in the *gethostname*(2) system call. The superuser can set the *hostname* by giving the argument *nameofhost*; this is usually done in the startup script **/etc/rc**. See the node manager's documentation supplied with your system for details on how this name is used.

AUTHOR

Hostname was developed by the University of California, Berkeley.

SEE ALSO

uname(1), gethostname(2), sethostname(2), uname(2).

NAME

hp – handle special functions of HP 2640 and HP 2621-series terminals

SYNOPSIS

hp [**−e**] [**−m**]

DESCRIPTION

Hp supports special functions of the Hewlett-Packard 2640 and 2621 series of terminals, with the primary purpose of producing accurate representations of most *nroff*(1) output. A typical use is:

nroff −h files ... | hp

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

Options

−e Specify that your terminal has the "display enhancements" feature, to make maximal use of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, that is, dark-on-light, rather than light-on-dark.

−m Request minimization of output by removing new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; that is, any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

DIAGNOSTICS

"line too long" if the representation of a line exceeds 1,024 characters.
The exit codes are **0** for normal termination, and **2** for all errors.

WARNINGS

An "overstriking sequence" is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text "disappear"; in particular, tables generated by *tbl*(1) that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to *hp* is piped through *col*(1).
Although some terminals do provide numerical superscript characters, no attempt is made to display them.

SEE ALSO

col(1), neqn(1), nroff(1), tbl(1).

NAME

hpiutil – create and maintain ALLBASE/HP-UX HP IMAGE network database

SYNOPSIS

hpiutil

REMARKS

The ALLBASE/HP-UX product must be previously installed on the system for *hpiutil* to function.

DESCRIPTION

Hpiutil invokes the HP IMAGE utility program for creating and maintaining an ALLBASE/HP-UX HP IMAGE network database. No options are available with this command. *Hpiutil* can be executed on all database files by all system users.

AUTHOR

Hpiutil was developed by Hewlett-Packard.

FILES

/usr/bin/hpdbdaemon	Cleanup daemon program file.
/usr/bin/hpimage	HP IMAGE program file.
/usr/bin/hpiutil	HPIUTIL program file.
/usr/lib/hpica000	HP IMAGE message catalog file.

SEE ALSO

ALLBASE/HP-UX HP IMAGE Reference Manual.

NAME

hyphen – find hyphenated words

SYNOPSIS

hyphen [files]

DESCRIPTION

Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

EXAMPLE

The following will allow the proofreading of *nroff* hyphenation in *textfile*.

```
mm textfile | hyphen
```

SEE ALSO

mm(1), nroff(1).

BUGS

Hyphen cannot cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

iconv – code set conversion

SYNOPSIS

iconv -f *fromcode* **-t** *tocode* [*file* ...]

DESCRIPTION

Iconv converts the encoding of characters in the input files from the *fromcode* code set to the *tocode* code set and writes the results on standard output. If no input files are given, *iconv* reads from standard input. If – appears as an input file name, *iconv* reads standard input at that point. You can use – – to delimit the end of options (see *getopt*(3C)).

Options

The following options are recognized:

–f *fromcode* Identify the code set corresponding to option argument *fromcode* as the code set that the input will be converted "from".

–t *tocode* Identify the code set corresponding to option argument *tocode* as the code set that the input will be converted "to".

The *fromcode* and *tocode* names can be any length, but only the first four and the last letter are used to identify the code set. The names can contain any letter except a closing curly bracket ("}"). HP supplied *fromcode* and *tocode* names and their corresponding code sets include:

Names	Code Set
iso8859_1	ISO 8859/1
arabic8	HP ARABIC8
greek8	HP GREEK8
hebrew8	HP HEBREW8
kana8	HP KATAKANA8
roman8	HP ROMAN8
turkish8	HP TURKISH8
japanese15	HP JAPANESE15
korean15	HP KOREAN15
roc15	HP Traditional CHINESE15
american_e	American EBCDIC
arabic_e	Arabic EBCDIC
c-french_e	Canadian-French EBCDIC
cht_e	Traditional Chinese EBCDIC
danish_e	Danish EBCDIC
dutch_e	Dutch EBCDIC
english_e	English EBCDIC
finnish_e	Finnish EBCDIC
french_e	French EBCDIC
german_e	German EBCDIC
greek_e	Greek EBCDIC

hebrew_e	Hebrew EBCDIC
icelandic_e	Icelandic EBCDIC
italian_e	Italian EBCDIC
japanese_e	Japanese EBCDIC
katakana_e	Katakana EBCDIC
korean_e	Korean EBCDIC
norwegian_e	Norwegian EBCDIC
portuguese_e	Portuguese EBCDIC
spanish_e	Spanish EBCDIC
swedish_e	Swedish EBCDIC
turkish_e	Turkish EBCDIC
jis	Japanese Industrial Standard
sjis	Shifted-Japanese Industrial Standard
ujis	Japanese Extended UNIX Code

WARNINGS

If an input character does not have a valid equivalent in the code set selected by the **−t** option (the "to" code set), it is mapped to as system defined default character.

If an input character does not belong to the code set selected by the **−f** option (the "from" code set), the command will terminate.

EXAMPLES

The following example converts the contents of file **foo** from code set Roman8 to ISO 8859/1 and stores the results in file **bar**.

```
iconv -f roman8 -t iso8859_1 foo > bar
```

AUTHOR

Iconv was developed by HP.

SEE ALSO

roman8(5), getopt(3C)

EXTERNAL INFLUENCES

Environment Variables

The LANG environment variable determines the language in which messages are displayed.

International Code Set Support

Single and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

iconv: XPG3

NAME

id − print user and group IDs and names

SYNOPSIS

id [**−gnru**]

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed. The following options are available:

−g Output only the group ID. The default output format is "%d\n". To modify, use the −n option. The default group ID is the effective group ID. To modify, use the −r option.

−n With −u or −g, output the name in the format "%s\n" instead of the numeric ID.

−r With −u or −g, output the real ID instead of the effective ID.

−u Output only the user ID. The default output format is "%d\n". To modify, use the −n option. The default group ID is the effective group ID. To modify, use the −r option.

AUTHOR

Id was developed by Hewlett-Packard Company and AT&T Bell Laboratories.

SEE ALSO

logname(1), getuid(2).

STANDARDS CONFORMANCE

id: SVID2, XPG2, XPG3

NAME

ident – identify files in RCS

SYNOPSIS

ident file ...

DESCRIPTION

Ident searches the named files for all occurrences of the pattern *$keyword:...$*, where *keyword* is one of the following:

Author
Date
Header
Locker
Log
Revision
Source
State

These patterns are normally inserted automatically by the RCS command *co*(1), but can also be inserted manually.

Ident works on text files as well as object files. For example, if the C program in file **f.c** contains:

```
char rcsid[] = "$Header: Header information $";
```

and **f.c** is compiled into **f.o**, the command:

```
ident f.c f.o
```

will print:

```
f.c:
        $Header: Header information $
f.o:
        $Header: Header information $
```

AUTHOR

Ident was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

ci(1), co(1), rcs(1), rcsdiff(1), rcsintro(5), rcsmerge(1), rlog(1), rcsfile(4).

NAME

insertmsg – use findstr(1) output to insert calls to catgets(3C)

SYNOPSIS

insertmsg [**−h**] [**−n** *number*] [**−i** *amount*] [**−s** *number*] *stringlist*

DESCRIPTION

Insertmsg examines the file *stringlist*, which is assumed to be the output of *findstr*(1) less any strings that do not need to be localized and have been removed by editing. If the **−h** option is specified, *insertmsg* places the following lines at the beginning of each file named in *stringlist:*

```
#ifndef NLS
#define catgets(i,sn,mn,s) (s)
#else NLS
#define NL_SETN number
#include <nl_types.h>
#endif NLS
```

The *number* is a set number defined by the **-s** option; the default is **1**. For each string in *stringlist*, *insertmsg* surrounds the string in the corresponding file with an expression of the form:

(catgets(catd,NL_SETN,*msg_num,***"***default string***"))**

The *default string* is the original string referenced by the line in *stringlist*, and *msg_num* is replaced by the message number assigned to that string. The assigned message numbers begin with the number defined in the **−n** option and are incremented by the amount defined in the **−i** option. The default is **1** for both the starting message number and the increment. If *name***.c** is the file to be modified, as specified within the *stringlist* file, *insertmsg* places the modified source in **nl_***name***.c**. The user must then manually edit the file **nl_***name***.c** to insert the following statements:

```
nl_catd catd;
catd = catopen("appropriate message catalog",0);
```

The data type **nl_catd** is defined in **<nl_types.h>** and **catd** is a parameter to the calls to *catgets*, which are inserted for each string from *stringlist*.

Insertmsg also sends to the standard output a file that can be used as input to *gencat*(1).

DIAGNOSTICS

If *insertmsg* does not find opening or closing double quotes where required in the strings file, it prints "insertmsg exiting : lost in strings file" and dies. If this happens, check the strings file to ensure that the lines that have been kept there have not been altered.

WARNINGS

Previous implementations of *insertmsg* inserted a call to *nl_msg* rather than *catgets*. *Nl_msg* outputs *default string* if the message retrieval is unsuccessful for any reason, whereas *catgets* reverts to *default string* only when it is unable to open the message catalog. Any other failed attempt to retrieve the message results in the output of a null string.

If the **−h** hoption is not used, the user may need to manually add the following statement to the file created by *insertmsg*:

#include <nl_types.h>

Insertmsg inserts a pointer to a static area that is overwritten on each call.

AUTHOR

Insertmsg was developed by HP.

SEE ALSO

findstr(1), gencat(1), catgets(3C), catopen(3C).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *insertmsg* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [-t] [interval [count]]

DESCRIPTION

Iostat iteratively reports for each disk the number of seeks per second, kilobytes transferred per second, and the milliseconds per average seek. If given a **-t** argument, it also reports the number of characters read from and written to terminals, and the percentage of time the system has spent in user mode, in user mode running low priority (niced) processes, in system mode, and idling.

To compute this information, for each disk, seeks and data transfer completions and number of words transferred are counted; for terminals collectively, the number of input and output characters are counted. Also, the state of each disk is examined HZ times per second (as found in <sys/param.h>) and a tally is made if the disk is active. From these numbers and given the transfer rates of the devices it is possible to determine average seek times for each device.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for the time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports.

AUTHOR

Iostat was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/dev/kmem
/hp-ux

SEE ALSO

vmstat(1).

NAME

ipcrm – remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

–q *msqid* Remove the message queue identifier *msqid* from the system and destroy the message queue and data structure associated with it.

–m *shmid* Remove the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

–s *semid* Remove the semaphore identifier *semid* from the system and destroy the set of semaphores and data structure associated with it.

–Q *msgkey* Remove the message queue identifier, created with key *msgkey*, from the system and destroy the message queue and data structure associated with it.

–M *shmkey* Remove the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

–S *semkey* Remove the semaphore identifier, created with key *semkey*, from the system and destroy the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

In the HP Clustered environment messages, semaphores, and shared memory are not global across members of the cluster. Therefore, *ipcrm* can be used to remove only local identifiers.

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

STANDARDS CONFORMANCE

ipcrm: SVID2

NAME

ipcs – report inter-process communication facilities status

SYNOPSIS

ipcs [*options*]

DESCRIPTION

Ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

–q Print information about active message queues.

–m Print information about active shared memory segments.

–s Print information about active semaphores.

If any of the options **–q**, **–m**, or **–s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

–b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

–c Print creator's login name and group name. See below.

–o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

–p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

–t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.

–a Use all print *options*. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

–C *corefile* Use the file *corefile* in place of **/dev/kmem**.

–N *namelist* The argument will be taken as the name of an alternate *namelist* (**/hp-ux** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:
- **q** message queue;
- **m** shared memory segment;
- **s** semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:
The first two characters are:

R if a process is waiting on a *msgrcv*;
S if a process is waiting on a *msgsnd*;
D if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
C if the associated shared memory segment is to be cleared when the first attach is executed;
– if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.
The permissions are indicated as follows:

r if read permission is granted;
w if write permission is granted;
a if alter permission is granted;
– if the indicated permission is *not* granted.

OWNER (all) The login name of the owner of the facility entry.

GROUP (all) The group name of the group of the owner of the facility entry.

CREATOR (a,c) The login name of the creator of the facility entry.

CGROUP (a,c) The group name of the group of the creator of the facility entry.

CBYTES (a,o) The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o) The number of messages currently outstanding on the associated message queue.

QBYTES (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p) The process ID of the last process to send a message to the associated queue.

LRPID (a,p) The process ID of the last process to receive a message from the associated queue.

STIME (a,t) The time the last message was sent to the associated queue.

RTIME (a,t) The time the last message was received from the associated queue.

CTIME (a,t) The time when the associated entry was created or changed.

NATTCH (a,o) The number of processes attached to the associated shared memory segment.

SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

In the HP Clustered environment messages, semaphores, and shared memory are not global across members of the cluster. Therefore, *ipcs* can be used to report status only on local interprocess communication facilities.

WARNINGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

FILES

/etc/group	group names
/hp-ux	system namelist
/dev/kmem	memory
/etc/passwd	user names

SEE ALSO

msgop(2), semop(2), shmop(2).

STANDARDS CONFORMANCE

ipcs: SVID2

NAME

iquery – interactively add, delete, update, report data in an ALLBASE/HP-UX HP IMAGE network database

SYNOPSIS

iquery

REMARKS

The ALLBASE/HP-UX product must be previously installed on the system for *iquery* to function.

DESCRIPTION

Iquery invokes the interactive tool for adding, deleting, updating, and reporting data in an ALLBASE/HP-UX HP IMAGE network database. No options are available with this command. *Iquery* can be executed by all system users.

FILES

/usr/bin/hpdbdaemon	Cleanup daemon program file.
/usr/bin/hpimage	HP IMAGE program file.
/usr/bin/iquery	IQUERY program file.
/usr/bin/iquerycf	IQUERY program file.
/usr/lib/hpica000	HP IMAGE message catalog file.
/usr/lib/hpiqc000	IQUERY message catalog file.

AUTHOR

Iquery was developed by Hewlett-Packard.

SEE ALSO

ALLBASE/HP-UX IQUERY Reference Manual.

NAME

isql – ALLBASE/HP-UX interactive SQL interface

SYNOPSIS

isql

REMARKS

The ALLBASE/HP-UX product must be previously installed on the system for *isql* to function.

DESCRIPTION

Isql invokes the Interactive SQL interface for defining and accessing an ALLBASE/HP-UX relational DataBase Environment (DBEnvironment). There are no options available with this command. *Isql* can be executed by all system users.

AUTHOR

Isql was developed by Hewlett-Packard.

FILES

/usr/bin/hpdbdaemon	Cleanup daemon program file.
/usr/lib/hpsqlproc	HP SQL program file.
/usr/bin/isql	Interactive SQL program file.
/usr/bin/sqlutil	SQLUTIL program file.
/usr/lib/hpsqlcat	HP SQL message catalog file.
/usr/lib/nls/$LANG/hpsqlcat	Localized message catalog file.
/usr/lib/isqlwel	Interactive SQL welcome banner file.
/usr/lib/nls/$LANG/isqlwel	Localized welcome banner file.

SEE ALSO

ALLBASE/HP-UX ISQL Reference Manual.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

join − relational database operator

SYNOPSIS

join [*options*] *file1 file2*

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is −, the standard input is used.

File1 and *file2* must be sorted in increasing collating sequence (see Environment Variables below) on the fields on which they are to be joined; normally the first in each line.

The output contains one line for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, followed by the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

−**a***n*	In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
−**e** *s*	Replace empty output fields by string *s*.
−**j** *n m*	Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.
−**o** *list*	Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
−**t***c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.
−**1 f**	Join on the *field*th field of file 1. Fields are numbered starting with 1.
−**2 f**	Join on the *field*th field of file 2. Fields are numbered starting with 1.

EXAMPLE

The following command line joins the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name, and the login directory. It is assumed that the files have been sorted in the collating sequence defined by the LC_COLLATE or LANG environment variable on the group ID fields.

```
join −1 4 −2 3 −o 1.1 2.1 1.6 −t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

With default field separation, the collating sequence is that of **sort −b**; with **−t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk* are incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence *join* expects from input files.

LC_CTYPE determines the alternative blank character as an input field separator, and the interpretation of data within files as single and/or multi-byte characters. LC_CTYPE also determines whether the separator defined through the **−t** option is a single- or multi-byte character.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *join* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

STANDARDS CONFORMANCE

join: SVID2, XPG2, XPG3

NAME

jtos, jtou, stoj, stou, utoj, utos − code set conversion for JIS, Shift JIS, and UJIS.

SYNOPSIS

jtos [options] [files]
jtou [options] [files]
stoj [options] [files]
stou [options] [files]
utoj [options] [files]
utos [options] [files]

DESCRIPTION

Jtos (*jtou, stoj, stou, utoj, utos*) converts Japanese characters in *files* from one code set to another, and outputs the converted files by default, to the standard output. The standard input is used for *files* by default.

Jtos converts JIS to SJIS (Shift JIS).

Jtou converts JIS to UJIS ("UNIXized" extended JIS).

Stoj converts SJIS to JIS.

Stou converts SJIS to UJIS.

Utoj converts UJIS to JIS.

Utos converts UJIS to SJIS.

The meanings of the options are:

−KI=*string* Specify the control sequence which designates JIS Kanji to the G0 set. Notation for special characters is the same as in *echo*(1). One arbitrary character can be represented in *string* by two hexadecimal digits preceded by

as in

In this case, backslash (or Yen currency symbol) must be protected from interpretation by the shell. Also, an entire string can be expressed with continuous two-digit hexadecimal numbers and only one preceding **X**, as in **−KI**=**X***nn*... . Without this option, the system-default control sequence, 33$B , is used. This option is supported only for *jtos*, *jtou*, *stoj*, and *utoj*.

−KO=*string* Specify the control sequence which designates a single-byte character set such as ASCII and katakana to the G0 set. Notation for special characters is as same as that for **−KI**. Without this option, the system-default control sequence, 33(J , is used. This option is supported only for *jtos*, *jtou*, *stoj*, and *utoj*.

−K2I=*string* Specify the control sequence which designates extension Kanji character set to the G0 set. Notation for special characters is as same as that for **−KI**. Without this option, the system-default control sequence, "**\033$C**", is used. This option is supported only for *jtos*, *jtou*, *stoj*, and *utoj*.

−7 Effectuate 7-bit code processing (SI/SO processing) for JIS single-byte katakana. Without this option, only 8-bit code processing is effective for JIS single-byte katakana. This option is supported only for *jtos*, *jtou*, *stoj*, and *utoj*.

−a Abort the process when an invalid Kanji code is encountered in *files*. Without this option, the process is continued regardless of invalid Kanji

code.

−c Allow the control characters to appear on the output only while JIS Kanji is designated and invoked. Without this option, the control characters always appear on the output whenever a single-byte character set is designated and invoked. This option is supported only for *stoj* and *utoj*.

−e Do not output invalid Kanji codes contained in *files*. When this option is not used, any invalid Kanji codes appear in the output without conversion.

−d Direct the output to *files* instead of the standard output.

−D Instead of using the standard output, Direct the output to files whose names are determined by converting the file names in *files* to uppercase (for *jtos*, and *utos*), or converting uppercase letters in the names of *files* to lowercase (for *stoj*, and *stou*). This option is supported only for *jtos*, *stoj*, *stou*, and *utos*.

−r Map carriage-return and line-feed character pairs (**\r\n**) to line-feed only (**\n**), and process **0x1a** as the end-of-file character in SJIS input code (for *stoj* and *stou*), or map line-feed characters (**\n**), to carriage-return and line-feed character pairs (**\r\n**) and append the end-of-file character **0x1a** in output SJIS code (for *jtos* and *utos*). This option is supported only for *jtos*, *stoj*, *stou*, and *utos*.

−w Map one Kanji space character to two ASCII space characters (**0x2020**) on the output. This option is supported only for *jtos* and *utos*.

WARNINGS

Only text files are converted properly by the *jtos*, *jtou*, *stoj*, *stou*, *utoj*, and *utos* commands.

SEE ALSO

iconv(1).

EXTERNAL INFLUENCES

International Code Set Support

Multi-byte-character code sets are supported with the exception of multi-byte-character file names.

NAME

kermit – kermit file transfer

SYNOPSIS

kermit [*options*] [*file*]

DESCRIPTION

Kermit is a file transfer program for moving files between many machines of different operating systems and architectures.

Arguments are optional. If *kermit* is executed without arguments, it enters command mode. Otherwise, *kermit* parses the arguments from the command line and interprets them.

The following terms are used in the command descriptions that follow:

fn	HP-UX file specification, possibly containing the "wildcard" characters '*' or '?' ('*' matches all character strings, '?' matches any single character).
fn1	HP-UX file specification which must not contain '*' or '?'.
rfn	Remote file specification in the remote system's own syntax. May denote a single file or a group of files.
rfn1	Remote file specification which should denote only a single file.
n	Decimal number between 0 and 94.
c	Decimal number between 0 and 127 representing the value of an ASCII character.
cc	Decimal number ranging from 0 through 31 or exactly 127, which represents the value of an ASCII control character.
[ ]	Any field in square braces is optional.
{*x* | *y* | *z*}	Alternatives are listed in curly braces.

Kermit command line options can specify either actions or settings. If *kermit* is invoked with a command line that specifies no actions, it will issue a prompt and begin interactive dialogue. Action options specify either protocol transactions or terminal connection.

The command line must contain no more than one protocol action option.

Action Options

–s *fn*

Send the specified file or files. If *fn* contains wildcard (meta) characters the HP-UX shell will expand it into a list. If *fn* is "–" then *kermit* sends from standard input, which must come from a file:

kermit –s – < foo.bar

or a pipeline:

ls -l | kermit –s –

This mechanism will not work to send terminal typein. If you want to send a file whose name is "–" you can precede it with a path name, as in

kermit –s ./–

–r

Receive a file or files. Wait passively for files to arrive.

–k

Receive (passively) a file or files, sending them to standard output. This option can be used in several ways:

kermit –k

Displays the incoming files on your terminal; to be used only in "local mode" (see below).

kermit −k > fn1

Sends the incoming file or files to the named file, *fn1*. If more than one file arrives, all are concatenated together into the single file *fn1*.

kermit −k | command

Pipes the incoming data (single or multiple files) to the indicated command, as in

kermit −k | sort > sorted.stuff

−a *fn1* If you have specified a file transfer option, you may specify an alternate name for a single file with the **−a** option. For example,

kermit −s foo −a bar

sends the file *foo* telling the receiver that its name is *bar*. If more than one file arrives or is sent, only the first file is affected by the **−a** option:

kermit −ra baz

stores the first incoming file under the name *baz*.

−x Begin server operation. May be used in either local or remote mode.

Setting Options

−l *dev* Line — Specify a terminal line to use for file transfer and terminal connection, as in

kermit −l /dev/tty1p4

The **−l** option places *kermit* in "local" mode (see below).

When an external line is being used, you might also need some additional options for successful communication with the remote system:

−b *n* Set Baud Rate — Specify the baud rate for the line given in the **−l** option, as in

kermit −l /dev/tty1p4 −b 9600

This option should always be included with the **−l** option, since the speed of an external line is not necessarily what you expect.

−p *x* Set Parity — **e**, **o**, **m**, **s**, **n** (even, odd, mark, space, or none). If parity is other than none, then the 8th-bit prefixing mechanism will be used for transferring 8-bit binary data, provided the opposite *kermit* agrees. The default parity is none.

−t Specifies half duplex, line turnaround with XON as the handshake character.

The following commands may be used only with a *kermit* which is local — either by default or because the **−l** option has been specified.

−g *rfn* Actively request a remote server to send the named file or files; *rfn* is a file specification in the remote host's own syntax. If *rfn* happens to contain any special shell characters, like '*', these must be quoted, as in:

kermit −g x*.\?

−f Send a "finish" command to a remote server.

−c Establish a terminal connection over the specified or default communication line, before any protocol transaction takes place. Control can be returned to the local system by typing the escape character (normally Control-Backslash) followed by the letter "c".

−n Like **−c**, but connect after a protocol transaction takes place. The use of **−n** and **−c** is illustrated below.

kermit −l /dev/tty0p4 −b 9600 −c

Causes *kermit* to connect through */dev/tty0p4* at 9600 baud.

kermit −l /dev/tty0p4 −b 9600 −rn

Causes *kermit* to wait for a file to be received, then connects through */dev/tty0p4* at 9600 baud.

On a timesharing system, the **−l** and **−b** options will also have to be included with the **−r**,**−k**, or **−s** options if the other *kermit* is on a remote system.

If *kermit* is in local mode, the standard output is continuously updated to show the progress of the file transfer. A dot (".") is printed for every four data packets, other packets are shown by type (e.g. 'S' for Send-Init), 'T' is printed when a timeout occurs , and '%' is printed for each retransmission. In addition, you may type certain "interrupt" commands during file transfer. These commands must be preceded by the escape character (by default, control-\).

Control-F: Interrupt the current File, and go on to the next (if any).

Control-B: Interrupt the entire Batch of files, terminate the transaction.

Control-R: Resend the current packet

Control-A: Display a status report for the current transaction.

Several other command-line options are provided:

−i Specifies that files should be sent or received exactly "as is" with no conversions. This option is necessary for transmitting binary files. It may also be used to slightly boost efficiency in HP-UX to HP-UX transfers of text files by eliminating CRLF/newline conversion.

−w Write-Protect — Avoid filename collisions for incoming files.

−q Quiet — Suppress screen update during file transfer, for instance to allow a file transfer to proceed in the background.

−d Debug — Record debugging information in the file debug.log in the current directory.

−h Help — Display a brief synopsis of the command line options.

Local vs. Remote Mode

Kermit is "local" if it is running on a PC or workstation that you are using directly, or if it is running on a multiuser system and transferring files over an external communication line — not your job's controlling terminal or console. *Kermit* is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line, connected to your PC or workstation.

If you are running *kermit* on a PC, it is in local mode by default, with the "back port" designated for file transfer and terminal connection. If you are running *kermit* on a multiuser (timesharing) system, it is in remote mode unless you explicitly point it at an external line for file transfer or terminal connection with the **−l** option.

Interactive operation

Kermit's interactive command prompt is "C-Kermit>". In response to this prompt, you may type any valid command. *Kermit* executes the command and then prompts you for another command. The process continues until you instruct the program to terminate.

Commands begin with a keyword, normally a verb, such as "send". You may omit trailing characters from any keyword, so long as you specify sufficient characters to distinguish it from

any other keyword valid in that field. Certain commonly-used keywords (such as "send", "receive", "connect") have special non-unique abbreviations ("s" for "send", "r" for "receive", "c" for "connect").

Certain characters have special functions in interactive commands:

? Question mark, typed at any point in a command, will produce a message explaining what is possible or expected at that point. Depending on the context, the message may be a brief phrase, a menu of keywords, or a list of files.

ESC (The Escape key) — Request completion of the current keyword or filename, or insertion of a default value. The result will be a beep if the requested operation fails due to multiple matches.

DEL, BS (The Delete or Rubout key, Backspace or control-H) — Delete the previous character from the command.

^W (Control-W) — Erase the rightmost word from the command line.

^U (Control-U) — Erase the entire command.

^R (Control-R) — Redisplay the current command.

SP, TAB (Space, Horizontal tab) — Delimits fields (keywords, filenames, numbers) within a command.

CR, LF, FF (Carriage Return, Linefeed, Formfeed) — Enters the command for execution.

**** (Backslash) — Enter any of the above characters into the command, literally. To enter a backslash, type two backslashes in a row (\\).

You may type the editing characters (DEL, ^W, etc) repeatedly, to delete all the way back to the prompt. No action will be performed until the command is entered by typing carriage return, linefeed, or formfeed. If you make any mistakes you will receive an informative error message and a new prompt will be displayed. Make liberal use of '?' and ESC to feel your way through the commands. One important command is **help** — you should use it the first time you run *kermit*.

Interactive *kermit* accepts commands from files as well as from the keyboard. When you enter interactive mode, *kermit* looks for the file **.kermrc** in your home or current directory (first it looks in the home directory, then in the current one) and executes any commands it finds there. These commands must be in interactive format, not HP-UX command-line format. A **take** command is also provided to execute commands from a file at any time during an interactive session. Command files may be nested to any reasonable depth.

Here is a brief list of *kermit* interactive commands:

! *command* Execute the HP-UX shell command *command*.

bye Terminate and log out a remote *kermit* server.

close *log_file* Close a log file. *Log_file* is one of the following: debugging, packets, session, or transactions.

connect Establish a terminal connection to a remote system as if it were a local terminal to that computer. The connection is made through the device specified in the most recent **set line** command. All characters you type at your keyboard are sent out the communication line, all characters arriving at the communication port are displayed on your terminal. Current settings of speed, parity, duplex, and flow control are honored. If you have issued a **log session** command, everything you see on your terminal will also be recorded to your session log. This provides a way to "capture" files from systems that don't have *kermit* programs available.

To get back to your own system, you must type the escape character, which is FS (control-\, ASCII 28) unless you have changed it with the **set escape** command, followed by a single character command, such as "c" for "close connection". The recognized single character commands are:

c	Close the connection.
b	Send a BREAK signal to remote machine.
0	(zero) Send a NUL character.
s	Give a status report about the connection.
^\	Send control-\ itself (whatever you have defined the escape character to be, typed twice in a row sends one copy of it).

cwd *dir* Change Working Directory. Changes the current working directory to *dir*.

dial *number* Dial a telephone number. Tells the modem to dial the number *num*. *Kermit* must be told what type of modem is being used (see **set** command below).

directory *dir* ... Display a directory listing.

echo *arg* ... Display arguments literally.

exit Exit from the program, closing any open logs.

finish Instruct a remote *kermit* server to exit, but not log out.

get [*file* [*dname*]]
Get files from a remote *kermit* server. *File* may contain wildcard characters. If *dname* is specified, the first incoming file will be stored under that name.

Since a remote file specification (or list) might contain spaces, which normally delimit fields of a *kermit* command, an alternate form of the command is provided to allow the inbound file to be given a new name: entering **get** alone on a line, and *kermit* will prompt separately for the remote and local file specifications. For example:

C-Kermit>*get*
Remote file specification: *foo*
Local name to store it under: *bar*

help [*topic*] Display a help message for a given command.

log *log_file file_name*
Open a log file with name *file_name*. *Log_file* must be one of the following: debugging, packet, session, or transaction.

quit Same as **exit**.

receive [*dname*]
Passively wait for files to arrive. If *dname* is specified, *kermit* will store the first incoming file under that name. The **receive** command may be abbreviated to **r**.

remote Issue file management commands to a remote *kermit* server. The valid *remote* commands are:

cwd *dir*	Change remote working directory.
delete *file* ...	Delete remote files.
directory *dir* ...	Display a listing of remote file names.

help [*topic*] Request help from a remote server.

host *command* Issue a command to the remote host in its own command language.

space Display current disk space usage on remote system.

type *file* Display a remote file on your screen.

who [*user*] Display who's logged in, or get information about a user.

script *text* Login to the remote system using the text provided. The login script is intended to operate similarly to *uucp* **Systems** entries. A login script is a sequence of the form:

expect send [*expect send*] ...

where *expect* is a prompt or message to be issued by the remote site, and *send* is the data to return to the remote host. *Send* may also be "EOT" to send control-d, or BREAK to send a break. Letters in *send* may be prefixed by "~" to send special characters. These are: ~b for backspace, ~s for space, ~q for "?", ~n for linefeed, ~r for return, ~c for don't append a return, and ~### (### are octal digits) for octal of a character. As with some *uucp* systems, sent strings are followed by ~r unless they end with ~c.

Only the last 7 characters in each expect are matched. A null expect, e.g. ~0 or two adjacent dashes, causes a short delay. If you expect that a sequence might not arrive, as with *uucp*, conditional sequences may be expressed in the form:

–send-expect[*–send-expect*[...]]

where dashed sequences are followed as long as previous expects fail.

send *file* [*dname*]

Send files. Send the file of files specified by *file* to the other *kermit*, which should be running as a server, or which should be given the **receive** command. Each file is sent under its own name (or converted as specified by the **set names** command). The **send** command may be abbreviated to **s** even though "s" is not a unique abbreviation for a top-level *kermit* command.

The wildcard characters "*" and "?" are accepted in *file*. If "?" is to be included, it must be prefixed by "\" to override its normal function of providing help. "*" matches any string, "?" matches any single character. Other notations for file groups, like "**[a–z]og**", are not available in interactive commands. When *file* contains wildcard characters, there is a limit to the number of files that can be matched, which varies from system to system. If you get the message "Too many files match" then you will have to make a more judicious selection.

Kermit does not skip over "invisible" files that match the file specification. HP-UX systems usually treat files whose names start with a dot (like **.login**, **.profile**, and **.kermrc**) as invisible.

If *dname* is specified, *file* must not contain any wildcard characters and *dname* specifies the name to send it under.

Note: *kermit* sends only from the current or specified directory. It does not traverse directory trees. If the source directory contains subdirectories, they will be skipped. Conversely, *kermit* does not create directories when receiving files. If you need to do this, you can pipe *tar*(1) through *kermit*. For example, on the origination system, type:

tar cf – /usr/src | kermit –is –

This causes *tar* to send the directory **/usr/src** (and all files in its subdirectories) to standard output. *Kermit* receives the standard input and sends it as a binary file. On the receiving system type:

kermit –il /dev/tty1p3 –b 9600 –k | tar xf –

This causes *kermit* to receive the *tar* archive and sends it via standard output to its own copy of *tar*, which extracts from it a replica of the original directory tree.

server Begin server operation. Places the local *kermit* into server mode. All further commands must arrive as valid *kermit* packets from the *kermit* on the other end of the line. The HP-UX *kermit* server can respond to the following commands:

get	Sends files.
send	Receives files.
bye	Attempts to log itself out.
finish	Exits the server mode.
remote directory	Sends directory listing.
remote delete	Removes files.
remote cwd	Changes working directory.
remote type	Sends files to your screen.
remote space	Reports about its disk usage.
remote who	Shows who's logged in.
remote host	Executes an HP-UX shell command.
remote help	Lists these capabilities.

set Set various parameters. The "set" parameters are:

block-check { **1** | **2** | **3** }
Level of packet error detection. **1** is a single character 6-bit checksum, folded to included the values of all bits from each character. **2** is a 2 character 12-bit checksum. 3 is a 3 character 16-bit cyclic redundancy check (CRC). The higher the block check, the better the error detection and correction and the higher the resulting overhead. Type **1** is most commonly used; it is supported by all *kermit* implementations, and has proven adequate in most circumstances. Types **2** or **3** would be used to advantage when transferring 8-bit binary files over noisy lines.

delay *n* How long to wait before sending first packet. *N* is specified in seconds.

duplex { **full** | **half** }
Specify which side echoes during "connect". **Full** means the other side, **half** means the local *kermit* must echo typein itself.

escape-character *cc*
Character to prefix "escape commands" during "connect". The escape character is also used to prefix interrupt commands

during file transfers. The default value is 28 (control backslash).

file *param* Set various file parameters. Valid values for *param* are:

display { **on** | **off** }
Normally **on**. When in local mode, display progress of file transfers to the standard output, and listen to the standard input for interruptions. If off (also settable by **−p** on the *kermit* invocation line), none of this is done.

names { **converted** | **literal** }
Normally **converted**, which means that outbound file names have path specifications stripped, lowercase letters upshifted, tildes and extra periods changed to "X"'s and an "X" inserted in front of any name that starts with period. Incoming files have uppercase letters downshifted. **Literal** means none of these conversions are done. When using **literal** naming, the sender should not use path names in the file specification unless the same path exists on the target system and is writable.

type { **binary** | **text** }
Normally **text**, which means that conversion is done between HP-UX newline characters and the carriage return/linefeed sequences required by the canonical *kermit* file transmission format, and in common use on non-HP-UX systems. **Binary** means to transmit file contents without conversion. **Binary** is necessary for binary file transmission and is desirable in all HP-UX to HP-UX transactions to cut down on overhead.

warning { **on** | **off** }
Normally **off**, which means that incoming files will silently overwrite existing files of the same name. When **on**, *kermit* will check of an arriving file would overwrite an existing file. If so, it will construct a new name for the arriving file of the form *file~n*, where *file* is the name they share and *n* is a generation number. For example if **foo** exists, then the new file will be called **foo~1**. If **foo** and **foo~1** exist, the new file will be **foo~2**, and so on.

flow-control { **none** | **xon/xoff** }
Communication line full-duplex flow control. Normally **xon/xoff** for full duplex flow control. Should be set to **none** if the other system cannot do XON/XOFF flow control.

handshake { **xon** | **xoff** | **cr** | **lf** | **bell** | **esc** | **none** }
Normally **none**. Otherwise half duplex communication line turnaround handshaking is done, which means *kermit* will not reply to a packet until it has received the indicated handshake character or has timed out waiting for it.

line [*device*] Communication line device name. If you specify a device name, *kermit* will be in local mode and you should remember to issue any other necessary **set** commands, such as **set speed**. If you omit the device name, *kermit* will revert to its default

mode of operation.

modem-dialer { **direct** | **hayes** | **ventel** }
Type of modem-dialer on communication line. **Direct** indicates either there is no dialout modem, or that if the line requires carrier detection to open, then **set line** will hang waiting for an incoming call. **Hayes** and **ventel** indicate that the subsequent **set line** will prepare for a subsequent **dial** command for Hayes and Ventel dialers, respectively. *kermit* does not need to have incoming packets preceded *kermit* to request the other *kermit* to

parity { **even** | **odd** | **mark** | **space** | **none** }
Communication line character parity. If other than **none**, *kermit* will seek to use the 8th bit prefixing mechanism for transferring 8-bit binary data, which can be used successfully only if the other *kermit* agrees.

prompt [*string*]
Change the *kermit* program's prompt. If *string* is given, the prompt will be set to *string*. If *string* is omitted, the prompt will revert to the default "C-Kermit>".

receive *parameter value*
Parameters to request or inspect for incoming packets, as follows:

end-of-line *cc*
Normally *carriage return* (15) by default.

packet-length *number*
Maximum length packet for the other side to send; a decimal number between 10 and 94. Shorter packet lengths can be used on noisy lines, or with systems, front ends, or networks that have small buffers. The shorter the packet, the higher the overhead, but the lower the chance of a packet being corrupted by noise, and the less time to retransmit the corrupted packets.

timeout *number*
How many seconds the other Kermit should wait for a packet before asking for retransmission.

pad-character *cc*
Character to use for inter-packet padding. Normally *kermit* does not need to have incoming packets preceded by pad characters. This command allows *kermit* to request the other *kermit* to use *cc* as a pad character. Default of NUL (ASCII 0). No *kermits* are known to need padding, and if one did, it would request it without your having to tell it to do so.

padding *number*
How many padding characters to request before each incoming packet.

start-of-packet *cc*
Set control character to mark beginning of incoming packets. The *kermit* packet prefix character is SOH

(control-A). The only reasons it would ever be changed would be: Some piece of equipment somewhere between the two *kermit* programs will not pass SOH, or some piece of equipment similarly placed is echoing its input. In the latter case, the recipient of such an echo can change the packet prefix for outbound packets, so that the echoed packets will be ignored. The opposite *kermit* must also be told to change the prefix for its inbound packets.

send *parameter value*

Establish paramters for outgoing packets. This command is generally used to override negotiated values, or to establish before negotiation takes place.

end-of-line *cc*

The ASCII character to be used as a line terminator for outbound packets, if one is required by the other system. Normally *carriage return* (15) by default. You will only have to use this command for systems that require a line terminator other than carriage return.

packet-lenght *number*

Maximum lenght packet to send, decimal number, between 10 and 94, decimal. Shorter packet lengths can be used on noisy lines, or with systems or front ends or networks that have small buffers. The shorter the packet, the higher the overhead, but the lower the chance of a packet being corrupted by noise, and the less time to retransmit the corrupted packets. This command can be used to specify a shorter lenght than the one requested by the other *kermit*, but not a longer one.

timeout *number*

How many seconds to wait for a packet before asking for trying again. A value of zero means do not time out, wait forever.

pad-character *cc*

Character to use for inter-packet padding. Default of NUL (ASCII 0).

padding *number*

How many padding characters to send before a packet. Defaults are no padding.

start-of-packet *cc*

Set control character to mark start of packets. The *kermit* packet prefix character is SOH (control-A). The only reasons it would ever be changed would be: Some piece of equipment somewhere between the two *kermit* programs will not pass SOH; or, some piece of equipment similarly placed is echoing its input. The opposite *kermit* must also be told to change the prefix for its inbound packets.

speed { **0** | **110** | **300** | **600** | **1200** | **1800** | **2400** | **4800** | **9600** }
Set communication line speed. This command cannot be used to change the speed of your own console terminal. Normally, you must use this command after a **set line** before you can use the line. *kermit* packet prefix character is SOH (control-A). *kermit* programs will not *kermit* must also be told to change the prefix for *kermit* presently can be told to change only its outbound *kermit* partner sets its packet timeout interval *kermit* requests. *kermit* will wait forever for expected packets to arrive.

show { **parameters** | **versions** }
If **show parameters** is entered, *kermit* will display the values of all the **set** parameters described above. If **show versions** is entered, *kermit* will display the version numbers and dates of all its interval modules.

space Display current disk space usage.

statistics Display statistics about most recent *kermit* protocol transaction including file and communication line I/O, as well as what encoding options were in effect (such as 8th-bit prefixing, repeat-count compression, etc.).

take *file* Execute commands from *file*. *File* may contain any interactive *kermit* commands, including **take**. Command files may be nested to any reasonable depth. The **echo** command may be used within command files to issue greetings, announce progress, etc.

Command files are in exactly the same syntax as interactive commands. Note that this implies that if you want to include special characters like "?" or "\" you have to quote these characters the same way as when typing interactive commands.

Command files may be used in lieu of command macros, which have not been implemented in this version of *kermit*. For example, if you commonly connect to a system called "B" that is connected to tty1p3 at 4800 baud, you could create a file called *b* that contains the commands:

```
set line /dev/tty1p3
set speed 4800
echo Connecting to System B...
connect
```

and then simply type "**take b**" (or "**t b**" since no other commands begin with the letter "t") whenever you wish to connect to system B.

An implicit **take** command is executed upon your **.kermrc** file upon *kermit*'s initial entry into interactive dialog. The **.kermrc** file should contain **set** or other commands you want to be in effect at all times. For instance, you might want to override the default action when incoming files have the same names as existing files — in that case, put the command

```
set file warning on
```

in your **.kermrc** file.

DIAGNOSTICS

The diagnostics produced by kermit itself are intended to be self-explanatory.

WARNINGS

File renaming: When filename collision avoidance (**"set file warning"**) is selected, *kermit* constructs unique names by appending a generation number to the end of the file name. Currently, no checking is done to ensure that the result is still within the maximum length for a file name.

UUCP line locking: *kermit* locks lines, to prevent *uucp* and multiuser conflicts, when it first opens a communications line. This occurs either when a **set line** is issued of if the **–l** argument is used, when the first **dial, connect**, or protocol operation occurs. The lock is released if another **set line** is issued, or if the program quits, exits, or is terminated by SIGINT. If a user connects and returns to the shell command level, for example to initiate *kermit* by piped commands, the line lock is released when returning to the shell. Locking is not needed, or used, if communication occur over the local terminal line (e.g. /dev/tty). In that case, there is no difficulty with piped operations releasing locks and lines.

Removing stale lock files: For various reasons, lock files sometimes get left after *uucp* or *kermit* activities. (The most common reason is that the *uucp* or *kermit* activity was killed by a shell command.) *Uucp* supports a function called *uuclean* which is customarily used to remove these files after a predetermined age. If in doubt about a lock file on the dial-out line you need, contact your system administrator.

Modem controls: If a connection is made over a communication line (rather than on the controlling terminal line), and that line has modem controls (e.g. data terminal ready and carrier detect implementations), returning to the shell will disconnect the conversation. In that case, one should use interactive mode commands, and avoid use of piped shell-level operation.

Resetting the terminal after abnormal termination of kill: When *kermit* terminates abnormally (say, for example, by a kill command issued by the operator) the user may need to reset the terminal state. If commands do not seem to be accepted at the shell prompt, try typing "^J reset ^J". That should take the terminal out of raw mode if it was stuck there.

AUTHOR

The *kermit* protocol was developed by Columbia University. *Kermit* is available for many systems for a nominal fee from Columbia and various user groups.

FILES

/usr/bin/kermit $HOME/.kermrc

SEE ALSO

cu(1), tar(1), uucp(1)

NAME

kill – terminate a process

SYNOPSIS

kill [*–signo* | *–signame*] *PID* ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This normally kills processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the Shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless the current user is the super-user.

If a signal number or signal name preceded by – is given as first argument, that signal is sent instead of terminate (see *signal*(5)). In particular "kill –9 ..." is a sure kill.

EXAMPLES

The command:

kill 6135

signals process number **6135** to terminate (assuming you own the process). This gives the process a chance to exit gracefully (removing temporary files, etc.).

The commands:

kill -9 6135

kill -SIGKILL 6135

kill -KILL 6135

terminate process number **6135** by sending a SIGKILL signal to the process (assuming you own the process). This tells the kernel to remove the process immediately.

SEE ALSO

ps(1), sh(1), kill(2), signal(5).

BUGS

If a process becomes hung during some operation (such as I/O) so that it is never scheduled, that process will not die until it is allowed to run. Thus, such a process may never go away after the kill.

STANDARDS CONFORMANCE

kill: SVID2, XPG2, XPG3

NAME

ksh, rksh − shell, the standard/restricted command programming language

SYNOPSIS

ksh [**−acefhikmnorstuvx**] [**−o** *option*] ... [*arg* ...]
rksh [**−acefhikmnorstuvx**] [**−o** *option*] ... [*arg* ...]

DESCRIPTION

Ksh is a command programming language that executes commands read from a terminal or a file. *Rksh* is a restricted version of the command interpreter *ksh*, used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Definitions

A *metacharacter* is one of the following characters:

; & ( ) | < > new-line space tab

A *blank* is a tab or a space. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *aliases*, *functions*, and *named parameters*. A *word* is a sequence of *characters* separated by one or more non-quoted *metacharacters*.

Commands

A *simple-command* is a sequence of blank-separated words that may be preceded by a parameter assignment list. (See *Environment* below). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(5) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by |. The standard output of each command but the last is connected by a pipe (see *pipe*(2)) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command to terminate.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or ||, and optionally terminated by **;**, **&**, or **|&**. Of these five symbols, **;**, **&**, and **|&** have equal precedence, which is lower than that of **&&** and ||. The symbols **&&** and || also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (that is, the shell does not wait for that pipeline to finish). The symbol **|&** causes asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and read from by the parent shell using the **−p** option of the special commands **read** and **print** described later. Only one such command can be active at any given time. The symbol **&&** (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines can appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *identifier* [**in** *word* ...] **do** *list* **done**

Each time **for** is executed, *identifier* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, **for** executes the **do** *list* once for each positional parameter set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

select *identifier* [**in** *word* ...] **do** *list* **done**
A **select** command prints on standard error (file descriptor 2), the set of *words*, each preceded by a number. If **in** *word* ... is omitted, the positional parameters are used instead (see *Parameter Substitution* below). The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed **words**, the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty, the selection list is printed again. Otherwise the value of the parameter *identifier* is set to **null**. The contents of the line read from standard input is saved in the parameter **REPLY**. The *list* is executed for each selection until a **break** or *end-of-file* is encountered.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is identical to that used for file name generation (see *File Name Generation* below).

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, **if** returns a zero exit status.

while *list* **do** *list* **done**
until *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list*, and if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, **while** returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list***)** Execute *list* in a separate environment. If two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below. A parenthesized list (for systems that support **/dev/fd**), used as a command argument, denotes *process substitution* as described below.

{ *list;*} Execute *list*, but not in a separate environment. Note that { is a keyword and requires a trailing blank to be recognized.

function *identifier* { *list* ;}
identifier **()** { *list* ;}
Define a function referred to by *identifier*. The body of the function is the *list* of commands between { and } (see *Functions* below).

time *pipeline* The *pipeline* is executed and the elapsed time, the user time, and the system time are printed on standard error.

The following keywords are recognized only as the first word of a command and when not quoted: **if then else elif fi case esac for while until do done { } function select time**

Comments
A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

Aliasing
The first word of each command is replaced by the text of an **alias**, if an **alias** for this word has been defined. The first character of an **alias** name can be any printable character, but the remaining characters must be identical to a valid *identifier*. The replacement string can contain

any valid shell script, including the metacharacters listed above. The first word of each command of the replaced text is not tested for additional aliases. If the last character of the alias value is a *blank*, the word following the alias is also checked for alias substitution. Aliases can be used to redefine special built-in commands, but cannot be used to redefine the keywords listed above. Aliases can be created, listed, and exported with the **alias** command and can be removed with the **unalias** command. Exported aliases remain in effect for subshells but must be reinitialized for separate invocations of the shell (see *Invocation* below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for it to take effect, **alias** must be executed before the command referring to the alias is read.

Aliases are frequently used as a shorthand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full path name of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the identifier is read and becomes undefined each time the **PATH** variable is reset. These aliases remain *tracked* so that the next reference will redefine the value. Several tracked aliases are compiled into the shell. The **−h** option of the **set** command converts each command name that is an *identifier* into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```
echo='print −'
false='let 0'
functions='typeset −f'
history='fc −l'
integer='typeset −i'
nohup='nohup '
pwd='print − $PWD'
r='fc −e −'
true=':'
type='whence −v'
hash='alias −t'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, the word up to a / is checked to see if it matches a user name in the **/etc/passwd** file. If a match is found, the ~ and the matched login name are replaced by the login directory of the matched user. This is called a tilde substitution. If no match is found, the original text is left unchanged. A ~, alone or before a /, is replaced by the value of the **HOME** parameter. A ~ followed by a + or − is replaced by the value of the parameter **PWD** and **OLDPWD**, respectively.

In addition, the value of each *keyword parameter* is checked to see if it begins with ~ or if ~ appears after :. In either of these cases a tilde substitution is attempted.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign (**$(**) or a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See *Quoting* below). The command substitution **$(cat file)** can be replaced by the equivalent but faster **$(<file)**. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

Process Substitution

This feature is available only on systems that support the **/dev/fd** directory for naming open files. Each command argument of the form **(***list***)**, **<(***list***)**, or **>(***list***)** will run process *list* asynchronously connected to some file in **/dev/fd**. The name of this file will become the argument

to the command. If the form with > is selected, writing on this file provides input for *list*. If < is used or omitted, the file passed as an argument will contain the output of the *list* process. For example,

paste (cut −f1 *file1***) (cut −f3** *file2***) | tee >(***process1***) >(***process2***)**

cuts fields 1 and 3 from the files *file1* and *file2* respectively, *pastes* the results together, and sends it to the processes *process1* and *process2*, as well as putting it onto the standard output. Because the file that is passed as an argument to the command is a pipe (see *pipe*(2)), programs that expect to use *lseek*(2) on the file will not work.

Parameter Substitution

A *parameter* is an *identifier*, one or more digits, or any of the characters *, @, #, ?, −, $, and !. A *named parameter* (a parameter denoted by an identifier) has a value and zero or more attributes. Named parameters can be assigned values and attributes by using the **typeset** special command. Attributes supported by *ksh* are described later with the **typeset** special command. Exported parameters pass values and attributes to subshells but only values to the environment.

The shell supports a limited one-dimensional array facility. An element of an array parameter is referenced by a subscript. A subscript is denoted by a **[**, followed by an arithmetic expression (see *Arithmetic Evaluation* below) followed by a **]**. The value of all subscripts must be in the range of **0** through **511**. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array is created if necessary. Referencing an array without a subscript is equivalent to referencing the first element.

The value of a named parameter may also be assigned by writing:

name=*value* [*name*=*value*] ...

If the **−i** integer attribute is set for *name*, the *value* is subject to arithmetic evaluation as described below.

Positional parameters, parameters denoted by a number, may be assigned values with the **set** special command. Parameter **$0** is set from argument zero when the shell is invoked.

The character **$** is used to introduce substitutable *parameters*.

${***parameter***}** Substitute the value of the parameter, if any. Braces are required when *parameter* is followed by a letter, digit, or underscore that should not be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits, it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is * or @, all the positional parameters, starting with **$1, are substituted (separated by a field separator character). If an array *identifier* with subscript * or @ is used, the value for each element is substituted (separated by a field separator character).

${#*parameter***}** If *parameter* is * or @, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

${#*identifier***[*]}**
Substitute the number of elements in the array *identifier*.

${*parameter***:−***word***}**
If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter***:=***word***}**
If *parameter* is not set or is null, set it to *word*; then substitute the value of the parameter. Positional parameters may not be assigned in this way.

${*parameter***:?***word***}**
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, a standard message is printed.

${*parameter***:+***word***}**
If *parameter* is set and is non-null, substitute *word*; otherwise substitute

nothing.

${*parameter***#***pattern***}**
${*parameter***##***pattern***}**
If the shell *pattern* matches the beginning of the value of *parameter*, the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise the value of this *parameter* is substituted. In the former case, the smallest matching pattern is deleted; in the latter case, the largest matching pattern is deleted.

${*parameter***%***pattern***}**
${*parameter***%%***pattern***}**
If the shell *pattern* matches the end of the value of *parameter*, the value of *parameter* with the matched part is deleted; otherwise substitute the value of *parameter*. In the former, the smallest matching pattern is deleted; in the latter, the largest matching pattern is deleted.

In the above, *word* is not evaluated unless it is used as the substituted string. Thus, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (**:**) is omitted from the above expressions, the shell only checks to determine whether or not *parameter* is set.

The following parameters are set automatically by the shell:

The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the **set** command.
? The decimal value returned by the last executed command.
$ The process number of this shell.
_ The last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail. Finally, the value of this parameter is set to the full path name of each program the shell invokes and is passed in the *environment*.
! The process number of the last background command invoked.
PPID The process number of the parent of the shell.
PWD The present working directory set by the **cd** command.
OLDPWD The previous working directory set by the **cd** command.
RANDOM Each time this parameter is evaluated, a random integer is generated. The sequence of random numbers can be initialized by assigning a numeric value to **RANDOM**.
REPLY This parameter is set by the **select** statement and by the **read** special command when no arguments are supplied.
SECONDS Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following parameters are used by the shell:

CDPATH The search path for the **cd** command.
COLUMNS If this variable is set, its value is used to define the width of the edit window for the shell edit modes and for printing **select** lists.
EDITOR If the value of this variable ends in **emacs**, **gmacs**, or **vi** and the **VISUAL** variable is not set, the corresponding option is turned on. (See **set** in *Special Commands* below.)

ENV If this parameter is set, parameter substitution is performed on the value to generate the path name of the script to be executed when the shell is invoked. (See *Invocation* below.) This file is typically used for *alias* and *function* definitions.

FCEDIT The default editor name for the **fc** command.

IFS Internal field separators, normally **space**, **tab**, and **new-line** that are used to separate command words that result from command or parameter substitution and for separating words with the special command **read**. The first character of the **IFS** parameter is used to separate arguments for the **"$*"** substitution (see *Quoting* below).

HISTFILE If this parameter is set when the shell is invoked, its value is the path name of the file that is used to store the command history. The default value is **$HOME/.sh_history**. If the user is super-user and no HISTFILE is given, then no history file is used. (See *Command Re-entry* below.)

HISTSIZE If this parameter is set when the shell is invoked, the number of previously entered commands accessible to this shell will be greater than or equal to this number. The default is **128**.

HOME The default argument (home directory) for the **cd** command.

LINES If this variable is set, the value is used to determine the column length for printing **select** lists. **Select** lists print vertically until about two-thirds of **LINES** lines are filled.

MAIL If this parameter is set to the name of a mail file and the **MAILPATH** parameter is not set, the shell informs the user of arrival of mail in the specified file.

MAILCHECK This variable specifies how often (in seconds) the shell checks for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is **600** seconds. When the time has elapsed the shell checks before issuing the next prompt.

MAILPATH A list of file names separated by colons (:). If this parameter is set, the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a **?** and a message to be printed, in which case the message will undergo parameter and command substitution with the parameter **$_** defined as the name of the changed file. The default message is *you have mail in $_*.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing *rksh* (except in the **.profile** file).

PS1 The value of this parameter is expanded for parameter substitution, to define the primary prompt string which, by default, is "**$** ". Replace the character **!** in the primary prompt string is by the *command* number (see *Command Re-entry* below).

PS2 Secondary prompt string, by default "> ".

PS3 Selection prompt string used within a **select** loop, by default "**#?** ".

SHELL The path name of the shell is kept in the environment. When invoked, the shell is restricted if the value of this variable contains an **r** in the basename.

TMOUT If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the **PS1** prompt. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)

VISUAL Invokes the corresponding option when the value of this variable ends in *emacs*, *gmacs*, or *vi*. (See **set** in *Special Commands* below.)

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, **TMOUT** and **IFS**. **HOME**, **SHELL**, **ENV**, and **MAIL** are never set automatically by the shell (although **HOME** is set by *login*(1)). On some systems, **MAIL** and **SHELL** are also set by *login*(1)).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for field separator characters (found in **IFS**), and split into distinct arguments where such characters are found. *Ksh* retains explicit null arguments (**""** or ´´) but removes implicit null arguments (those resulting from *parameters* that have no values).

File Name Generation

Following substitution, each command *word* is processed as a pattern for file name expansion unless the **−f** option has been **set**. The form of the patterns is the Pattern Matching Notation defined by *regexp*(5). The word is replaced with sorted file names matching the pattern. If no file name is found that matches the pattern, the word is left unchanged.

Quoting

Each of the *metacharacters* listed above (See *Definitions* above) has a special meaning to the shell and causes termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (´´), are quoted. A single quote cannot appear within single quotes. Inside double quote marks (**""**), parameter and command substitution occurs and \ quotes the characters \, `, **"**, and **$**. The meaning of **$***∗* and $@ is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, **"$***∗***"** is equivalent to **"$1***d***$2***d*...**"**, where *d* is the first character of the **IFS** parameter, whereas **$@** is equivalent to **"$1" "$2"** .... Inside grave quote marks (``) \ quotes the characters \, `, and **$**. If the grave quotes occur within double quotes, \ also quotes the character **"**.

The special meaning of keywords or aliases can be removed by quoting any character of the keyword. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

The ability to perform integer arithmetic is provided with the special command **let**. Evaluations are performed using long arithmetic. Constants take the form [*base***#**]*n*, where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted, base 10 is used.

An internal integer representation of a *named parameter* can be specified with the **−i** option of the **typeset** special command. When this attribute is selected the first parameter assignment parameter determines the arithmetic base used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the **let** command is provided. For any command beginning with **((**, all characters until the matching **))** are treated as a quoted expression. More precisely, **((** ... **))** is equivalent to **let "** ...**"**.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output can be redirected using a special notation interpreted by the shell. The following can appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used, except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<word Use file *word* as standard input (file descriptor **0**).

>*word*	Use file *word* as standard output (file descriptor **1**). If the file does not exist, it is created; otherwise, it is truncated to zero length.
>>*word*	Use file *word* as standard output. If the file exists, output is appended to it (by first searching for the end-of-file); otherwise, the file is created.
<<[−]*word*	The shell input is read up to a line that matches *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document. Otherwise, parameter and command substitution occurs, **new-line** is ignored, and \ must be used to quote the characters \, **$**, `, and the first character of *word*. If − is appended to <<, all leading tabs are stripped from *word* and from the document.
<&*digit*	The standard input is duplicated from file descriptor *digit* (see *dup*(2)).
>&*digit*	The standard output is duplicated to file descriptor *digit* (see *dup*(2)).
<&−	The standard input is closed.
>&−	The standard output is closed.

If one of the above is preceded by a digit, the file descriptor number cited is that specified by the digit (instead of the default **0** or **1**). For example:

... **2>&1**

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

Redirection order is significant. The shell evaluates each redirection in terms of the (*file descriptor, file*) assignment at the time of evaluation. For example:

... **1>***fname* **2>&1**

first assigns file descriptor 1 to file *fname*. It then assigns file descriptor 2 to the file assigned to file descriptor 1 (that is, *fname*). If the order of redirection were reversed, file descriptor 2 would be assigned to the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be assigned to file *fname*.

If a command is followed by **&** and job control is inactive, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment

The *environment* (see *environ*(5)) is a list of name-value pairs passed to an executed program much like a normal argument list. The names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. When invoked, the shell scans the environment and creates a parameter for each name found, gives it the corresponding value and marks it *export*. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones by using the **export** or **typeset** **−x** commands, the values become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in **export** or **typeset** **−x** commands.

The environment for any *simple-command* or function may be augmented by prefixing it with one or more parameter assignments. A parameter assignment argument takes the form *identifier=value*. For example,

TERM=450 cmd args
and
(export TERM; TERM=450; cmd args)

are equivalent (as far as the above execution of *cmd* is concerned).

If the **−k** flag is set, all parameter assignment arguments are placed in the environment, even if they occur after the command name. The following echo statement prints **a=b c**. After **−k** flag is set, the second echo statement prints only **c**:

```
echo a=b c
set −k
echo a=b c
```

Functions

The **function** keyword (described in the *Commands* section above) is used to define shell functions. Shell functions are read and stored internally. Alias names are resolved when the function is read. Functions are executed like commands, with the arguments passed as positional parameters. (See *Execution* below.)

Functions execute in the same process as the caller and share all files, traps (other than **EXIT** and **ERR**) and present working directory with the caller. A trap set on **EXIT** inside a function is executed after the function completes. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the **−f** option of the **typeset** special command. The text of functions will also be listed. Function can be undefined with the **−f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **−xf** option of the **typeset** command allows a function to be exported to scripts that are executed without reinvoking the shell. Functions that must be defined across separate invocations of the shell should be placed in the **ENV** file.

Jobs

If the **monitor** option of the **set** command is turned on, an interactive shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the **jobs** command, and assigns them small integer numbers. When a job is started asynchronously with **&**, the shell prints a line that looks like:

[1] 1234

indicating job number 1 was started asynchronously and had one (top-level) process whose process ID was 1234.

If you are running a job and wish to do something else, you can type **^Z** (control-Z) to send a STOP signal to the current job. The shell then indicates that the job has been 'Stopped', and print another prompt. The user can then manipulate the state of this job by putting it in the background with the **bg** command, running other commands and eventually returning the job to the foreground with the **fg** command. A **^Z** takes effect immediately and resembles an interrupt, since pending output and unread input are discarded when **^Z** is typed.

A job run in the background stops if it tries to read from the terminal. Background jobs normally are allowed to produce output, but can be disabled by giving the **stty tostop** command. If the user sets this tty option, background jobs stop when trying to produce output.

There are several ways to refer to jobs in the shell. The character % introduces a job name. If you wish to refer to job number 1, type **%1**. Jobs are also named by prefixes of the string typed in to **kill** or restart them. Thus, **fg %ed** normally restarts a suspended *ed*(1) job, if the name of the suspended job begins with the string **ed**.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a −. The abbreviation **%+** refers to the current job and **%−** refers to the previous job. **%%** is also a synonym for the current job.

This shell learns immediately when a process changes state. It informs the user when a job is blocked and prevented from further progress, but only just before it prints a prompt.

If you try to leave the shell while jobs are running or stopped, you are warned, "You have stopped (running) jobs." You may use the **jobs** command to identify them. If you immediately try to exit again, the shell will not warn you a second time, and the stopped jobs will be terminated.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by **&** and job **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Substitutions are made each time a command is executed. If the command name matches one of the *Special Commands* listed below, it is executed within the current shell process. Next, *ksh* checks the command name to determine whether it matches one of the user-defined functions. If it does, *ksh* saves the positional parameters and resets to the arguments of the *function* call. When the *function* completes or issues a **return**, *ksh* restores the positional parameter list and executes any trap set on **EXIT** within the function. The value of a *function* is the value of the last command executed. A function is also executed in the current shell process. If a command name is not a *special command* or a user-defined *function*, *ksh* creates a process and attempts to execute the command using *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **/bin:/usr/bin:** (specifying **/bin**, **/usr/bin**, and the current directory in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between colon delimiters, or at the end of the path list. The search path is not used if the command name contains a /. Otherwise, each directory in the path is searched for an executable file. If the file has execute permissions but is not a directory or an executable object code file, it is assumed to be a script file, which is a file of data for an interpreter. If the first two characters of the script file are "#!", *exec*(2) expects an interpreter path name to follow. *Exec*(2) then attempts to execute the specified interpreter as a separate process to read the entire script file. If a call to *exec*(2) fails, **/bin/ksh** is spawned to interpret the script file. All non-exported aliases, functions, and named parameters are removed in this case. A parenthesized command is also executed in a subshell. If the shell command file does not have read permission, or if the *setuid* and/or *setgid* bits are set on the file, the shell executes an agent to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is also executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file **$HOME/.sh_history** is used if the **HISTFILE** variable is not set or writable. A shell can access the commands of all *interactive* shells that use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not

specify an editor program as an argument to **fc**, the value of the parameter **FCEDIT** is used. If **FCEDIT** is not defined, **/bin/ed** is used. The edited command is printed and re-executed upon leaving the editor. The editor name – is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if **r** is aliased to **'fc -e -'**, typing '**r bad=good c**' re-executes the most recent command that starts with the letter **c** and replaces the first occurrence of the string **bad** with the string **good**.

In-line Editing Options

Normally, each command line typed at a terminal device is followed by a new-line ('RETURN' or 'LINE FEED'). If either the **emacs**, **gmacs**, or **vi** option is set, the user can edit the command line. An editing option is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept 'RETURN' as carriage return without line feed and that a space (' ') must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to 'space'. Hewlett-Packard terminal users should set the straps to 'bcGHxZ etX'.

The editing modes enable the user to look through a window at the current line. The default window width is 80, unless the value of **COLUMNS** is defined. If the line is longer than the window width minus two, a mark displayed at the end of the window notifies the user. The mark is a > (<, *) if the line extends on the right (left, both) side(s) of the window. As the cursor moves and reaches the window boundaries, the window is centered about the cursor.

Emacs Editing Mode

This mode is invoked by either the **emacs** or **gmacs** option. Their sole difference is their handling of ^**T**. To edit, the user moves the cursor to the point needing correction and inserts or deletes characters or words. All editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, ^**F** is the notation for control-**F**. This is entered by depressing 'f' while holding down the 'CTRL' (control) key. The 'SHIFT' key is *not* depressed. (The notation ^**?** indicates the DEL (delete) key.)

The notation for escape sequences is **M-** followed by a character. For example, **M-f** (pronounced Meta f) is entered by depressing ESC (ASCII **033**) followed by 'f'. (**M-F** would be the notation for ESC followed by 'SHIFT' (capital) 'F'.)

All edit commands operate from any place on the line (not only at the beginning). Neither the "RETURN" nor the "LINE FEED" key is entered after edit commands, except when noted.

^**F**	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^**B**	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^**A**	Move cursor to start of line.
^**E**	Move cursor to end of line.
^**]***char*	Move cursor to character *char* on current line.
^**X**^**X**	Interchange the cursor and mark.
erase	(User defined erase character as defined by the *stty*(1) command, usually ^**H** or **#**.) Delete previous character.
^**D**	Delete current character.
eof	End-of-file character, normally ^**D**, will terminate the shell if the current line is null.
M-d	Delete current word.
M-^**H**	(Meta-backspace) Delete previous word.

M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is **^?** (DEL, the default) this command will not work).
^T	Transpose current character with next character in **emacs** mode. Transpose two previous characters in **gmacs** mode.
^C	Capitalize current character.
M-C	Capitalize current word.
M-l	Change the current word to lowercase.
^K	Kill from the cursor to the end of the line. If given a parameter of zero, kill from the start of line to the cursor.
^W	Kill from the cursor to the mark.
M-p	Push the region from the cursor to the mark on the stack.
kill	(User-defined kill character, as defined by the *stty*(1) command, usually **^G** or @.) Kill the entire current line. If two *kill* characters are entered in succession, all subsequent consecutive kill characters cause a line feed (useful when using paper terminals).
^Y	Restore last item removed from line. (Yank item back to the line.)
^L	Line feed and print current line.
^@	(Null character) Set mark.
M-	(Meta space) Set mark.
^J	(New line) Execute the current line.
^M	(Return) Execute the current line.
^P	Fetch previous command. Each time **^P** is entered, the previous command in the history list is accessed.
^N	Fetch next command. Each time **^N** is entered the next command in the history list is accessed.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.
^R*string*	Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *String* is terminated by a "RETURN" or "NEW LINE". If *string* is omitted, the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.
^O	Operate – Execute the current line and fetch the next line relative to current line from the history file.
M-*digits*	(Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are **.**, **^F**, **^B**, *erase*, **^D**, **^K**, **^R**, **^P**, **^N**, **M-.**, **M-_**, **M-b**, **M-c**, **M-d**, **M-f**, **M-h** and **M-^H**.
M-*letter*	Soft-key – Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value is inserted on the input queue. The *letter* must not be one of the above meta-functions.
M-.	The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.
M-_	Same as **M-.**.
M-*	Attempt file name generation on the current word.
M-ESC	Attempt file name completion on the current word.
M-=	List files matching current word pattern as if an asterisk were appended.
^U	Multiply parameter of next command by 4.
\	Escape next character. Editing characters, the user's erase, kill and interrupt (normally **^?**) characters may be entered in a command line or in a search string if preceded by a \. The \ removes the next character's editing features (if any).

^V Display version of the shell.

Vi Editing Mode

There are two typing modes. Entering a command puts you into *input* mode. To edit, the user enters *control* mode by typing ESC (**033**) and moves the cursor to the point needing correction, then inserts or deletes characters or words. Most control commands accept an optional repeat *count* prior to the command.

In vi mode on most systems, canonical processing is initially enabled and the command is echoed again if the speed is 1200 baud or greater and contains any control characters, or if less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

Setting the **viraw** option will always disable canonical processing on the terminal. This mode is implicit for systems that do not support two alternate end-of-line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase	Delete previous character. (*Erase* is a user-defined erase character, as defined by the *stty*(1) command, usually ^**H** or **#**.)
^**W**	Delete the previous blank separated word.
^**D**	Terminate the shell.
^**V**	Escape next character. Editing characters, erase or kill characters may be entered in a command line or in a search string if preceded by a ^**V**. ^**V** removes the next character's editing features (if any).
\	Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands move the cursor. The designation [*count*] causes a repetition of the command the cited number of times.

[*count*]**l**	Cursor forward (right) one character.
[*count*]**w**	Cursor forward one alphanumeric word.
[*count*]**W**	Cursor to the beginning of the next word that follows a blank.
[*count*]**e**	Cursor to end of word.
[*count*]**E**	Cursor to end of the current blank-delimited word.
[*count*]**h**	Cursor backward (left) one character.
[*count*]**b**	Cursor backward one word.
[*count*]**B**	Cursor to preceding blank separated word.
[*count*]**f***c*	Find the next character *c* in the current line.
[*count*]**F***c*	Find the previous character *c* in the current line.
[*count*]**t***c*	Equivalent to **f** followed by **h**.
[*count*]**T***c*	Equivalent to **F** followed by **l**.
;	Repeats the last single character find command, **f**, **F**, **t**, or **T**.
,	Reverses the last single character find command.
0	Cursor to start of line.
^	Cursor to first nonblank character in line.
$	Cursor to end of line.

Search Edit Commands

These commands access your command history.

[*count*]**k**	Fetch previous command. Each time **k** is entered, the next earlier command in the history list is accessed.

[*count*]–	Equivalent to **k**.
[*count*]**j**	Fetch next command. Each time **j** is entered, the next later command in the history list is accessed.
[*count*]+	Equivalent to **j**.
[*count*]**G**	The command number *count* is fetched. The default is the first command in the history list.
/string	Search backward through history for a previous command containing *string*. *String* is terminated by a "RETURN" or "NEW-LINE". If *string* is null, the previous string is used.
?*string*	Same as **/** but search in the forward direction.
n	Search for next match of the last pattern to **/** or **?** commands.
N	Search for next match of the last pattern to **/** or **?**, but in reverse direction. Search history for the *string* entered by the previous **/** command.

Text Modification Edit Commands

These commands will modify the line.

a	Enter input mode and enter text after the current character.
A	Append text to the end of the line. Equivalent to **$a**.
[*count*]**c***motion*	
c[*count*]*motion*	Move cursor to the character position specified by *motion*, deleting all characters between the original cursor position and new position, and enter input mode. If *motion* is **c**, the entire line is deleted and input mode entered.
C	Delete the current character through the end of line and enter input mode. Equivalent to **c$**.
S	Equivalent to **cc**.
D	Delete the current character through the end of line. Equivalent to **d$**.
[*count*]**d***motion*	
d[*count*]*motion*	Move cursor to the character position specified by *motion*, deleting all characters between the original cursor position and new position. Equivalent to **d$**. If *motion* is **d**, the entire line will be deleted.
i	Enter input mode and insert text before the current character.
I	Insert text before the beginning of the line. Equivalent to the two character sequence **^i**.
[*count*]**P**	Place the previous text modification before the cursor.
[*count*]**p**	Place the previous text modification after the cursor.
R	Enter input mode and replace characters on the screen with characters you type overlay fashion.
r*c*	Replace the current character with *c*.
[*count*]**x**	Delete current character.
[*count*]**X**	Delete preceding character.
[*count*].	Repeat the previous text modification command.
~	Invert the case of the current character and advance the cursor.
[*count*]_	Causes the *count* word of the previous command to be appended at the current cursor location and places the editor in input mode at the end of the appended text. The last word is used if *count* is omitted.
*	Appends an * to the current word and attempts file name generation. If no match is found, the bell rings. If a match is found, the word is replaced by the matching string and the command places the editor in input mode.
ESC	Attempt file name completion on the current word.

Other Edit Commands

Miscellaneous commands.

[*count*]**y***motion*

y[*count*]*motion*	Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.
Y	Yanks from current position to end of line. Equivalent to **y$**.
u	Undo the last text modifying command.
U	Undo all the text modifying commands performed on the line.
[*count*]**v**	Returns the command **fc −e ${VISUAL:−${EDITOR:−vi}}** *count* in the input buffer. If *count* is omitted, the current line is used.
^L	Line feed and print current line. Has effect only in control mode.
^J	(New line) Execute the current line, regardless of mode.
^M	(Return) Execute the current line, regardless of mode.
#	Equivalent to **I#**<*cr*>. Sends the line after inserting a **#** in front of the line and after each new-line. Useful for inserting the current command line in the history list without executing it.
=	List the filenames that match the current word if an asterisk were appended to it.
@*letter*	The user's alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value is inserted on the input queue for processing.

Special Commands

The following simple-commands are executed in the shell process. They permit input/output redirection. File descriptor 1 is the default output location. Commands that are preceded by † or †† are treated specially in the following ways:

1. Parameter assignment lists preceding the command remain in effect when the command completes.
2. They are executed in a separate process when used within command substitution.
3. Errors in commands preceded by †† cause the script that contains them to abort.

† : [*arg* **...]** The command only expands parameters. A zero exit code is returned.

†† . *file* **[** *arg* **...]**
Read and execute commands from *file* and return. The commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged.

alias [**−tx**] [*name*[*=value*] ...]
Alias with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **−t** flag is used to set and list tracked aliases. The value of a tracked alias is the full path name corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the **−t** flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **−x** flag is used to set or print exported aliases. An exported alias is defined across sub-shell environments. Alias returns true unless a *name* is given for which no alias has been defined.

bg [*%job* **]** Puts the specified *job* into the background. The current job is put in the background if *job* is unspecified.

break [*n*] Exit from the enclosing **for while until** or **select** loop, if any. If *n* is specified, break *n* levels.

continue [*n*] Resume the next iteration of the enclosing **for while until** or **select** loop. If *n* is specified, resume at the *n*-th enclosing loop.

† **cd** [*arg*]
† **cd** *old new* This command can take either of two forms. In the first form it changes the current directory to *arg*. If *arg* is − the directory is changed to the previous directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). If **CDPATH** is null or undefined, the default value is the current directory. Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, the search path is not used. Otherwise, each directory in the path is searched for *arg*.

The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.

The **cd** command will not be executed by *rksh*.

echo [*arg* ...] See *echo*(1) for usage and description.

†† **eval** [*arg* ...]
Reads the arguments as input to the shell and executes the resulting command(s).

†† **exec** [*arg* ...]
Parameter assignments remain in effect after the command completes. If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given, the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 opened with this mechanism are closed when invoking another program.

exit [*n*] Causes the shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end-of-file also causes the shell to exit, except when a shell has the *ignoreeof* option set. (See **set** below.)

†† **export** [*name* ...]
The given *name*s are marked for automatic export to the *environment* of subsequently executed commands.

†† **fc** [−**e** *ename*] [−**nlr**] [*first* [*last*]]
†† **fc** −**e** − [*old=new*] [*command*]
In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands typed at the terminal. The arguments *first* and *last* may be specified as a number or string. A given string is used to locate the most recent command. A negative number is used to offset the current command number. The flag −**l** causes the commands to be listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, the value of the parameter **FCEDIT** (default **/bin/ed**) is used as the editor. Once editing has ended, the commands (if any) are executed. If *last* is omitted, only the command specified by *first* will be used. If *first* is not specified, the default is the previous command for editing and −16 for listing. The flag −**r** reverses the order of the commands and the flag −**n** suppresses command numbers when listing. In the latter, the *command* is re-executed after the substitution *old=new* is performed.

fg [%*job*] If *job* is specified it brings it to the foreground. Otherwise, the current job is brought into the foreground.

jobs [−**l**] Lists the active jobs; the −**l** option lists process IDs also.

kill [−*sig*] *process* ...
Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are given either by number or name (as given in *signal*(5), stripped of the prefix "SIG"). The signal names are listed by **kill** −**l**. No default exists; merely typing **kill** does not affect the current job. If the signal being sent is **TERM** (terminate) or **HUP** (hangup), the job or process will be sent a **CONT** (continue) signal when stopped. The *process* argument can be either a process ID or job.

let *arg* ... Each *arg* is an arithmetic *expression* to be evaluated. All calculations are done as long integers and no check for overflow is performed. Expressions consist of constants, named parameters, and operators. The following set of operators, listed in order of decreasing precedence, have been implemented:

−	unary minus
!	logical negation
* / %	multiplication, division, remainder
+ −	addition, subtraction
<= >= < >	comparison
== !=	equality inequality
=	arithmetic replacement

Sub-expressions in parentheses **()** are evaluated first and can be used to override the above precedence rules. The evaluation within a precedence group is from right to left for the = operator and from left to right for the others.

A parameter name must be a valid *identifier*. When a parameter is encountered, the value associated with the parameter name is substituted and expression evaluation resumes. Up to nine levels of recursion are permitted.

The return code is **0** if the value of the last expression is non-zero, and **1** otherwise.

†† **newgrp** [*arg* ...]
Equivalent to **exec newgrp** *arg* ....

print [−**Rnprsu**[*n*]] [*arg* ...]
The shell output mechanism. With no flags or with flag −, the arguments are printed on standard output as described by *echo*(1). Raw mode, −**R** or −**r**, ignores the escape conventions of *echo*. The −**R** option will print all subsequent arguments and options other than −**n**. The −**p** option causes the arguments to be written onto the pipe of the process spawned with **|&** instead of standard output. The −**s** option causes the arguments to be written onto the history file instead of standard output. The −**u** flag can be used to specify a one digit file descriptor unit number **n** on which the output will be placed. The default is 1. If the flag −**n** is used, no **new-line** is added to the output.

pwd Equivalent to **print** −**r** − **$PWD**

read [−**prsu**[*n*]] [*name*?*prompt*] [*name* ...]
The shell input mechanism. One line is read and is broken up into words using the characters in **IFS** as separators. In −**r** raw mode, \ at the end of a line does not signify line continuation. The first word is assigned to the first *name*, the second word to the second *name*, etc., with remaining words

assigned to the last *name*. The **−p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **−s** flag is present, the input will be saved as a command in the history file. The flag **−u** can be used to specify a one-digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is **0**. If *name* is omitted, **REPLY** is used as the default *name*. The return code is **0**, unless an end-of-file is encountered. An end-of-file with the **−p** option causes cleanup for this process so that another process can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* when the shell is interactive. If the given file descriptor is open for writing and is a terminal device, the prompt is placed on this unit. Otherwise the prompt is issued on file descriptor 2. The return code is **0**, unless an end-of-file is encountered.

†† **readonly** [*name* ...]
The given *names* are marked read only and these names cannot be changed by subsequent assignment.

†† **return** [*n*] Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted, the return status is that of the last command executed. If **return** is invoked while not in a *function*, it has the same effect as an **exit** command.

set [**−aefhkmnopstuvx** | **+aefhkmnopstuvx** | **−o** *option* | **+o** *option*] ... [*arg* ...]
The following flags are used for this command:

−a All subsequent defined parameters are automatically exported.

−e If the shell is non-interactive and if a command fails, execute the **ERR** trap, if set, and exit immediately. This mode is disabled while reading profiles.

−f Disables file name generation.

−h Each command whose name is an *identifier* becomes a tracked alias when first encountered.

−k All parameter assignment arguments (not just those that precede the command name) are placed in the environment for a command.

−m Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. This flag is turned on automatically for interactive shells.

−n Read commands but do not execute them. The **−n** option is ignored for interactive shells.

−o The **−o** argument takes any of several *option* names, but only one *option* can be specified with each **−o** flag. If none is supplied, the current option settings are printed. The **−o** argument *option* names follow:

allexport Same as **−a**.

bgnice All background jobs are run at a lower priority.

errexit Same as **−e**.

emacs Puts you in an *emacs* style in-line editor for command entry.

gmacs Puts you in a *gmacs* style in-line editor for command entry.

ignoreeof The shell will not exit on end-of-file. The command **exit** must be used.

keyword Same as **−k**.
markdirs All directory names resulting from file name generation have a trailing / appended.
monitor Same as **−m**.
noexec Same as **−n**.
noglob Same as **−f**.
nounset Same as **−u**.
protected Same as **−p**.
verbose Same as **−v**.
trackall Same as **−h**.
vi Puts you in insert mode of a *vi* style in-line editor until you hit escape character **033**. This puts you in move mode. A return sends the line.
viraw Each character is processed as it is typed in *vi* mode.
xtrace Same as **−x**.

−p Resets the **PATH** variable to the default value, disables processing of the **$HOME/.profile** file and uses the file **/etc/suid_profile** instead of the **ENV** file. This mode is automatically enabled whenever the effective uid (gid) is not equal to the real uid (gid).
−s Sort the positional parameters.
−t Exit after reading and executing one command.
−u Treat unset parameters as an error when substituting.
−v Print shell input lines as they are read.
−x Print commands and their arguments as they are executed.
− Turns off **−x** and **−v** flags and stops examining arguments for flags.
−− Do not change any of the flags; useful in setting **$1** to a value beginning with **−**. If no arguments follow this flag, the positional parameters are unset.

Using + instead of − before a flag causes the flag to be turned off. These flags can also be used when invoking the shell. The current set of flags can be examined by using **$−**.

The remaining *arg* arguments are positional parameters and are assigned consecutively to **$1**, **$2**, ... If neither arguments nor flags are given, the values of all names are printed on the standard output.

† **shift** [*n*] The positional parameters from **$***n*+**1** ... are renamed **$1** ... ; default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

test [*expr*] Evaluate conditional expression *expr*. See *test*(1) for usage and description. The arithmetic comparison operators are not restricted to integers. They allow any arithmetic expression. Four additional primitive expressions are allowed:

−L *file* True if *file* is a symbolic link.
file1 **−nt** *file2* True if *file1* is newer than *file2*.
file1 **−ot** *file2* True if *file1* is older than *file2*.
file1 **−ef** *file2* True if *file1* has the same device and i-node number as *file2*.

times Print the accumulated user and system times for the shell and for processes run from the shell.

trap [*arg*] [*sig*] ...
The *arg* is a command read and executed when the shell receives signal(s) *sig*.

(Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or name of the signal. Trap commands are executed signal number order. Any attempt to set a trap on a signal that was ignored upon entering the current shell is ineffective. If *arg* is omitted or is −, all traps for *sig* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *sig* is **ERR**, *arg* will be executed whenever a command has a non-zero exit code. This trap is not inherited by functions. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

†† **typeset** [−**LRZefilprtux**[*n*]] [*name*[=*value*]] ...

†† **typeset** [+**LRZefilprtux**[*n*]] [*name*[=*value*]] ...

Parameter assignments remain in effect after the command completes. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

−**L** Left justify and remove leading blanks from *value*. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. When the *name* is assigned, the value is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the −**Z** flag is also set. The −**R** flag is turned off.

−**R** Right justify and fill with leading blanks. If *n* is non-zero it defines the width of the field, otherwise it is determined by the width of the value of first assignment. The field is left-filled with blanks or truncated from the end if the parameter is reassigned. The **-L** flag is turned off.

−**Z** Right justify and fill with leading zeros if the first non-blank character is a digit and the −**L** flag has not been set. If *n* is non-zero it defines the width of the field; otherwise it is determined by the width of the value of first assignment.

−**e** Tag the parameter as having an error. This tag is currently unused by the shell and can be set or cleared by the user.

−**f** Cause *name* to refer to function names rather than parameter names. No assignments can be made to the *name* declared with the **typeset** statement. The only other valid flags are −**t** (which turns on execution tracing for this function) and −**x** (which allows the function to remain in effect across shell procedures executed in the same process environment).

−**i** Parameter is an integer. This makes arithmetic faster. If *n* is non-zero it defines the output arithmetic base; otherwise the first assignment determines the output base.

−**l** Convert all uppercase characters to lowercase. The uppercase −**u** flag is turned off.

−**p** Write the output of this command, if any, onto the two-way pipe.

−**r** Any given *name* is marked "read only" and cannot be changed by subsequent assignment.

−**t** Tag the named parameters. Tags are user definable and have no special meaning to the shell.

−u Convert all lowercase characters to uppercase characters. The lowercase **−l** flag is turned off.

−x Mark any given *name* for automatic export to the environment of subsequently executed commands.

Using + instead of − causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of names (and optionally the values) of the parameters that have these flags set is printed. Using + instead of − retains the values to be printed. If neither names nor flags are given, the names and attributes of all parameters are printed.

ulimit [**−acdfmpst**] [*n*]

−a List all of the current resource limits (BSD only).

−c Impose a size limit of *n* 512 byte blocks on the size of core dumps (BSD only).

−d Impose a size limit of *n* kbytes on the size of the data area (BSD only).

−f Impose a size limit of *n* 512 byte blocks on files written by child processes (files of any size may be read).

−m Impose a soft limit of *n* kbytes on the size of physical memory (BSD only).

−p Change the pipe size to *n* (UNIX/RT only).

−s Impose a size limit of *n* kbytes on the size of the stack area (BSD only).

−t Impose a time limit of *n* seconds to be used by each process (BSD only).

If no option is given, **−f** is assumed. If *n* is not given, the current limit is printed.

umask [*nnn*] The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

unalias *name* ...

The parameters given by the list of *name*s are removed from the *alias* list.

unset [**−f**] *name* ...

The parameters given by the list of *name*s are unassigned; that is, their values and attributes are erased. Read-only variables cannot be unset. If the **−f** flag is set, names refer to function names.

wait [*n*] Wait for the specified process to terminate or stop, and report its status. This status becomes the return code for the **wait** command. If *n* is not given, **wait** waits for all currently active child processes to terminate or stop. The termination status returned is that of the last process.

whence [**−v**] *name* ...

For each *name*, indicate how it would be interpreted if used as a command name.

The **−v** flag produces a more verbose report.

Invocation of *ksh*

If the shell is invoked by *exec*(2), and the first character of argument zero (**$0**) is −, the shell is assumed to be a login shell and commands are read first from **/etc/profile**, then from either **.profile** in the current directory or **$HOME/.profile**, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV**, if the file exists. If the **−s** flag is not present and *arg* is, a path search is performed on the first *arg* to determine the name of the script to execute. When running *ksh* with

arg, the script *arg* must have read permission and any *setuid* and *getgid* settings will be ignored. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked:

−c *string*	If the **−c** flag is present, commands are read from *string*.
−s	If the **−s** flag is present or if no arguments remain, commands are read from the standard input. Shell output, except for the output of some of the *Special commands* listed above, is written to file descriptor 2.
−i	If the **−i** flag is present or if the shell input and output are attached to a terminal (as reported by *tty*(3C)), the shell is interactive. In this case SIGTERM is ignored (so that **kill 0** does not kill an interactive shell) and SIGINT is caught and ignored (so that **wait** is interruptible). In all cases, SIGQUIT is ignored by the shell. (See *signal*(5).)
−r	If the **−r** flag is present, the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rksh Only

Rksh is used to set up login names and execution environments where capabilities are more controlled than those of the standard shell. The actions of *rksh* are identical to those of *ksh*, except that the following are forbidden:

changing directory (see *cd*(1))
setting the value of **SHELL**, **ENV**, or **PATH**
specifying path or command names containing **/**
redirecting output (> and >>)

The restrictions above are enforced after **.profile** and the **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, *rksh* invokes *ksh* to execute it. Thus, the end-user is provided with shell procedures accessible to the full power of the standard shell, while being restricted to a limited menu of commands. This scheme assumes that the end-user does not have write and execute permissions in the same directory.

These rules effectively give the writer of the **.profile** file complete control over user actions, by performing guaranteed set-up actions and leaving the user in an appropriate directory (probably not the login directory).

The system administrator often sets up a directory of commands (usually **/usr/rbin**) that can be safely invoked by *rksh*. HP-UX systems provide a restricted editor *red* (see *ed*(1)), suitable for restricted users.

RETURN VALUE

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (also see the **exit** command above). If the shell is being used non-interactively, execution of the shell file is abandoned. Runtime errors detected by the shell are reported by printing the command or function name and the error condition. If the line number on which the error occurred is greater than one, the line number is also printed in brackets (**[]**) after the command or function name.

WARNINGS

If a command which is a *tracked alias* is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to load and execute the original command. Use the **−t** option of the **alias** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Some very old shell scripts contain a caret (^) as a synonym for the pipe character (|). Note however, *ksh* does not recognize the caret as a pipe character.

If a command is piped into a shell command, all variables set in the shell command are lost when the command completes.

Using the **fc** built-in command within a compound command will cause the entire command to disappear from the history file.

The built-in command **.** *file* reads the entire file before any commands are executed. Therefore, **alias** and **unalias** commands in the file will not apply to any functions defined in the file.

The **export** built-in command does not handle arrays properly. Only the first element of an array is exported to the *environment*.

Background processes started from a non-interactive shell cannot be accessed by using job control commands.

In an international environment, character ordering is determined by the setting of LC_COLLATE, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particulary when using range expressions in file name generation patterns. For example, the command,

rm [a-z]*

might be expected to match all file names beginning with a lowercase alphabetic character. However, if dictionary ordering is specified by LC_COLLATE, it would also match file names beginning with an uppercase character (as well as those beginning with accented letters). Conversely, it would fail to match letters collated after 'z' in languages such as Danish or Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form:

rm [[:lower:]]*

This uses LC_CTYPE to determine character classes and works predictably for all supported languages and codesets. For shell scripts produced on non-internationalized systems (or without consideration for the above dangers), it is recommended that they be executed in a non-NLS environment. This requires that LANG, LC_COLLATE, etc., be set to "C" or not set at all.

Be aware that the value of the IFS variable in the user's environment affects the behavior of scripts.

AUTHOR

Ksh was developed by AT&T.

FILES

/etc/passwd	to find home directories
/etc/profile	read to set up system environment
/etc/suid_profile	security profile
$HOME/.profile	read to set up user's custom environment
/tmp/sh*	for here-documents

SEE ALSO

cat(1), cd(1), echo(1), env(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), gtty(2), pipe(2), signal(5), umask(2), ulimit(2), wait(2), rand(3C), a.out(4), profile(4), environ(5), lang(5), regexp(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the classification of characters as letters, and the characters matched by character class expressions in pattern matching notation.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ksh* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

NAME

lastcomm – show last commands executed in reverse order

SYNOPSIS

lastcomm [command name] ... [user name] ... [terminal name] ...

DESCRIPTION

Lastcomm gives information on previously executed commands. With no arguments, *lastcomm* prints information about all the commands recorded during the current accounting file's lifetime. If called with arguments, only accounting entries with a matching command name, user name, or terminal name are printed. So, for example,

lastcomm a.out root ttyd0

would produce a listing of all the executions of commands named *a.out* by user *root* on the terminal *ttyd0*.

For each process entry, the following are printed.

The name of the user who ran the process.
Flags, as accumulated by the accounting facilities in the system.
The command name under which the process was called.
The amount of cpu time used by the process (in seconds).
The time the process started.

The flags are encoded as follows: "S" indicates the command was executed by the super-user, "F" indicates the command ran after a fork, but without a following *exec*, "D" indicates the command terminated with the generation of a *core* file, and "X" indicates the command was terminated with the signal SIGTERM.

AUTHOR

Lastcomm was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

last(1), acct(4), core(4).

NAME

ld − link editor

SYNOPSIS

ld [−**dmnqrstvxzNQZ**] [−**e** *epsym*] [−**h** *symbol*] ... [−**o** *outfile*] [−**u** *symbol*] ... [−**A** *name*] [−**L** *dir*] ... [−**R** *offset*] [−**V** *num*] [−**X** *num*] [−**l***x* | *file*] ...

DESCRIPTION

Ld takes one or more object files as input and combines them to produce a single (usually executable) file. In doing so it resolves references to external symbols, assigns final addresses to procedures and variables, revises code and data to reflect new addresses (a process called "relocation"), and updates symbolic debug information (when present in the file). By default, *ld* processes one or more object files to produce an executable file that can be run by the HP-UX loader *exec*(2). Alternatively, the linker can generate a relocatable file, which is suitable for further processing by *ld* (see −**r** below). *Ld* will not generate an output file if any errors occur during its operation.

Ld recognizes two kinds of input files: object files created by the compilers or assembler (also known as '.o' files) and archives of such object files (called libraries). A library contains an index of all the externally-visible symbols from its component object files. (The archiver command *ar*(1) creates and maintains this index.) *Ld* uses this table to resolve references to external symbols.

Ld processes files in the same order as they appear on the command line. It includes code and data from a library element if and only if that object module provides a definition for a currently unresolved reference within the user's program. It is common practice to list libraries following the names of all simple object files on the command line.

Options

Ld recognizes the following options:

−**d** Forces definition of "common" storage, i.e., assign addresses and sizes, even for −**r** output.

−**e** *epsym* Set the default entry point address for the output file to be that of the symbol *epsym*. (This option only applies to executable files.)

−**h** *symbol* Prior to writing the symbol table to the output file, mark this name as "local" so that it is no longer externally visible. This ensures that this particular entry will not clash with a definition in another file during future processing by *ld*. (Of course, this only makes sense with the −**r** option.) More than one *symbol* can be specified, but −**h** must precede each one.

−**l***x* Search a library **lib***x***.a,** where *x* is one or more characters. Because a library is searched when its name is encountered, the placement of a −**l** is significant. By default, libraries are located in **/lib** and **/usr/lib**. If the environment variable **LPATH** is present in the user's environment, it should contain a colon-separated list of directories to search. These directories are searched instead of the default directories, but −**L** options can still be used.

−**m** Produce a load map on the standard output.

−**n** Generate an (executable) output file with code to be shared by all users. Compare with −**N**.

−**o** *outfile* Produce an output object file by the name *outfile*. (The default name is **a.out**.)

−**q** Generate an (executable) output file that is demand-loadable. Compare with −**Q**.

−r Retain relocation information in the output file for subsequent re-linking. *Ld* will not report undefined symbols.

−s Strip the output file of all symbol table, relocation, and debug support information. This might impair or prevent the use of a symbolic debugger on the resulting program. This option is incompatible with **−r**. (The *strip*(1) command also removes this information.)

−t Print a trace (to standard output) of each input file as *ld* processes it.

−u *symbol* Enter *symbol* as an undefined symbol in the symbol table. The resulting unresolved reference is useful for linking a program solely from object files in a library. More than one *symbol* can be specified, but each must be preceded by **−u**.

−v Display verbose messages during linking. For each library module that is loaded, the linker indicates which symbol caused that module to be loaded.

−x Partially strip the output file; that is, leave out local symbols. The intention is to reduce the size of the output file without impairing the effectiveness of object file utilities. Note: use of **−x** might affect the use of a debugger.

−z Arrange for run-time dereferencing of null pointers to produce a SIGSEGV signal. (This is the complement of the **−Z** option.)

-A *name* This option specifies incremental loading, that is, linking to enable the resulting object to be read into an already executing program. The argument *name* specifies a file whose symbol table provides the basis for defining additional symbols. Only newly linked material is entered into the text and data portions of **a.out**, but the new symbol table reflects all symbols defined before and after the incremental load. Also, the **−R** option can be used in conjunction with **−A**, and allows the newly linked segment to commence at the corresponding address. The default starting address is the old value of **_end**.

-L *dir* Change the algorithm of searching for **lib***x***.a** to look in *dir* before looking in the default places. More than one directory can be specified, but each must be preceded by **−L**. The **−L** option is effective only if it precedes the **−l** option on the command line.

−N Generate an (executable) output file that cannot be shared. This option also causes the data to be placed immediately following the text, and the text to be made writable.

−Q Generate an (executable) output file that is not demand-loadable. (This is the complement of the **−q** option.)

−R *offset* Set the origin (in hexadecimal) for the text (i.e. code) segment.

−V *num* Use *num* as a decimal version stamp identifying the **a.out** file that is produced. (This is not the same as the version information reported by the SCCS *what*(1) command.)

−X *num* Define the initial size for the linker's global symbol table. Thus you can reduce link time for very large programs, i.e., those with very many external symbols.

−Z Arrange for run-time dereferencing of null pointers to be permitted. (See in **cc**(1) the discussions of **−Z** and *pointers*.) (This is the complement of the **−z** option.)

Defaults

Unless otherwise directed, *ld* names its output **a.out**. The **−o** option overrides this. Executable output files can be shared.

DIAGNOSTICS

Ld returns a zero when the link is successful. A non-zero return code indicates that an error occurred.

EXAMPLES

The following command line links part of a C program for later processing by *ld*. It also specifies a version number of 2 for the output file. (Note the '.o' suffix for the output object file. This is an HP-UX convention for indicating a linkable object file.)

```
ld -V 2 -r file1.o file2.o -o prog.o
```

The next example links a simple FORTRAN program for use with the *cdb*(1) symbolic debugger. The output file name will be **a.out** since there is no **−o** option in the command line. (Note: the particular options shown here are for a Series 300.)

```
ld -e start /lib/frt0.o ftn.o -lI77 -lF77 -lm -lc /usr/lib/end.o
```

Finally, this command will link a Pascal program on a Series 300.

```
ld /lib/crt0.o main.o -lpc -lm -lc
```

WARNINGS

Ld recognizes several names as having special meanings. The symbol **_end** is reserved by the linker to refer to the first address beyond the end of the program's address space. Similarly, the symbol **_edata** refers to the first address beyond the initialized data, and the symbol **_etext** refers to the first address beyond the program text. The linker treats a user definition of any of these symbols as an error. The symbols **end**, **edata**, and **etext** are also defined by the linker, but only if the program contains a reference to these symbols and does not define them. (See *end*(3C) for details.)

Through its options, the link editor gives users great flexibility; however, those who invoke the linker directly must assume some added responsibilities. Input options should ensure the following properties for programs:

- When the link editor is called through *cc*(1), a start-up routine is linked with the user's program. This routine calls *exit*(2) after execution of the main program. If users call *ld* directly, they must ensure that the program always calls *exit*() rather than falling through the end of the entry routine.
- When linking for use with the symbolic debugger *cdb*, the user must ensure that the program contains a routine called *main*. Also, the user must link in the file **/usr/lib/end.o** as the last file named on the command line.

There is no guarantee that the linker will pick up files from libraries and include them in the final program in the same relative order that they occur within the library.

DEPENDENCIES

Series 300

The default entry point is taken to be text location 0x0 (which is also the default origin of the program text). This corresponds to the first procedure in the first input file that the linker reads. Use the **−e** option to select a different entry point.

The version number specified with the **−V** option must be in the range **0** to **32767**.

The Series 300 linker does not support the following options: **−m**, **−z**, and **−Z**.

On the Series 300, the compilers precede all external names with an underscore. Thus, the symbol **_end** appears to the linker as **__end**.

Series 800

The linker searches for the symbol **$START$** as the program entry point. This symbol is defined in the file **/lib/crt0.o**, which should be the first file loaded for all programs, regardless of source language. Use the **−e** option to select a different entry point.

When invoking *ld* directly to link a C program whose **main** procedure is located in a library, the **−u main** option should be used to force the linker to load **main** from the library, since this symbol is not actually referenced until the **_start** routine is loaded from the C library. If you use *cc*(1) to link the program, the compiler will automatically pass this option to the linker. Because of this "side effect," you should not use *cc* to link a program containing a FORTRAN or Pascal main program; use *f77* or *pc* instead.

Nonsharable, executable files generated with the **−N** option cannot be executed via *exec*(2). Typically, **−N** is used when rebuilding the kernel or when preparing an image for dynamic loading.

When the **−A** option is used to do an incremental link, the linker generates extra code where a procedure call crosses a quadrant boundary. (A quadrant is one gigabyte, or one fourth of the addressing space.) On the Series 800, text is normally in the first quadrant and data is in the second quadrant. When an object file is intended to be read into an already-executing program, both its code and data must be placed in the second quadrant, since the first quadrant is set to read-only. Procedure calls from one quadrant to the other require the extra code, called inter-space calling stubs. The linker generates an "export" stub for the entry point designated in the incremental link, and "import" stubs for each procedure in the basis program that is called by the new object file. The import stubs require the existence of a routine in the basis program called **_sr4export**, which is supplied in **/lib/crt0.o**.

The Series 800 linker does not support the **−V** option.

The following options are specific to the Series 800 linker:

-y *symbol* Indicate each file in which *symbol* appears. Many such options can be given to trace many symbols, but **−y** must precede each one.

-C*n* Set the maximum parameter checking level to *n*. The default maximum is 3. See the language manuals for the meanings of the parameter checking level.

-D *offset* Set the origin (in hexadecimal) for the data space. The default value for *offset* is 0x40000000.

-G Strip all unloadable data from the output file. This option is typically used to strip debug information.

-S Generate an Initial Program Loader (IPL) auxiliary header for the output file, instead of the default HP-UX auxiliary header.

-T Save the load data and relocation information in temporary files instead of memory during linking. This option reduces the virtual memory requirements of the linker. If the **TMPDIR** environment variable is set, the temporary files are created in the specified directory, rather than in **/tmp**.

AUTHOR

Ld was developed by AT&T and HP.

FILES

/lib/libx.a	libraries
/usr/lib/libx.a	libraries
a.out	output file

Series 300

/lib/crt0.o	run-time start-up for C and Pascal
/lib/mcrt0.o	run-time start-up for C and Pascal with profiling (see *prof*(1)
/lib/gcrt0.o	run-time start-up for C and Pascal with profiling (see *gprof*(1)
/lib/frt0.o	run-time start-up for FORTRAN
/lib/mfrt0.o	run-time start-up for FORTRAN with profiling (see *prof*(1)
/lib/gfrt0.o	run-time start-up for FORTRAN with profiling (see *gprof*(1)
/usr/lib/end.o	for use with *cdb*/*fdb*/*pdb*(1)

Series 800

/lib/crt0.o	run-time start-up
/lib/mcrt0.o	run-time start-up with profiling (see *prof*(1)
/lib/gcrt0.o	run-time start-up with profiling (see *gprof*(1)
/usr/lib/xdbend.o	for use with *xdb*(1)
/usr/lib/nls/$LANG/ld.cat	message catalog
/tmp/ld*	temporary files

SEE ALSO

ar(1), cc(1), cdb(1), f77(1), gprof(1), nm(1), pc(1), prof(1), strip(1), exec(2), end(3C), a.out(4), ar(4).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *ld* behaves as if all internationalization variables are set to "C". See *environ*(5).

STANDARDS CONFORMANCE

ld: SVID2, XPG2

NAME

leave – remind you when you have to leave

SYNOPSIS

leave [*hhmm*]

DESCRIPTION

Leave waits until the specified time, then reminds you to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. When you log off, *leave* exits.

The time of day is in the form *hhmm*, where *hh* is a time in hours (on a 12- or 24-hour clock). All times are converted to a 12-hour clock and assumed to be in the next 12 hours.

If no argument is given, *leave* prompts with "When do you have to leave?" A reply of newline causes *leave* to exit; otherwise the reply is assumed to be a time. This form is suitable for inclusion in a **.login** or **.profile.**

Leave ignores interrupts, quits, and terminate signals. To get rid of it you should either log off or use "kill –9" giving its process ID.

EXAMPLES

The command

leave 1204

sends an alarm (a beep) to your terminal to remind you that you have to leave at 12:04 and reminds you that you are late at one minute intervals after 12:04.

WARNINGS

Leave checks to see if a user has logged out by checking the **/etc/utmp** file every 100 seconds. If a user logs out and logs back in to the same tty before *leave* makes its periodic check, *leave* might not know that the user has logged out.

FILES

/etc/utmp

AUTHOR

Leave was developed by the University of California, Berkeley.

SEE ALSO

calendar(1).

NAME

lex − generate programs for lexical analysis of text

SYNOPSIS

lex [**−rctvn**] [**−X**<*secondary*><*n*> ...] [*file*] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text.

The input *files* contain strings and expressions to be searched for, and C text to be executed when strings are found. Multiple files are treated as a single file. If no files are specified, the standard input is used.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx−z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators ∗, +, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character **.** is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation *r*{*d*,*e*} in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than |, but lower than ∗, ?, +, and concatenation. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character **$** at the end of an expression requires a trailing new-line. The character **/** in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \. Thus **[a−zA−Z]**+ matches a string of letters.

Three subroutines defined as macros are expected: **input()** to read a character; **unput(***c***)** to replace a character read; and **output(***c***)** to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named **yylex()**, and the library contains a **main()** which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function **yymore()** accumulates additional characters into the same *yytext*; and the function **yyless(***p***)** pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext*+*yyleng*. The macros *input* and *output* use files **yyin** and **yyout** to read from and write to, defaulted to **stdin** and **stdout**, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the **lex.yy.c** file. All rules should follow a %%, as in *yacc*(1). Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with { }. Note that curly brackets do not imply parentheses; only string substitution is done.

The flags, which must appear before any *files*, are as follows:

−r	indicates *ratfor*(1) actions;
−c	indicates C actions − this is the default;
−t	causes the **lex.yy.c** program to be written instead to the standard output;
−v	provides a one-line summary of statistics for the machine generated;
−n	suppresses printing of the − summary.

The **−X**<*secondary*><*n*> option allows the sizes of certain internal *lex* tables to be reset. *Secondary* is one of the letters from the set { **d D s S a c** } and specifies the table; *n* is the new size.

Tables whose size can be changed by using secondary letters are:

d	table of definitions; default = 200.
D	table of characters in definition strings; default = 5000.
s	table of start conditions; default = 50.
S	table of characters in start condition names; default = 500.
c	array table for storing character classes; default = 1000.
a	right context/action array table; default = 100.

If an array overflows, *lex* issues a fatal error message including a suggestion of which table to reset. For example:

Definitions too long, try −XD option

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n*	number of positions is *n* (default is 2500);
%q *n*	number of positions for one state is *n* (default is 300);
%n *n*	number of states is *n* (default is 500);
%e *n*	number of parse tree nodes is *n* (default is 1000);
%a *n*	number of transitions is *n* (default is 2000).
%k *n*	number of packed character classes is *n* (default is 1000);
%o *n*	size of output array is *n* (default is 3000);

The use of one or more of the preceding table options automatically implies **−v**, unless **−n** is specified.

External names generated by *lex* all begin with the prefix **yy** or **YY**.

EXAMPLES

```
D       [0−9]
%%
if      printf("IF statement\n");
[a−z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"    {       loop:
                while (input() != '*');
                switch (input())
                        {
                        case '/': break;
                        case '*': unput('*');
                        default: goto loop;
                        }
                }
```

WARNINGS

The **−r** option is not yet fully operational.

The token buffer in the program built by *lex* is of fixed length,

yytext[YYLMAX]

where YYLMAX is defined to be 200 characters. Overflow of this array is not detected in the *lex.yy.c* program.

SEE ALSO

yacc(1), malloc(3X).
LEX – Lexical Analyzer Generator, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

lex: SVID2, XPG2, XPG3

NAME

lifcp – copy to or from LIF files

SYNOPSIS

lifcp [**–Txxx**] [**–Lxxx**] [**–vxxx**] [**–a**] [**–b**] [**–ixxx**] [**–r**] [**–t**] file1 file2
lifcp [**–Txxx**] [**–Lxxx**] [**–vxxx**] [**–a**] [**–b**] [**–ixxx**] [**–r**] [**–t**] file1 [file2 ...] directory

DESCRIPTION

Lifcp copies a LIF file to an HP-UX file, an HP-UX file to a LIF file, or a LIF file to another LIF file. It also copies a list of (HP-UX/LIF) files to a (LIF/HP-UX) directory. The last name on the argument list is the destination file or directory.

Options can be used singly or combined in any order before the file names. The space between option and argument is optional.

–T*xxx* Used only when copying files to a LIF volume. This option forces the file type of the LIF directory entry to be set to the argument given. The argument can be decimal, octal or hex, using standard "C" notation.

–L*xxx* Used only when copying files to a LIF volume. This option will set the "last volume flag" to *xxx* (0 or 1). The default "last volume flag" is **1**.

–v*xxx* Used only when copying files to a LIF volume. This option sets the "volume number" to *xxx*. The default "volume number" is one.

–a This option forces a ASCII mode of copying regardless of the file type. When copying in ASCII mode from HP-UX to LIF the default file type is BINARY (1). (For details on available modes of copying refer to *lif*(4)). This option has no effect when copying from LIF to LIF.

–b This option forces a BINARY mode of copying regardless of the file type. When copying in BINARY mode from HP-UX to LIF the default file type is BINARY (–2). (For details on available modes of copying refer to *lif*(4)). This option has no effect when copying from LIF to LIF.

–i*xxx* Used only when copying files to a LIF volume. This option sets the "implementation" field of the LIF directory entry to the argument given. The argument value can be decimal, octal or hex, using standard "C" notation. The "implementation" field can only be set for file types –2001 to –100000 (octal). The "implementation" field is set to zero for all interchange file types and for file types –2 to –200 (octal).

–r Forces RAW mode copying regardless of file type. When copying in RAW mode from HP-UX to LIF the default file type is BIN (–23951). **–T** option overrides the default file type. (various modes of copying are explained in *lif*(4).) –r option has no effect in LIF to LIF copy operations.

–t causes HP-UX file names to be translated to a name acceptable by a LIF utility; that is, all lowercase letters are converted to uppercase and all other characters except numerics are changed to an underscore (_). If the HP-UX file name starts with a non-letter, the file name is preceded by the capital letter X. Thus, for example, if two files named colon (:) and semicolon (;), were copied, both of them would be translated to X_. File names are truncated to a maximum of 10 characters. When copying a LIF file to an HP-UX or LIF file, **–t** has no effect.

Omitting **−t** causes an error to be generated if an improper name is used.

The default copying modes when copying from LIF to HP-UX are summarized in the following table:

File Type	Default Copying Mode
ASCII	ASCII
BINARY	BINARY
BIN	RAW
other	RAW

When copying from HP-UX to LIF, the default copying mode is ASCII and an ASCII file is created.

When copying from LIF to LIF, if no options are specified then all the LIF directory fields and file contents source to destination.

A LIF file name is recognized by the embedded colon (:) delimiter (see *lif*(4) for LIF file naming conventions). A LIF directory is recognized by a trailing colon. If an HP-UX file name containing a colon is used, the colon must be escaped with two backslash characters (\\) (the shell removes one of them).

The file name − (dash) is interpreted to mean standard input or standard output, depending on its position in the argument list. This is particularly useful if the data requires non-standard translation. When copying from standard input, if no other name can be found, the name "STDIN" is used.

LIF file naming conventions are known only to the LIF utilities. Since file name expansion is done by the shell, this mechanism cannot be used for expanding LIF file names.

Note that the media should *not* be mounted while using *lifcp*.

DEPENDENCIES

Series 800

The following option is also supported:

−K*nnn* forces each file copied in to begin on a *nnn* × 1024 byte boundary from the beginning of the volume. This is useful when files are used for Series 800 boot media. This option has no effect when copying from a LIF volume.

EXAMPLES

lifcp abc lifvol:CDE

Copy HP-UX file **abc** to LIF file **CDE** on LIF volume **lifvol** which is actually an HP-UX file initialized to be a LIF volume.

lifcp −t ∗ ../lifvol:

Copy all the HP-UX files in the current directory to the LIF volume **lifvol** which is present in the parent directory. File names are translated to appropriate LIF file names.

lifcp −r −T −5555 −t ∗.o lifvol:

Copy all the HP-UX object files in the current directory to the LIF volume *lifvol*. Copying mode is RAW and LIF file types are set to −5555.

lifcp −b ∗.o lifvol:

Copy all the object files in the current directory to the LIF volume **lifvol**. Copying mode is BINARY and LIF BINARY files are created.

lifcp −r −t ∗ /lifvol:

Copy all files in the current directory to the LIF volume **lifvol** in the **root** directory. Copying mode is RAW and LIF file types are set to BIN.

lifcp abc\\: lifvol:CDE
Copy file **abc:** to LIF file **CDE** in **lifvol**.

lifcp −t abc def lifvol:
Copy files **abc** and **def** to LIF files **ABC** and **DEF** within **lifvol**.

lifcp lifvol:ABC .
Copy LIF file **ABC** within **lifvol** to file **ABC** within current directory.

lifcp − /dev/dsk/1s2:A_FILE
Copy standard input to LIF file **A_FILE** on LIF volume **/dev/dsk/1s2**.

lifcp lifvol:ABC /dev/dsk/1s2:CDE
Copy LIF file **ABC** in **lifvol** to LIF file **CDE on /dev/dsk/1s2 .**

pr abc | lifcp − lifvol:ABC
Copy the output of *pr* to the LIF file **ABC**.

pr abc | lifcp − lifvol:
Copy the output of *pr* to the LIF volume **lifvol**. LIF file **STDIN** is created since no files names are specified.

lifcp lifvol:ABC −
Copy LIF file **ABC** in **lifvol** to *standard output*.

lifcp * ../lifvol:
Copy all files in current directory to LIF files of the same name on LIF volume **lifvol** (may cause errors if file names in the current directory do not obey LIF naming conventions!).

AUTHOR

Lifcp was developed by the Hewlett-Packard Company.

SEE ALSO

lifinit(1), lifls(1), lifrename(1), lifrm(1), lif(4).

DIAGNOSTICS

Lifcp returns exit code 0 if the file is copied successfully. Otherwise it prints a diagnostic and returns non-zero.

NAME

lifinit – write LIF volume header on file

SYNOPSIS

lifinit [–**v**nnn] [–**d**nnn] [–**n** string] file

DESCRIPTION

Lifinit writes a LIF volume header on a volume or file. *Option*s may appear in any order. Their meanings are:

–**v***nnn* Sets the volume size to *nnn* bytes. If *nnn* is not a multiple of 256, it will be rounded down to the next such multiple.

–**d***nnn* Sets the directory size to *nnn* file entries. If *nnn* is not a multiple of 8, it will be rounded up to next such multiple.

–**n** *string* sets the volume name to be *string*. If the –**n** option is not specified, the volume name is set to the last component of the path name specified by *file*. A legal LIF volume name is 6 characters long and is limited to uppercase letters (A-Z), digits (0-9) and the underscore character (_). The first character (if any) must be a letter. The utility will automatically perform translation to create legal LIF volume names. Therefore, all lowercase letters are up-shifted and all other characters except numeric and underscore will be replaced with capital letter (X). If the volume name does not start with a letter, the volume name will be preceded by the capital letter (X). The volume name will also be right padded with blanks or truncated as needed to be 6 characters long. If **-n** is used with no *string*, the default volume name is set to 6 blanks.

If *file* does not exist, a regular HP-UX disk file is created and initialized.

The default values for volume size are 256K bytes for regular files, and the actual capacity of the device for device files.

The default directory size is a function of the volume size. A percentage of the volume size is allocated to the volume directory as follows:

VOLUME SIZE	DIRECTORY SIZE
< 2MB	~1.3%
> 2MB	~0.5%

Each directory entry occupies 32 bytes of storage. The actual directory space is subject to the rounding rules stated above.

Note that you should **not** mount the special file before using *lifinit*.

DEPENDENCIES

Series 300

If your media has never been initialized, it must be initialized using *mediainit*(1) before *lifinit* can be used. (Refer to the HP-UX System Administrator Manual for details concerning *mediainit*.)

Series 800

The following options are also supported:

–**s***nnn* set the initial system load **(ISL)** start address to nnn in the volume label. This is useful when building boot media for Series 800 systems.

–**l***nnn* specifies the length in bytes of the **ISL** code in the **LIF** volume.

−e*nnn*	set the **ISL** blocksize to nnn bytes.
−K*nnn*	forces the directory start location to be the nearest multiple of nnn * 1024 bytes from the beginning of the volume. This is necessary for booting Series 800 systems off of **LIF** media.

EXAMPLES

lifinit −v500000 −d10 x
lifinit /dev/rdsk/1s2

AUTHOR

Lifinit was developed by the Hewlett-Packard Company.

SEE ALSO

lifcp(1), lifls(1), lifrename(1), lifrm(1), lif(4).

DIAGNOSTICS

Lifinit returns exit code 0 if the volume is initialized successfully. Otherwise it prints a diagnostic and returns non-zero.

WARNING

Do not terminate *lifinit* once it has started executing. Otherwise, your media could become corrupted.

NAME

lifls – list contents of a LIF directory

SYNOPSIS

lifls [option] name

DESCRIPTION

Lifls lists the contents of a LIF directory on STDOUT. The default output format calls for the file names to be listed in multiple columns (as is done by *ls*(1), except unsorted) if STDOUT is a character special file. If STDOUT is not a teletype, the output format is one file name per line. *Name* is a path name to an HP-UX file containing a LIF volume and optional file name. If *name* is a volume name, the entire volume is listed. If *name* is of the form *volume:file*, then only the file is listed. The following options are available and only one option should be specified at any one time:

–l List in long format, giving volume name, volume size, directory start, directory size, file type, file size, file start, "implementation" field (in hex), date created, last volume and volume number.

–C Force multiple column output format regardless of STDOUT type.

–L Will return the content of the "last volume flag" in decimal.

–i Will return the content of the "implementation" field in hex.

–v Will return the content of the "volume number" in decimal.

Note that you should **not** mount the special file before using *lifls*.

EXAMPLES

lifls -l ../TEST/header
lifls -C /dev/rdsk/1s2

AUTHOR

Lifls was developed by the Hewlett-Packard Company.

SEE ALSO

lifcp(1), lifinit(1), lifrename(1), lifrm(1), lif(4).

DIAGNOSTICS

Lifls returns exit code 0 if the directory was listed successfully. Otherwise it prints a diagnostic and returns non-zero.

NAME

lifrename – rename LIF files

SYNOPSIS

lifrename oldfile newfile

DESCRIPTION

Oldfile is a full LIF file specifier (see *lif*(4) for details) for the file to be renamed (e.g. **liffile:A_FILE**). *Newfile* is new name to be given to the file (only the file name portion). This operation does not include copy or delete. Old file names must match the name of the file to be renamed, even if that file name is not a legal LIF name.

Note that you should **not** mount the special file before using *lifrename*.

EXAMPLES

lifrename liffile:A_FILE B_FILE
lifrename /dev/dsk/1s2:ABC CDE

AUTHOR

Lifrename was developed by the Hewlett-Packard Company.

SEE ALSO

lifcp(1), lifinit(1), lifls(1), lifrm(1), lif(4).

DIAGNOSTICS

Lifrename returns exit code 0 if the file name is changed successfully. Otherwise it prints a diagnostic and returns non-zero.

NAME

lifrm – remove a LIF file

SYNOPSIS

lifrm file1 ... filen

DESCRIPTION

Lifrm removes one or more entries from a LIF volume. File name specifiers are as described in *lif*(4).

Note that you should **not** mount the special file before using *lifrm*.

EXAMPLES

lifrm liffile:MAN
lifrm /dev/rdsk/1s2.0:F

AUTHOR

Lifrm was developed by the Hewlett-Packard Company.

SEE ALSO

lifcp(1), lifinit(1), lifls(1), lifrename(1), lif(4).

DIAGNOSTICS

Lifrm returns exit code 0 if the file is removed successfully. Otherwise it prints a diagnostic and returns non-zero.

NAME

line – read one line from user input

SYNOPSIS

line

DESCRIPTION

Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

EXAMPLES

The following lines in a shell script prompt for a file name and display information about the file.:

```
echo 'Enter file name: \c'
reply='line'
ls -l $reply
```

SEE ALSO

sh(1), read(2).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

line: SVID2, XPG2, XPG3

NAME

lint – a C program checker/verifier

SYNOPSIS

lint [*options*] *file* ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **–c** or the **–o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

Lint will take all the **.c**, **.ln**, and **llib-l***x***.ln** files (specified by **–l***x* and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the **–c** option is used, the **.ln** and the **llib-l***x***.ln** files are ignored.

Any number of *lint* options may be used, in any order, intermixed with file name arguments. The following options are used to suppress certain kinds of complaints:

–a Suppress complaints about assignments of long values to variables that are not long.

–b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints).

–h Do not apply heuristic tests that attempt to intuitively find bugs, improve style, and reduce waste.

–u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program.)

–v Suppress complaints about unused arguments in functions.

–x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

–l*x* Include additional lint library **llib-l***x***.ln**. For example, you can include a lint version of the Math Library **llib-lm.ln** by inserting **–lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

–n Do not check compatibility against either the standard or the portable lint library.

–p Attempt to check portability to other dialects of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

−c Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for inter-function compatibility.

−o lib Cause *lint* to create a lint library with the name **llib-l***lib***.ln**. The **−c** option nullifies any use of the **−o** option. The lint library produced is the input that is given to *lint*'s second pass. The **−o** option simply causes this file to be saved in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **−x** option is suggested. The **−v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see below).

The **−D**, **−U**, and **−I** options of *cpp*(1) and the **−g**, and **−O**, options of *cc*(1) are also recognized as separate arguments. The **−g** and **−O** options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc*(1) command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like *exit*(2)).

/*VARARGS*n*/
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
turns on the **−v** option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the **−v** and **−x** options.

Lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the **−c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **−c** and the **−o** options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the **−c** option. Each of these invocations produces a **.ln** file which corresponds to the **.c** file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the **−c** option), listing all the **.ln** files with the needed **−l***x* options. This will print all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

DEPENDENCIES

Series 300

Lint utilizes a special version of the C compiler front end. The size of the internal compiler tables can be adjusted by using the **-N** option. The syntax for this option is described in the DEPENDENCIES section of the manual page for *cc*(1).

The following option is supported:

−Y Enable support of 16-bit characters inside string literals and comments. Note that 8-bit parsing is always supported. See *hpnls*(5) for more details on International Support.

FILES

/usr/lib	the directory where the lint libraries specified by the **−lx** option must exist
/usr/lib/lint[12]	first and second passes
/usr/lib/llib−lc.ln	declarations for C Library functions (binary format; source is in **/usr/lib/llib−lc**)
/usr/lib/llib−port.ln	declarations for portable functions (binary format; source is in **/usr/lib/llib−port**)
/usr/lib/llib-lm.ln	declarations for Math Library functions (binary format; source is in **/usr/lib/llib-lm**)
/usr/tmp/*lint*	temporaries

WARNINGS

Exit(2), *longjmp* (on *setjmp*(3C)), and other functions that do not return are not understood; this causes various inaccuracies.

SEE ALSO

cc(1), cpp(1), make(1).

Lint C Program Checker, tutorial in *HP-UX Concepts and Tutorials: Programming Environment*.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of comments and string literals as single- and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *lint* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported within file names, comments, and string literals.

STANDARDS CONFORMANCE

lint: SVID2, XPG2, XPG3

NAME

lisp – HP Common Lisp environment

SYNOPSIS

lisp [*options*]

Remarks:

This command requires installation of optional HP Common Lisp software before it can be used.

DESCRIPTION

Lisp is the HP Common Lisp environment. It is provided in the following versions:

Series 300:	Lucid version 2.15
Series 800:	Lucid version 3.0

This manual entry focuses on HP Common Lisp version 3.0.

You can specify options when invoking the Lisp environment (see the section Options below). Once in the environment, you can compile functions, files, execute and debug programs, use the Lisp editor, have multiple processes running at the same time, use the windowing system, or use the object-oriented programming model called "Flavors". Refer to the *HP Common Lisp User's Guide* for more information.

Invoking Lisp

To invoke Lisp, type the Lisp environment or image name on the keyboard and press Return. For example, to invoke the Lisp environment from a Lisp image called **mylisp**, use the command:

mylisp [Return]

Shell environment variables can also be used to invoke a Lisp environment. First, set and export the variable in your **.profile** file (Bourne Shell or Korn Shell) or set the variable in your **.login** file (C Shell). For example, if using Korn Shell, add the following to **.profile**:

```
LISP=/usr/bin/lisp-de
export LISP
```

To invoke Lisp, type:

$LISP [Return]

Using this method, there is no need to update the environment variable until you move Lisp to a new directory or change the image name.

To initialize the Lisp environment when Lisp is invoked, create a file named **lisp-init.lisp** in your home directory. This initialization file can then be compiled for faster performance. *Lisp-init* is loaded whenever Lisp is invoked, and all Lisp expressions in the file are evaluated.

Another variable called ***enter-top-level-hook*** can be set to control the behavior of Lisp when Lisp is started. If the variable is bound to a compiled function or a symbol that names a compiled function, that function is called before Lisp enters the top level.

Options

The following command-line options are recognized when invoking the *Lisp* environment.

–load | **–l** *filename*	Load the specified file when starting Lisp.
–no-init-file | **–n**	Do not load the initialization file. The initialization file is **init-lisp.lisp** or **init-lisp.hbin**.
–eval | **–e** *form*	Evaluate *form*. The return value is lost. Produces side-effects only.

−quit | −q Terminate the Lisp environment. To terminate Lisp from the top level of the Lisp environment, type **(quit)**.

Lisp Compilation and Execution

The options listed under the commands below all start with **:** The options listed will not be discussed in detail in this document. For more information on the commands and their options, refer to the *HP Common Lisp User's Guide*.

compile-file "*path/filename*" [**:output-file** "*outputname*"]
[*compiler-options*
:messages :file-messages
:warnings :undef-warnings
:show-optimizations
:fast-entry :write-safety
:read-safety :tail-merge
:notinline :egc]

Compile-file compiles the Lisp source file specified by "*path/filename*". If no path is specified, Lisp uses the value of ***default-pathname-defaults*** to look for a file. The compiled code name is placed in a file having the same name as the source except it ends in **.hbin.** If the *:output-file* "*outputname*" option is specified, the compiled code is named "*outputname*".

Use compiler options to control the code optimization, increase the speed of compiled code, control the size of compiled code, suppress warnings, convert tail-recursive calls to iterative constructions, and enable execution of code during ephemeral garbage collection. Enter options when calling **compile-file** or by calling **compiler-options** first followed by **compile-file.**

For example, the following commands (denoted in **BOLD)** entered from the Lisp environment:

> (compiler-options :warnings nil
:show-optimizations t)
> (:WARNINGS NIL :SHOW-OPTIMIZATIONS T)
> (compile-file "dr.l" :output-file " mydr.hbin")

cause the **dr.l** Lisp source code to be compiled with compiler warning messages suppressed and code optimizations reported to the terminal. The compiled code will be written to a file called **mydr.hbin.**

load *filename* [*:verbose :print :if-does-not-exist*
:if-source-only :if-source-newer
:ignore-binary-dependencies]

Read and evaluate all forms in *filename* (i.e. load binary code after it is compiled so that it is available for use).

filename can be a pathname, stream, string, or a symbol. Specify **load** options to:

- Compile a source file first and then load it if the binary file did not exist when calling **load.**
- Print the values of all expressions that are loaded.
- Specify that you want to load whichever is newer, the source or the binary code.
- Load a binary file that was compiled for a certain run-time feature onto a machine that does not have the correct architecture.
- Load a source file without getting prompted to compile it.

Communicating With HP-UX

run-program *name* [*:input :output :error-output*
:wait :arguments

:if-input-does-not-exist
:if-output-exists
:if-error-output-exists]

Run-program runs HP-UX programs from the Lisp environment. *Name* contains the full pathname to the program to be run. This form is used to run shell scripts, specify options to get the HP-UX process id of the program being run, get the error output from a program, get the program exit status, and pass arguments to a program.

EXAMPLES

The following invokes the Lisp environment, loads a file called *myfun*, evaluates it (a message will print), and terminates Lisp — all by specifying command-line options when invoking the Lisp environment.

```
$ lisp -load "myfun" -eval "(myfun)" -quit
> hi there
$
```

This example shows how to run the Korn shell from Lisp, execute the *ls* command, and get back into Lisp.

```
> (run-program "/bin/ksh")
$ ls *.l
dr.l
mylisp.l
nl.l
$ exit
NIL
NIL
0
NIL
>
```

WARNINGS

When running a large application, be sure enough swap space is available to execute the program. In some cases it may be necessary to ensure that Lisp is the only process running in order to prevent running out of space, causing Lisp to halt.

DEPENDENCIES

If using the Window Tool Kit, X11 is required.

Level 0 Kanji support is provided. Kanji is allowed in comments, strings, and symbol print names. Kanji can be read and printed transparently and is treated like two Lisp ASCII characters. Kanji characters are not fully integrated as a Lisp data type into the Lisp system.

The Window Tool Kit and Editor do not support Kanji.

Series 300:

HP Common Lisp II 2.15 is available on Series 300 systems.

HP Common Lisp II 2.15 cannot be installed and run unless you have an ID-module, codeword certificate, and the necessary codeword.

Series 800:

A color monitor is required when using the color Window Tool Kit.

FILES

Files that end in **.l** or **.lisp** are understood to be Lisp source files, although it is not a requirement to end Lisp source files in **.l** or **.lisp.** When compiled, the resulting binary file ends with **.b** on Series 300 and **.hbin** on Series 800 systems.

Series 300:

file.l	Lisp source file
file.b	Lisp Binary file

Series 800:

file.lisp	Lisp source file
file.hbin	Lisp Binary file

AUTHOR

HP Common Lisp was developed by HP and Lucid, Incorporated.

SEE ALSO

HP Common Lisp User's Guide, HP Part No. 92640-60001
HP Common Lisp Advanced User's Guide, HP Part No. 92640-60002
HP Common Lisp Editor, HP Part No. 92640-60003
HP Common Lisp 2.15 User's Guide, HP Part No. 92688-90010
Common Lisp: The Language by Guy Steele, Jr., HP Part No. 9320-6047

NAME

ln − link files and directories

SYNOPSIS

ln [**−f**] [**−s**] *file1 new_file*
ln [**−f**] [**−s**] *file1* [*file2* ...] *dest_directory*
ln [**−f**] [**−s**] *directory1* [*directory2* ...] *dest_directory*

DESCRIPTION

Ln links:

- *file1* to new or existing *new_file*,
- *file1* to a new or existing file named *file1* in existing *dest_directory*,
- *file1*, *file2*, ... to new or existing files of the same name in existing *dest_directory*,
- or creates symbolic links between files or between directories.

If links are to *dest_directory*, corresponding filenames in that directory are linked to *file1*, *file2*, ..., etc. If two or more files (excluding destination file name *new_file*) are specified, the destination must be a directory. If *new_file* already exists as a regular file (or link to another file), its contents (or the existing link) are destroyed.

If the destination is an ordinary file and the access permissions of the file forbid writing, *ln* asks permission to overwrite the file. If the access permissions of the directory forbid writing, *ln* aborts and returns with the error message "cannot unlink *new_file*" (even if the file is an ordinary file and not a link to another file). When asking for permission to overwrite an existing file or link, *ln* prints the mode (see *chmod*(2) and Access Control Lists below), followed by the first letters of the words *yes* and *no* in the current native language, prompting for a response, and reading one line from the standard input. If the response is affirmative and is permissible, the operation occurs; if not, the command proceeds to the next source file, if any.

Ln does not permit hard links to a directory.

If *file1* is a file and *new_file* is a link to an existing file or an existing file with other links, *new_file* is disassociated from the existing file and links and linked to *file1*. When *ln* creates a link to a new or existing filename, ownerships and permissions are always identical to those for the file to which it is linked. If *chown*(1), *chgrp*(1), or *chmod*(1) is used to change ownership or permissions of a file or link, the change applies to the file and all associated links. The last modification time and last access time of the file and all associated links are identical.

Options

−f Perform *ln* commands without prompting for permission. This option is assumed when the standard input is not a terminal.

−s Cause *ln* to create symbolic links instead of the usual hard links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open*(2) operation is performed on the link. A *stat*(2) on a symbolic link returns the linked-to file; an *lstat*(2) must be performed to obtain information about the link. A *readlink*(2) call can be used to read the contents of the symbolic link. Symbolic links can span file systems and can refer to directories.

Access Control Lists (ACLs)

If optional ACL entries are associated with *new_file*, *ln* displays a plus sign (+) after the access mode when asking permission to overwrite the file.

If *new_file* is a new file, it inherits the access control list of *file1*, altered to reflect any difference

in ownership between the two files (see *acl*(5)).

EXAMPLES

The following command creates filenames *new_file1* and *new_file2* in *dest_dir* which are linked back to the original files *file1* and *file2*.

ln *file1 file2 dest_dir*

If *new_file1* and/or *new_file2* exists, it is removed and replaced by a link to *file1* or *file2*, respectively. If existing *new_file1* or *new_file2* is a link to another file or a file with links, the existing file remains. Only the link to *new_file* is broken and replaced by a new link to the *file1* or *file2*.

WARNINGS

Ln does not create hard links across file systems.

DEPENDENCIES

RFA and NFS

Access control lists of networked files are summarized (as returned in *st_mode* by *stat*(2)), but not copied to the new file. When using *ln* on such files, a + is not printed after the mode value when asking for permission to overwrite a file.

AUTHOR

Ln was developed by AT&T, the University of California, Berkeley and HP.

SEE ALSO

cp(1), cpio(1), mv(1), rm(1), link(1M), lstat(2), readlink(2), stat(2), symlink(2), symlink(4), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG and LC_CTYPE determine the local language equivalent of y (for yes/no queries).

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ln* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

ln: SVID2, XPG2, XPG3

NAME

lock – reserve a terminal

SYNOPSIS

lock

DESCRIPTION

Lock requests a password from the user, then prints "LOCKED" on the terminal and refuses to relinquish the terminal until the password is repeated. If the user forgets the password, he has no other recourse but to login elsewhere and kill the lock process.

NAME

logger – make entries in the system log

SYNOPSIS

logger [**−t** *tag*] [**−p** *pri*] [**−i**] [**−f** *file*] [*message* ...]

DESCRIPTION

Logger provides a program interface to the *syslog*(3C) system log module.

A message can be given on the command line, which is logged immediately, or a file is read and each line is logged. If no *file* or *message* is specified, the contents of the standard input are logged.

Options

−t *tag* — Mark every line in the log with the specified *tag*.

−p *pri* — Enter the message with the specified priority. The priority may be specified numerically or as a "facility.level" pair. For example, "−p local3.info" logs the message(s) as informational level in the **local3** facility. The default is "user.notice."

−i — Log the process ID of the logger process with each line.

−f *file* — Log the contents of the specified file.

message — The message to log; if not specified, the **−f** file or standard input is logged.

EXAMPLES

This command sends the message "System rebooted" to the *syslog* daemon:

```
logger System rebooted
```

This command sends the output of the *users*(1) command to the *syslog* daemon, marked as priority "info" and facility "local0". The message is tagged with the "USERS":

```
users | −p local0.info −t USERS
```

WARNINGS

Logger has no effect if the syslog daemon (*syslogd*(1M)) is not running on the system.

AUTHOR

Logger was developed by the University of California, Berkeley.

SEE ALSO

syslogd(1M), syslog(3C).

NAME

login – sign on

SYNOPSIS

login [name [env-var ...]]
login - -r rhost

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file."

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session. An invalid login name will cause a request for a password. This is done to make it more difficult for an unauthorized user to log in on the system by trial and error. After three unsuccessful login attempts, a *hangup* signal is issued.

The **-r** option is only useful on those installations which support the Berkeley remote login service *rlogin*. This option is used by the *rlogin* server to inform *login* that a remote login is being attempted from the given remote hostname *rhost*. *Login* then reads the remote user's name *remuser*, the local user's name *locuser*, and the user's remote terminal type. *Login* then uses the following three conditions to decide if the user can be logged in without asking for a password:

The remote host *rhost* appears in the file **/etc/hosts.equiv** and *remuser* = *locuser*.

The file **$HOME/.rhosts** contains a line listing just *rhost* and *remuser* = *locuser*, where **$HOME** is locuser's login directory.

The file **$HOME/.rhosts** contains a line listing the remote host *rhost* followed by the remote user *remuser*, seperated by exactly one space.

If none of these conditions are met, then a password is prompted for as if *locuser* had been specified as the user name on the command line. Once the user is logged in, *login* proceeds as in a normal login.

For security reasons, the following conditions also apply to the **-r** option:

Login must be running as the super-user (uid = 0).

If attempting to login as the super-user (uid = 0), then the file **/etc/hosts.equiv** is not checked, though the file **$HOME/.rhosts** is still searched.

The file **$HOME/.rhosts** must be owned either by *locuser* or by the super-user.

The file **$HOME/.rhosts** must not be a symbolic link on those installations which support them.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login. See *dialups*(4) for details on dialup security.

If password aging has been invoked by the super-user on your behalf, your password may have expired. In this case, you will be diverted into *passwd*(1) to change it, after which you may attempt to login again.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you will be silently disconnected.

After a successful login, the accounting files are updated, the command interpreter (usually *sh*(1)) is determined, and the user and group id's, group access list, and working directory are initialized. These specifications are found in the **/etc/passwd** and **/etc/logingroup** file entries for the user. The name of the command interpreter as passed to it is – followed by the last component of the interpreter's pathname (i.e., **–sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used. The command interpreter performs its own initialization, and does login initialization if the name by which it is called starts with –.

If *sh*(1) is the command interpreter, it executes the profile files **/etc/profile** and **$HOME/.profile** if they exist. Depending on what these profile files contain, you are notified of mail in your mail file or any messages you may have received since your last login.

If the command name field is "*", then a *chroot*(2) is done to the directory named in the directory field of the entry. At that point *login* is re-executed at the new level which must have its own root structure, including **/etc/login** and **/etc/passwd**.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

For the super-user, PATH is augmented to include **/etc**. In the case of a remote login, the enviroment variable TERM is also set to the remote user's terminal type.

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

L*n*=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

If **/etc/btmp** is present, all unsuccessful login attempts are logged to this file. This feature is disabled if the file is not present. A summary of bad login attempts may be viewed using **lastb**, see *last*(1).

If **/etc/securetty** is present, login security is in effect and the super-user may only login successfully on the ttys listed in this file. Ttys are listed by device name, one per line. Valid tty names are dependent on installation. Some examples could be "console", "tty01", "ttya1", etc. Note that this feature does not inhibit a normal user from using **su.**

FILES

$HOME/.profile	personal profile (individual user initialization)
$HOME/.rhosts	personal equivalence file for the remote login server
/etc/btmp	history of bad login attempts

/etc/d_passwd	dialup security encrypted passwords
/etc/dialups	lines which require dialup security
/etc/hosts.equiv	system list of equivalent hosts allowing logins without passwords
/etc/logingroup	group file – defines group access lists
/etc/motd	message-of-the-day
/etc/passwd	password file – defines users, passwords, and primary groups
/etc/profile	system profile (initialization for all users)
/etc/securetty	list of valid ttys for root login
/etc/utmp	users currently logged in
/etc/wtmp	history of logins, logouts, and date changes
/usr/mail/*your-name*	mailbox for user *your-name*

VARIABLES

HOME	The users home directory.
PATH	The path to be searched for commands.
SHELL	Which command interpreter is being used.
MAIL	Where to look for mail.
TERM	The user's terminal type.
TZ	The current timezone.
xxx	User specified named variables.
L *xxx*	User specified unnamed variables.

SEE ALSO

mail(1), newgrp(1), passwd(1), sh(1), su(1), getty(1M), last(1M), initgroups(3C), dialups(4), group(4), passwd(4), profile(4), utmp(4), environ(5).

DIAGNOSTICS

The following diagnostics will appear if problems occur:

Login incorrect:
if the user name or the password cannot be matched.

No shell, **cannot open password file**, or **no directory**:
consult your system manager.

Your password has expired. Choose a new one:
if password aging is implemented.

No Root Directory:
attempted to log into a subdirectory that does not exist (i.e., passwd file entry had shell name "*", but the system cannot *chroot* to the given directory).

No /bin/login or /etc/login on root:
same as above except sub-root login command not found.

Bad user id. or **Bad group id.**:
setuid or *setgid* failed.

Unable to change to directory <name>:
cannot *chdir* to your home directory.

No shell: your shell (or /bin/sh if your shell name is null in **/etc/passwd**) could not be *exec*'d.

Sorry, single-user:
occurs if the version field from *uname*(2) starts with **A** (or if the *uname* system call fails) and if your terminal name is not /dev/console and if your home shell is not named /usr/lib/uucp/uucico. You are not logged in.

No utmp entry. You must exec "login" from the lowest level "sh":
if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

.rhosts is a soft link:
if your personal equivalence file is a symbolic link.

Bad .rhosts ownership:
if your personal equivalence file is not owned by the local user or by the super-user.

Remuser too long, **locuser too long**, or **terminal type too long**:
if the indicated string was too long for *login's* internal buffer.

AUTHOR

Login was developed by AT&T and HP.

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

Logname writes the user's login name to standard output. The login name is equivalent to that returned by *getlogin*(2).

FILES

/etc/profile

SEE ALSO

env(1), login(1), getlogin(3C), logname(3C), environ(5).

STANDARDS CONFORMANCE

logname: SVID2, XPG2, XPG3

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1). Note that the link editor *ld*(1) is capable of multiple passes over an archive in the archive format and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar cr library 'lorder *.o | tsort'
```

FILES

*symref, *symdef temporary files

SEE ALSO

ar(1), ld(1), tsort(1).

BUGS

Object files whose names do not end with **.o**, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

lorder: SVID2, XPG2

NAME

lp, cancel, lpalt – send/cancel/alter requests to an LP line printer or plotter

SYNOPSIS

lp [**−c**] [**−d***dest*] [**−m**] [**−n***number*] [**−o***option*] [**−p***priority*] [**−s**] [**−t***title*] [**−w**] [*files* ...]

cancel [*ids*] [*printers*] [**−a**] [**−e**] [**−i**] [**−u***user*]

lpalt *id* [**−d***dest*] [**−i**] [**−m**] [**−n***number*] [**−o***option*] [**−p***priority*] [**−s**] [**−t***title*] [**−w**]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer or supported plotter. If no file names are specified, standard input is assumed. The file name − stands for standard input and can be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed or plotted.

Lp associates a unique ID with each request and prints it on the standard output. This ID can be used later to cancel (see *cancel*), alter (see *lpalt*), or find the status (see *lpstat*(1)) of the request.

The following options to *lp* can be specified in any order, and can be intermixed with file names:

−c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will be linked into a spool directory. Ownership and mode of the linked *files* remains unchanged. If the **−c** option is given or linking is not possible then *files* are copied, in which case the ownership and mode are set to allow read access to owner *lp* and group *bin* only. It should be noted that if the *files* are linked rather than copied, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output. The standard input is always copied instead of linked.

−d *dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

−m Send mail (see *mail*(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n *number* Print *number* copies (default of 1) of the output.

−o *option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once. For more information about what *options* are valid for printers supported on your system, see the particular model script in **/usr/spool/lp/model** associated with the specified printer.

−p *priority* Give *priority* to the print request. This parameter is used to select next spooled file for the targeted printer or class of printers. *Priority* must be in between 0 (lowest priority) and 7 (highest priority). If the value is less than *fence* (see *lpfence*(1M)), the print request is deferred. Default is the default *priority* set by *lpadmin*(1M) of the printer when printer is specified for *dest*, and is the highest default *priority* among printers of the class when class is specified for *dest*.

−s Suppress messages from *lp* such as "request ID is ...".

−t *title* Print *title* on the banner page of the output.

−w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in or *rlpdaemon(1M)* is not running on local system for remote printing, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp* command. At least one *id* or *printer* must be specified.

ids Specify a request *id* (as returned by *lp*) for a local printer. This cancels the associated request even if it is currently printing. Specifying a request *id* (as returned by *lp*) for a remote printer cancels the associated request if it is owned by the user, even if it is currently printing.

printers Specify the names of *printers* (for a complete list, use *lpstat*(1)). Specifying a local printer cancels the request which is currently printing on that printer. Specifying a remote printer cancels the request which is currently printing on that printer if it is owned by the user. If the **−a**, **−e** or the **−u** option is specified, this option specifies the printer on which to perform the cancel operation.

−a Remove all requests a user owns on the specified printer (see *printers*). The owner is determined by the user's login name and host name on the machine where the *lp* command was invoked.

−e Empty the spool queue of all requests for the specified printer (see *printers*). Only the super-user can use the **−e** option.

−i Cancel only local requests.

−u *user* Remove any requests queued belonging to user. Multiple **−u** options are allowed. Only the super-user can use the **−u** option.

In any case, the cancellation of a request that is currently printing frees the printer to print its next available request.

Lpalt alters a line printer request that was made by the *lp* command. New unique ID is returned to the standard output.

id Specify a request *id* (as returned by *lp*) for a local printer. This alters the associated request only if it is not currently printing. Specifying a request *id* (as returned by *lp*) for a remote printer alters the associated request if it is owned by the user, only if it is not currently printing.

−i Alter only local requests.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

RETURN VALUE

Exit values are:

0 Successful completion.
>0 Error condition occured.

EXAMPLES

Assuming there is an existing HP 2934A line printer named *lp2*, configured with the **hp2934a** model interface program. This model has the **−c** option which causes the printer to print in a compressed mode. To obtain compressed print on *lp2*, use the command:

```
lp -dlp2 -oc files
```

WARNINGS

A remote print request can be cancelled only by the user who requested it, and only on the system from which the request was spooled.

If the restrict cancel feature (selected by **-orc** in *lpadmin*(1M)) is enabled for the specified printer, a user can only cancel/alter requests owned by the user.

For remote system, *lpalt* cannot change *dest* and *priority*.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1), slp(1), accept(1M), lpadmin(1M), lpana(1M), lpsched(1M), mklp(1M), rcancel(1M), rlp(1M), rlpdaemon(1M), rlpstat(1M).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *lp* and *cancel* behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

lp: SVID2, XPG2, XPG3

cancel: SVID2

NAME

lpstat – print LP status information

SYNOPSIS

lpstat [options] [id...]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
−u"user1, user2, user3"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat −o
```

prints the status of all output requests.

−a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

−c[*list*] Print class names and their members. *List* is a list of class names.

−d Print the system default destination for *lp*.

−i Inhibit the reporting of remote status.

−o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*. See the **−i** option.

−p[*list*] Print the status of printers. *List* is a list of printer names.

−r Print the status of the LP request scheduler

−s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

−t Print all status information. Same as specifying **−r**, **−s**, **−a**, **−p**, **−o**. See the **−i** option.

−u[*list*] Print status of output requests for users. *List* is a list of login names.

−v[*list*] Print the names of printers and the path names of the devices associated with them. *List* is a list of printer names.

In the HP Clustered environment, all printers are attached to the root server and all spooling is handled as if the cluster were a single system. Remote spooling applies to spooling from or to machines outside of the cluster.

EXAMPLES

To check if your job is queued, type:

```
lpstat
```

To check where in line a queued job is, type:

lpstat -t

To check if the job scheduler is running, type:

lpstat -r

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), rlpstat(1M).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

LANG determines the language in which messages are displayed.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *lpstat* behaves as if all internationalization variables are set to "C". See *environ*(5).

STANDARDS CONFORMANCE

lpstat: SVID2, XPG2, XPG3

NAME

ls, l, ll, lsf, lsr, lsx – list contents of directories

SYNOPSIS

ls [**−abcdfgilmnopqrstuxACFHLR1**] [*names*]
l [*ls options*] [*names*]
ll [*ls options*] [*names*]
lsf [*ls options*] [*names*]
lsr [*ls options*] [*names*]
lsx [*ls options*] [*names*]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted in ascending collation order by default (see Environment Variables below). When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

If you are the super-user, all files except . and .. are listed by default.

There are three major listing formats. The format chosen depends on whether the output is going to a login device, and may also be controlled by option flags. The default format for a teletype is to list the contents of directories in multi-column format, with the entries sorted down the columns. (When individual file names (as opposed to directory names) appear in the argument list, those file names are always sorted across the page rather than down the page in columns. This is because the individual file names may be arbitrarily long.) If the standard output is not a teletype, the default format is to list one entry per line, the **−C** and **−x** options enable multi-column formats, and the **−m** option enables stream output format in which files are listed across the page, separated by commas. In order to determine output formats for the **−C**, **−x**, and **−m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo* database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns is assumed.

Options

There are numerous options:

−a List all entries; usually entries whose names begin with a period (**.**) are not listed.

−b Force printing of non-graphic characters to be in the octal \ddd notation.

−c Use time of last modification of the inode (file created, mode changed, etc.) for sorting (**−t**) or printing (**−l**).

−d If an argument is a directory, list only its name (not its contents); often used with **−l** to get the status of a directory.

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l**, **−t**, **−s**, and **−r**, and turns on **−a**; the order is the order in which entries appear in the directory.

−g The same as **−l**, except that only the group is printed (owner is omitted.) (If both **-l** and **-g** are specified, the owner is not printed.)

−i For each file, print the i-number in the first column of the report.

−l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see further DESCRIPTION and Access Control Lists below). If the file is a special file, the size field contains the major and minor device numbers rather than a size.

−m Stream output format.

−n The same as **−l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

−o The same as **−l**, except that only the owner is printed (group is omitted.) (If both **-l** and **-o** are specified, the group is not printed.)

−p Put a slash (/) after each file name if that file is a directory.

−q Force printing of non-graphic characters in file names as the character (**?**).

−r Reverse the order of sort to get reverse (descending) collation or oldest first, as appropriate.

−s Give size in blocks, including indirect blocks, for each entry.

−t Sort by time modified (latest first) instead of by name.

−u Use time of last access instead of last modification for sorting (with the **−t** option) or printing (with the **−l** option).

−x Multi-column output with entries sorted across rather than down the page.

−A The same as **−a**, except that the current directory "." and parent directory ".." are not listed. For the super-user, this flag defaults to ON, and is turned off by **−A**.

−C Multi-column output with entries sorted down the columns.

−F Put a slash (/) after each file name if that file is a directory, put an asterisk (*) after each file name if that file is executable, and put an at sign (@) after each file name if that file is a symbolic link.

−H Put a plus sign (+) after each file name if that file is a hidden directory (context-dependent file). This option also implies **−F**. See *cdf*(4).

When used with the **−l** option on device files, the cnode name (if found in **/etc/clusterconf**) or cnode ID of the device file is printed following the minor device number. A cnode ID of "0" is used to indicate a device file that can be accessed from any cnode.

−L If the argument is a symbolic link, list the file or directory to which the link refers rather than the link itself.

−R Recursively list subdirectories encountered.

−1 The file names will be listed in single column format regardless of the output device. This will force single column format to the user's terminal.

Ls normally is known by several names which provide shorthands for the various formats:

l is equivalent to **ls −m**.
ll is equivalent to **ls −l**.
lsf is equivalent to **ls −F**.
lsr is equivalent to **ls −R**.
lsx is equivalent to **ls −x**.

The shorthand notations are implemented as links to *ls*. Option arguments to the shorthand versions behave exactly as if the long form above had been used with the additional arguments.

The mode printed under the **−l** option consists of 10 characters. The first character is:

d if the entry is a directory;
b if the entry is a block special file;

c if the entry is a character special file;
l if the entry is a symbolic link;
H if the entry is a hidden directory (context-dependent file). Hidden directories are only visible when the **−H** option has been specified;
p if the entry is a fifo (a.k.a. "named pipe") special file;
n if the entry is a network special file;
s if the entry is a socket;
− if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions, the next to permissions of others in the user-group of the file, and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
− if the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **−**) is **t** if the 1000 (octal) bit of the mode is on; see *chmod*(1) for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively), if the corresponding execute permission is not set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

Access Control Lists (ACLs)

If a file has optional ACL entries, the **−l** option displays a plus sign (+) after the file's permissions. The permissions shown are a summary representation of the file's access control list, as returned by *stat*(2) in the *st_mode* field. To list the contents of an access control list, use the *lsacl*(1) command. See also *acl*(5).

EXAMPLES

The following command prints a long listing of all the files (including the file sizes) in the current working directory. The file most recently modified (the youngest) is listed first, then the next youngest file, and so forth, to the oldest. Files beginning with a **.** are also printed.

```
ls -alst
```

WARNINGS

The option setting based on whether the output is a teletype is undesirable as **ls −s** is much different than **ls −s | lpr**. On the other hand, not using this setting would make old shell scripts that used *ls* almost inevitably fail.

Unprintable characters in file names may confuse the columnar output options.

DEPENDENCIES

RFA and NFS

The **−l** option does not display a plus sign (+), representing existence of optional access control list entries, after the access permission bits of networked files.

AUTHOR

Ls was developed by AT&T, the University of California, Berkeley and HP.

FILES

/etc/passwd	to get user IDs for **ls** **−l** and **ls** **−o**.
/etc/group	to get group IDs for **ls** **−l** and **ls** **−g**.
/usr/lib/terminfo/?/*	to get terminal information.

SEE ALSO

chmod(1), find(1), lsacl(1), stat(2), cdf(4), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

LC_CTYPE determines which characters are classified as non-graphic for the **−b** and **−q** options, and the interpretation of single and/or multi-byte characters within file names.

LC_TIME determines the date and time strings output by the **−g**,**−l**,**−n** and **−o** options.

LANG determines the language in which messages (other than the date and time strings) are displayed.

If LC_COLLATE, LC_CTYPE, or LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ls* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

ls: SVID2, XPG2, XPG3

NAME

lsacl – list access control lists (ACLs) of files

SYNOPSIS

lsacl [**–l**] *file ...*

Remarks:

To ensure continued conformance with emerging industry standards, features described in this manual entry are likely to change in a future release.

DESCRIPTION

Lsacl lists access control lists (ACLs) of one or more files in symbolic, "short" form, one file's ACL per line of output, followed by the file name; see *acl*(5) for ACL syntax.

Options

–l Print ACLs in long form. Each file's ACL can be more than one line long, and is always preceded by file name, colon, and newline. Consecutive file names are separated by blank lines.

If a hyphen (–) is given as a file name argument, *lsacl* prints the ACL for the standard input. By default, it prints the file name as –. For the **–l** option it prints a file name of "<stdin>".

Unlike *ls*(1), *lsacl* cannot list ACLs of files in subdirectories automatically or list the ACL entries of the files in the current directory by default. A good way to do the latter is:

```
lsacl *
```

or

```
lsacl .* *
```

RETURN VALUE

If *lsacl* succeeds, it returns zero. If invoked incorrectly, it returns a value of **1**. If *lsacl* is unable to read the ACL for a specified file, it prints an error message to standard error, continues, and later returns a value of **2**.

EXAMPLES

The following command lists the ACL of the file "dir/file1."

```
lsacl dir/file1
```

Here's how to list ACLs for all files in the current directory, in long form.

```
lsacl –l .* *
```

Here's how to list ACLs for all files under "mydir".

```
find mydir -print | sort | xargs lsacl
```

DEPENDENCIES

RFA and NFS

Lsacl is not supported on remote files.

AUTHOR

Lsacl was developed by HP.

SEE ALSO

chacl(1), getaccess(1), ls(1), getacl(2), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *lsacl* behaves

as if all internationalization variables are set to "C". See *environ*(5).

NAME

m4 – macro processor

SYNOPSIS

m4 [*options*] [*file* ...]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is –, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

–e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode may be very difficult.

–s Enable line sync output for the C preprocessor (#line ...)

–B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.

–H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.

–S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

–T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, the flags listed above must appear before any file names and before any **–D** or **–U** flags.

–D*name*[=*val*]
Defines *name* to *val* or to null in *val*'s absence.

–U*name*
undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The **(** must immediately follow the name of the macro. If the name of a defined macro is not followed by a **(**, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is

done the original meaning is lost. Their values are null unless otherwise stated.

changecom change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.

changequote change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., ` ').

decr returns the value of its argument decremented by 1.

define the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **$***n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **@ $@** is like **$***, but each argument is quoted (with the current quotes).

defn returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

divert *m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

divnum returns the value of the current output stream.

dnl reads and discards characters up to and including the next new-line.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.

errprint prints its argument on the diagnostic output file.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ** (exponentiation), bitwise **&**, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

ifdef if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on HP-UX system versions of *m4*.

ifelse has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

include returns the contents of the file named in the argument.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

index returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

len returns the number of characters in its argument.

m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup()')
maketemp	fills in a string of XXXXX in its argument with the current process ID.
popdef	removes current definition of its argument(s), exposing the previous one, if any.
pushdef	like *define*, but saves any previous definition.
shift	returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
sinclude	is identical to *include*, except that it says nothing if the file is inaccessible.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
syscmd	executes the HP-UX system command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd*.
traceoff	turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.
traceon	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
undefine	removes the definition of the macro named in its argument.
undivert	causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

SEE ALSO

cc(1), cpp(1).

The M4 Macro Processor, tutorial in *HP-UX Concepts and Tutorials: Programming Environment.*

STANDARDS CONFORMANCE

m4: SVID2, XPG2, XPG3

NAME

hp9000s200, hp9000s300, hp9000s500, hp9000s800, pdp11, u3b, u3b5, vax – provide truth value about your processor type

SYNOPSIS

hp9000s200

hp9000s300

hp9000s500

hp9000s800

pdp11

u3b

u3b5

vax

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

hp9000s200
True if you are on a Hewlett-Packard 9000 Series 200.

hp9000s300
True if you are on a Hewlett-Packard 9000 Series 300.

hp9000s500
True if you are on a Hewlett-Packard 9000 Series 500.

hp9000s800
True if you are on a Hewlett-Packard 9000 Series 800.

pdp11 True if you are on a PDP-11/45 or PDP-11/70.

u3b True if you are on a 3B 20S computer.

u3b5 True if you are on a 3B 5 computer.

vax True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (non-zero) value. These commands are often used within *make*(1) makefiles and shell procedures to increase portability.

SEE ALSO

make(1), sh(1), test(1), true(1).

NAME

mail, rmail − send mail to users or read mail

SYNOPSIS

mail [+] [**−epqr**] [**−f** *file*]

mail [**−dt**] *person* ...

rmail [**−dt**] *person* ...

DESCRIPTION

Note: An enhanced user mail interface is presented in *mailx*(1).

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input to determine the disposition of the message. Commands that automatically proceed to the next message will exit from *mail* if already on the last message. The following is a list of commands:

<new-line>	Go on to next message. Exit if already on last message.
+	Same as <new-line>.
n	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
−	Go back to previous message.
s [*files*]	Save message in the named *files* (**mbox** is default), mark the message for deletion from the user's *mailfile* and proceed to next message.
y [**files**]	Same as **s**.
w [*files*]	Save message without its header (the "From ..." line), in the named *files* (**mbox** is default), mark the message for deletion, and go on to next message.
m *person* ...	Mail the message to each named *person*, mark the message for deletion, and go on to next message.
q	Put undeleted mail back in the *mailfile* and stop.
EOT (control-d)	Same as **q**.
x	Put all mail back in the *mailfile* unchanged and stop.
!*command*	Escape to the command interpreter to do *command*.
?	Print a command summary.
*	Same as **?**.

The following optional arguments alter the printing of the mail:

+	Cause messages to be printed in first-in, first-out order.
−e	Cause mail not to be printed. An exit value is returned: 0 = Mail present 1 = No mail 2 = Other error
−p	Cause all mail to be printed without prompting for disposition.
−q	causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the printing of the current message.
−r	Same as +.

−f *file* Causes *mail* to use *file* (for example, **mbox**) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark.

The **−t** option causes the message to be preceded by each *person* the mail is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of **dead.letter**.

The **−d** option causes *mail* to deliver mail directly. This isolates *mail* from making routing decisions and allows it to be used as a local delivery agent. Typically this option is used by auto-routing facilities when they deliver mail locally.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1)). Everything after the first exclamation mark in *person* is interpreted by the remote system. In particular, if *person* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will interpret that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**. *Mail* will not use *uucp* if the remote system is the local system name (i.e., localsystem!user).

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment. In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

Rmail only permits the sending of mail; *uucp*(1) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

FILES

/usr/mail/*.lock	Lock for mail directory
dead.letter	Unmailable text
/tmp/ma*	Temporary file
$MAIL	Variable containing path name of *mailfile*
$HOME/mbox	Saved mail
/etc/passwd	To identify sender and locate persons
/usr/mail	Directory for incoming mail (mode "775", group ID "mail")
/usr/mail/*user*	Incoming mail for *user*; that is, the *mailfile* (mode "660", group ID "mail")

WARNINGS

Conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed. Printing may be forced by typing a **p**.

Lines that look like postmarks in the message, (that is, "From ...") are preceded with a >.

Mail treats a line consisting solely of a dot (".") as the end of the message.

SEE ALSO

login(1), mailx(1), uucp(1), write(1).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of the displayed date and time strings.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *mail* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Between HP-UX systems, single- and multi-byte character code sets are supported within mail text. Headers are restricted to characters from the 7-bit USASCII code set (see *ASCII*(5)).

STANDARDS CONFORMANCE

mail: SVID2, XPG2, XPG3

rmail: SVID2

NAME

mailfrom – summarize mail folders by subject and sender

SYNOPSIS

mailfrom [**–n**] [*filename*]

DESCRIPTION

Mailfrom outputs a line per message in the folder specified, or the users incoming mailbox of the form;

from subject

If a *filename* is specified, the program reads that file rather than the incoming mailbox. If it's not possible to access the specified file due to its permission, it only returns the message "No mail".

Furthermore, if the option **–n** is specified, the headers will be numbered using the same numbering scheme that, for example, **readmail(1)** will understand.

AUTHOR

mailfrom was developed by Hewlett-Packard Company.

SEE ALSO

elm(1), readmail(1).

NAME

mailx − interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

Mailx provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the system *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage (unless specific action is taken) so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see **MBOX**, in ENVIRONMENT VARIABLES below for a description of this file). Messages remain in this file until specifically removed.

On the command line, *options* start with a hyphen (−), and any other arguments are assumed to be destinations (recipients). If no recipients are specified, *mailx* attempts to read messages from the *mailbox*. Command line options are:

−d	Turn on debugging output. Neither particularly interesting nor recommended.
−e	Test for presence of mail. *Mailx* prints nothing and exits with a successful return code if there is mail to read.
−f [*filename*]	Read messages from *filename* instead of *mailbox* . If no *filename* is specified, the *mbox* is used.
−F	Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
−h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.
−H	Print header summary only.
−i	Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
−n	Do not initialize from the system default *mailx.rc* file.
−N	Do not print initial header summary.
−r *address*	Pass *address* to network delivery software. All tilde commands are disabled.
−s *subject*	Set the Subject header field to *subject*.
−u *user*	Read *user's mailbox*. This is only effective if *user's mailbox* is not read protected.
−U	Convert UUCP style addresses to internet standards. Overrides the "conv" environment variable.

When reading mail, *mailx* is in *command*mode. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input*mode. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands can be entered by beginning a line with

the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any given time, the behavior of *mailx* is governed by a set of *environment variables;* flags and valued parameters which are set and cleared by using the **set** and **unset** commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line can be of three types: login names, shell commands, or alias groups. Login names can be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the **alias** command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are treated as input for the message.

Each message is assigned a sequential number, and there is always the notion of a 'current' message, marked by a '>' in the header summary. Many commands take an optional list of messages (*msglist*)*to*operate*on*, A *msglist* is a list of message specifications separated by spaces which can include:

n	Message number *n*.
.	The current message.
^	The first undeleted message.
$	The last message.
*	All messages.
n–*m*	An inclusive range of message numbers, *n* through *m*.
user	All messages from *user*.
/string	All messages with *string* in the subject line (uppercase/lowercase differences ignored).
:*c*	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands, and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized

variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **C**opy, **e**dit, **f**ollowup, **F**ollowup, **h**old, **m**ail, **pre**serve, **r**eply, **R**eply, **sh**ell, and **v**isual. Any errors in the start-up file cause the remaining lines in the file to be ignored.

COMMANDS

The following is a complete list of *mailx* commands:

! *command* — Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

*comment* — Null command (comment). This may be useful in *.mailrc* files.

= — Print the current message number.

? — Prints a summary of commands.

<new-line> — Advance to next message and **p**rint. If this is the first command entered, the first unread message is printed.

alias *alias name...*
group *alias name...*
Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates *name...*
Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alt**ernates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]
chdir [*directory*] Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*
Copy messages to the file without marking the messages as saved. Otherwise equivalent to the **s**ave command.

Copy [*msglist*] Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the **S**ave command.

delete [*msglist*] Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES). See also **dp**.

discard [*header-field...*]
ignore [*header-field...*]
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

dp [*msglist*]
dt [*msglist*] Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a **d**elete command followed by a **p**rint command.

echo *string...* Echo the given strings (like *echo*(1)).

edit [*msglist*] Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT

VARIABLES). Default editor is *ed*(1).

exit
xit
: Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **q**uit).

file [*filename*]
folder [*filename*]
: Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current *mailbox*.
%**user**	the *mailbox* for **user**.
#	the previous file.
&	the current *mbox*.

Default file is the current *mailbox*.

folders
: Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]
: Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **F**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]
: Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **fo**llowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]
: Prints the header summary for the specified messages.

group *alias name...*
alias *alias name...*
: Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]
: Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the **z** command.

help
: Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
: Holds the specified messages in the *mailbox*.

if *s*|*r*
*mail-command*s
else
*mail-command*s
endif
: Conditional execution, where *s* will execute following *mail-command*s, up to an **el**se or **en**dif, if the program is in *send* mode, and *r* causes the *mail-command*s to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore *header-field...*
discard *header-field...*
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

list Prints all commands available. No explanation is given.

mail *name...* Mail a message to the specified users.

mbox [*msglist*] Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **ex**it and **q**uit commands.

next [*message*] Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglist*s above for a description of possible message specifications.

pipe [*msglist*] [*command*]
| [*msglist*][*command*]
Pipe the message through the given *command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*] Preserve the specified messages in the *mailbox*.

Print [*msglist*]
Type [*msglist*] Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

print [*msglist*]
type [*msglist*] Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]
respond [*message*]
Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*] Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with

all network addressing stripped off. See also the **C**opy, **f**ollowup, and **F**ollowup commands and "outfolder" (see ENVIRONMENT VARIABLES).

save [*filename*]
save [*msglist*] *filename*
Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the **ex**it and **q**uit commands).

set
set *name*
set *name=string*
set *name=number*
Define a variable called *name*. The variable may be given a null, string, or numeric value. **Se**t by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [*msglist*] Print the size in characters of the specified messages.

source *filename* Read commands from the given file and return to command mode.

top [*msglist*] Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*] Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See **ex**it and **q**uit.

Type [*msglist*]
Print [*msglist*] Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

type [*msglist*]
print [*msglist*] Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

unalias *alias* Discard the specified *alias* names.

undelete [*msglist*]
Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name...* Cause the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version Prints the current version and release date.

visual [*msglist*] Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*
Write the given messages on the specified file, minus the header (the "From ..."

line) and trailing blank line. Otherwise equivalent to the **save** command.

xit
exit Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

z[+|-] Scroll the header display forward or backward one screen–full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~!*command* Escape to the shell.

~. Simulate end of file (terminate message input).

~:*mail-command*
~_ *mail-command* Perform the command-level request. Valid only when sending a message while reading mail.

~? Print a summary of tilde escapes.

~**A** Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~**a** Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~**b** *name ...* Add *name* to the blind carbon copy (Bcc) list.

~**c** *name ...* Add *name* to the carbon copy (Cc) list.

~**d** Read in the *dead.letter* file. See "DEAD" (under ENVIRONMENT VARIABLES) for a description of this file.

~**e** Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~**f** [*msglist*] Forward the specified messages. The messages are inserted into the message, without alteration.

~**h** Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~**i** *string* Insert the value of the named variable into the text of the message. For example, ~**A** is equivalent to '~**i** Sign.'

~**m** [*msglist*] Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~**p** Print the message being entered.

~**q** Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See "DEAD" (under ENVIRONMENT VARIABLES) for a description of this file.

~**R** *name ...* Add *name* to the Reply-To list.

~**r** *filename*
~< *filename*
~< !*command*
: Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~**s** *string ...*
: Set the subject line to *string*.

~**t** *name ...*
: Add the given *name*s to the To list.

~**v**
: Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~**w** *filename*
: Write the partial message onto the given file, without the header.

~**x**
: Exit as with ~**q** except the message is not saved in *dead.letter*.

~| *command*
: Pipe the body of the message through the given *command*. If the *command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **unset** command may be used to erase variables.

allnet
: All network names whose login name match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alt**ernates command and the "metoo" variable.

append
: Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**

askbcc
: Prompt for the Bcc list after the message is entered. Default is **noaskbcc.**

askcc
: Prompt for the Cc list after the message is entered. Default is **noaskcc**.

asksub
: Prompt for a subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint
: Enable automatic printing of messages after **d**elete and **u**ndelete commands. Default is **noautoprint**.

bang
: Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang.**

cmd = *command*
: Set the default command for the **pi**pe command. No default value.

conv = *conversion*
: Convert UUCP addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **-U** command line option.

crt = *number*
: Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (*pg*(1) by default). Disabled by default.

DEAD = *filename*
: The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is $HOME/dead.letter.

debug Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR = *command* The command to run when the **e**dit or ~**e** command is used. Default is *ed*(1).

escape = *c* Substitute *c* for the ~ escape character.

folder = *directory* The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is used as a prefix. There is no default for the "folder" variable. See also "outfolder" below.

header Enable printing of the header summary when entering *mailx*. Enabled by default.

hold Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the ~. command. Default is **noignoreeof**. See also "dot" above.

keep When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX = *filename* The name of the file to save messages which have been read. The **x**it command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo Usually, when a group (alias) containing the sender is expanded, the sender is removed from the expansion. Setting this option causes the sender to be included in the group. Default is **nometoo**.

LISTER = *command* The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder Cause the files used to record outgoing messages to be located in the directory specified by the "folder" variable. Default is **nooutfolder**. See "folder" above and the **S**ave, **C**opy, **fo**llowup, and **F**ollowup commands.

page Used with the **pi**pe command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER = *command* The command to use as a filter for paginating output. This can also be used to specify the options to be used (for example, set PAGER="more -c"). Default is *pg*(1).

prompt = *string* Set the *command mode* prompt to *string*. Default is "? ".

quiet Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record = *filename* Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen = *number* Set the number of lines in a screen–full of headers for the **h**eaders command.

sendmail = *command* Alternate command for delivering messages. Default is *mail*(1).

sendwait Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL = *command* The name of a preferred command interpreter. Default is *sh*(1).

showto When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign = *string* The variable inserted into the text of a message when the ~**a** (autograph) command is given. No default (see also ~**i** in TILDE ESCAPES).

Sign = *string* The variable inserted into the text of a message when the ~**A** command is given. No default (see also ~**i** (TILDE ESCAPES)).

SMARTMAILER When **SMARTMAILER** is set, various commands use the "From:" line, instead of the default "From " line.

toplines = *number* The number of lines of header to print with the **top** command. Default is 5.

VISUAL = *command* The name of a preferred screen editor. Default is *vi*(1).

WARNINGS

Where *command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new internationalization standards need some time to settle down.

Mail(1) (the standard mail delivery program) treats a line consisting solely of a dot (".") as the end of the message.

FILES

/usr/mail/	Post office directory (mode 775, group ID **mail**)
/usr/mail/*user*	System *mailbox* for *user* (mode 660, owned by *user*,*group* ID **mail**)
$HOME/.mailrc	Personal start-up file

/usr/lib/mailx/mailx.help*	Help message files
/usr/lib/mailx/mailx.rc	Global start-up file
$HOME/mbox	Secondary storage file
/tmp/R[emqsx]*	Temporary files

SEE ALSO

mail(1), pg(1), ls(1).

Mailx Mail Handler, tutorial in *HP-UX Concepts and Tutorials: Shells and Miscellaneous Tools.*

EXTERNAL INFLUENCES

Environment Variables

The following are environment variables taken from the execution environment and are not alterable within *mailx.*

HOME *directory* The user's base of operations.

MAILRC *filename*
The name of the start-up file. Default is $HOME/.mailrc.

LC_COLLATE, LC_CTYPE
LC_COLLATE and LC_CTYPE influence *mailx* when the command interpreter (see SHELL below) is invoked. See the manual page for the applicable comand interpreter to determine the behavior of LC_COLLATE and LC_CTYPE.

LC_TIME LC_TIME determines the format and contents of the date and time strings displayed. If LC_TIME is not specified in the environment, or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *mailx* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported within mail text. Headers are restricted to characters from the 7-bit USASCII character code set (see *ascii*(5)).

STANDARDS CONFORMANCE

mailx: SVID2, XPG2, XPG3

NAME

make − maintain, update, and regenerate groups of programs

SYNOPSIS

make [**−f** *makefile*] [**−p**] [**−i**] [**−k**] [**−s**] [**−r**] [**−n**] [**−b**] [**−e**] [**−t**] [**−d**] [**−q**] [*names*]

DESCRIPTION

The following is a brief description of all options and some special names. Options can occur in any order.

−f *makefile* Description file name. *Makefile* is assumed to be the name of a description file. A file name of − denotes the standard input. The contents of *makefile* override the built-in rules if they are present. Note that the space between **−f** and *makefile* **must** be present.

−p Print out the complete set of macro definitions and target descriptions.

−i Ignore error codes returned by invoked commands. This mode is also entered if the fake target name **.IGNORE** appears in the description file.

−k When a command returns nonzero status, abandon work on the current entry, but continue on other branches that do not depend on that entry.

−s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.

−r Do not use the built-in rules.

−n No execute mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed.

−b Compatibility mode for old (Version 7) *makefiles*.

−e Environment variables override assignments within *makefiles*.

−t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

−d Debug mode. Print out detailed information on files and times examined. (This is intended for debugging the *make* command itself.)

−q Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

The "built-in" dependency targets are:

.DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS Dependents of this target will not be removed when QUIT or INTERRUPT are hit.

.SILENT Same effect as the **−s** option.

.IGNORE Same effect as the **−i** option.

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **−f** option is present, **makefile**, **Makefile**, **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is −, the standard input is taken. More than one **−f** *makefile* argument pair may appear.

Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, followed by a colon (:), followed by a (possibly null) list of prerequisite files or dependencies. Pattern Matching Notation (see *regexp*(5)) is supported for the generation of file names as dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target, see the **Environment** section below about **SHELL**. The first line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by *make* (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments before the rules. Comments in the rules depend on the setting of the **SHELL** macro.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command can be the following: **-**, **@**, **-@**, or **@-**. If **@** is present, printing of the command is suppressed. If **-** is present, *make* ignores an error. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a **@**. The **-n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed, see discussion below of the **MAKEFLAGS** macro under **Environment**. Note that this feature does not work if **MAKE** is enclosed in braces, as in **${MAKE}**. The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **-i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains **-**. the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old *makefiles* (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of *make* assumed, if no command was specified explicitly, that the command was null.

INTERRUPT and QUIT cause the target to be deleted unless the target depends on the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any *makefile* and after the internal rules; thus, macro assignments in a *makefile* override environment variables. The **-e**

option causes the environment to override the macro assignments in a *makefile*.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **−f**, **−p**, and **−d**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **−n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make −n** recursively on a whole software system to see what would have been executed. This is because the **−n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the *makefiles* for a software project without actually doing anything.

Each of the commands in the rules is given to a shell to be executed. The shell that is used is determined by the **SHELL** environment variable, which is usually set to the shell with which the user logs in. To ensure the same shell is used each time a *makefile* is executed, the line:

SHELL=/bin/sh

should be put in the macro definition section of the *makefile*.

Include Lines

If the string *include* appears as the first seven letters of a line in a makefile, and it is followed by one or more space or tab characters, the rest of the line is assumed to be a file name and will be read by the current invocation after substituting for any macros.

Macros

Entries of the form *string1* = *string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1*[:*subst1*= [*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under **Libraries**.

Internal Macros

There are five internally maintained macros that are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module that is out-of-date with respect to the target, i.e., the "manufactured" dependent file name. Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc −c −O $*.c
```

or:

```
.c.o:
        cc −c −O $<
```

$?** The **$? macro is evaluated when explicit rules from the *makefile* are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules that must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member **file.o**.

Four of the five macros can have alternative forms. When an uppercase **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules that allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h

To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used (if using **/bin/sh** as a shell):

make −fp − 2>/dev/null </dev/null

The only peculiarity in this output is the (**null**) string that *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file, see *sccsfile*(4). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix, i.e., **".c:"**, is the definition of how to build *x* from *x***.c**. In effect, the other suffix is null. This is useful for building targets from only one source file, e.g., shell procedures, simple C programs.

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite.

The default list is:

.SUFFIXES: .o .c .y .l .s

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly:

```
pgm: a.o b.o
        cc a.o b.o −o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library that contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form *.XX***.a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @ @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $? @ @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

EXAMPLES

The following example creates an object file from the source code file:

```
make program.o
```

Make compiles the source code and creates an object file. Note that there must be a file such as **program.c** or **program.s** in the current working directory.

WARNINGS

Be wary of any file (such as an include file) whose access, modification, and last change times cannot be altered by the *make*-ing process. For example, if a program depends on an include file that in turn depends on another include file, and if one or both of these files are out-of-date, *make* will try to update these files each time it is run, thus unnecessarily re-*make*ing up-to-date files dependent on the include file. The solution is to manually update these files with the *touch*(1) command before running *make*. (Note that it is generally a bad idea to include the *touch*(1) command in your *makefile*, because it can cause *make* to update a program that otherwise did not need to be updated.)

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

File names with the characters **@ = : @** will not work.

Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across newlines in *make*.

The syntax **(lib(file1.o file2.o file3.o)** is illegal.

You cannot build **lib(file.o)** from **file.o**.

The macro **$(a:.o=.c~)** does not work.

There is a limit of 2500 characters, including the terminating new-line, for expanded dependency lines.

Make will not properly expand a macro within another macro when string substitution is involved.

DEPENDENCIES

NFS

WARNINGS

When comparing modification times of files located on different NFS servers, *make* behaves unpredictably if the clocks on the servers are unsynchronized.

FILES

[Mm]akefile and s.[Mm]akefile

SEE ALSO

cc(1), cd(1), lex(1), sh(1), yacc(1), environ(5), lang(5), regexp(5).

Make: A Program for Maintaining Computer Programs, tutorial in *HP-UX Concepts and Tutorials: Programming Environment.*

A Nutshell Handbook, Managing Projects With Make by Steve Talbot, Second Edition, O'Reilly & Associates, Inc., 1986.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in pattern matching notation.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *make* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

make: SVID2, XPG2, XPG3

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

REMARKS

The decryption facilities provided by this software are under control by the United States Government and cannot be exported without special licenses. These capabilities can be sold only to domestic customers at this time.

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and uppercase and lowercase letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

Makekey is intended for programs that perform encryption (e.g., *ed*(1) and *crypt*(1)). Usually, its input and output will be pipes.

SEE ALSO

crypt(1), ed(1), passwd(4).

NAME

man – find manual information by keywords; print out a manual page

SYNOPSIS

man **–k** *keyword ...*
man **–f** *file ...*
man [–] [*section*] *title ...*

DESCRIPTION

Man is a program that accesses information from the on-line version of the *HP-UX Reference*. It can be asked to provide one-line descriptions of commands specified by name, or to list all commands whose description contains any of a set of keywords. It can also provide on-line access to individual entries and sections of the printed manual.

When given the option **–k** and a set of keywords, *man* prints out a one-line synopsis of each manual section whose listing in the table of contents contains that keyword. The **–k** option requires that the file **/usr/lib/whatis** exist. The **/usr/lib/whatis** file can be created with *catman*(1M).

When given the option **–f** and a list of file names, *man* attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither **–k** nor **–f** is specified, *man* formats a specified set of manual pages. If a section specifier is given, *man* looks in that section of the manual for the given *titles. Section* is an arabic section number, for example, 3, which may be followed by a single letter classifier, for example, 3c, indicating a library routine in section 3. If *section* is omitted, *man* searches all sections of the manual, giving preference to commands over subroutines in system libraries, and printing the first section it finds, if any. If any *title* is longer than 11 characters, *man* first searches for the full-length *title.* If not found, *title* is truncated to 11 characters to ensure that there will be room for the *section*. The files in the **/usr/man** directories are normally installed truncated to 11 characters to work on short filename systems with room allowed for the suffix.

If the standard output is a teletype, or if the flag – is given, *man* pipes its output through *more*(1), with the **–s** option, to stop after each page.

Man searches in three directories for the specified manual entry. First *man* searches in **/usr/man**, then in **/usr/contrib/man**, and finally in **/usr/local/man**. Within each of these directories, *man* searches in the **cat*.Z** subdirectories, the **man*.Z** subdirectories, the **cat*** subdirectories, and the **man*** subdirectories. The **man*.Z** and **man*** directories contain the *nroff*(1) source for the entries. The **cat*.Z** and **cat*** directories contain the formatted versions of the entries. The **man*.Z** and **cat.Z** directories contain entries in compressed form and are uncompressed before being processed for printing or display.

If LANG environment variable is set to any valid language name defined by *langid*(5), *man* searches in three additional directories for the manual entry before searching in **/usr/man**. First, *man* searches in **/usr/man/$LANG**, then in **/usr/contrib/man/$LANG**, and then in **/usr/local/man/$LANG**. Therefore, the native language manual pages will be displayed if they are present and installed properly in the system.

Man uses the most recent version that it finds in the subdirectories searched. If the most recent version is in:

man*.Z the entry is uncompressed, formatted, and displayed. If the **cat*.Z** directory exists, the formatted entry is compressed and installed in **cat*.Z**. If the **cat*** directory exists, the formatted entry is installed in **cat***.

cat*.Z the entry is uncompressed and displayed.

man* the entry is formatted, and displayed. If the **cat*.Z** directory exists, it is compressed, and installed in **cat*.Z**. If the **cat*** directory exists, the formatted

entry is installed in **cat***.

cat* the entry is displayed.

If only the **cat*** or **cat*.Z** subdirectory is present and/or *nroff*(1) is not installed, only those pages that have already been formatted are displayable.

If you choose to have the formatted entries on your system, run *catman*(1M) with the default, which creates the **cat*.Z** directories (after removing any **cat*** directories that exist on your system) and also creates the file **/usr/lib/whatis** used by the *man* **−k** option. If you choose to have the **cat*** directories, it would be space-saving to remove any **cat*.Z** directories that may exist on your system. Beware that *man* will update both directories (**cat*** and **cat*.Z**) if they both exist.

Special Manual Entries

Some situations may require creation of manual entries for local use or distribution by third-party software suppliers. The manual formatting macros have been structured to redefine page footers so that manual entries not originating from Hewlett-Packard Company do not show the HP name in the footer, unlike previous releases. For more information about this change and a description of the manual formatting macros used with *nroff* or *troff*, see *man*(5).

EXAMPLES

The following command lists the manual page entries that contain the word **grep** in their respective NAME lines.

man -k grep

The following is printed after the above command is entered:

grep, egrep, fgrep (1) − search a pattern

The following prints the entire manual page for *grep*(1):

man grep

FILES

/usr/lib/whatis	keyword database
/usr/man/cat*[.Z]/*	formatted manual pages [compressed]
/usr/man/man*[.Z]/*	raw (*nroff*(1) source) manual pages [compressed]
/usr/contrib/man/cat*[.Z]/*	
/usr/contrib/man/man*[.Z]/*	
/usr/local/man/cat*[.Z]/*	
/usr/local/man/man*[.Z]/*	
/usr/man/$LANG/cat*[.Z]/*	formatted native language manual pages [compressed]
/usr/man/$LANG/man*[.Z]/*	raw (*nroff*(1) source) native language manual pages [compressed]
/usr/contrib/man/$LANG/cat*[.Z]/*	
/usr/contrib/man/$LANG/man*[.Z]/*	
/usr/local/man/$LANG/cat*[.Z]/*	
/usr/local/man/$LANG/man*[.Z]/*	

SEE ALSO

catman(1M), col(1), compress(1), fixman(1), more(1), col(1), man(5).

WARNINGS

Manual entries are structured such that they can be printed on a phototypesetter or conventional line printer and screen display devices. However, due to line printer and display device limitations, some information may be lost.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed. LANG is also used to determine the search path (as described above).

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *man* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

mediainit – initialize hard disk, flexible disk, or cartridge tape media

SYNOPSIS

mediainit [**–vr**] [**–f** *fmt_optn*] [**–i** *interleave*] *pathname*

DESCRIPTION

Mediainit initializes mass storage media by formatting the media, writing and reading test patterns to verify media integrity, then sparing any defective blocks found. This process prepares the disk or tape for error-free operation. Initialization destroys all existing user data in the area being initialized.

The following command options are recognized. They can be specified in any order, but all must precede the *pathname*. Options without parameters can be listed individually or grouped together. Options with parameters must be listed individually, but white space between the option and its parameter is discretionary.

Options

–v Normally, *mediainit* provides only fatal error messages, and they are directed to diagnostic output (stderr). The **–v** (verbose) option sends device-specific information related to low-level operation of *mediainit* to standard output (stdout). This option is most useful to trained service personnel because it usually requires detailed knowledge of device operation before the information can be interpreted correctly.

–r The **–r** (re-certify) option forces a complete tape certification whether or not the tape has been certified previously. All record of any previously spared blocks is discarded, so any bad blocks will have to be rediscovered. This option should be used only if: (a) it is suspected that numerous blocks on the tape have been spared which should not have been, or (b) it is necessary to destroy (overwrite) all previous data on the tape.

–f *fmt_optn* The format option is a device-specific number in the range **0** through **239.** It is intended solely for use with certain SS/80 devices that support multiple media formats (independent from interleave factor). For example, certain microfloppy drives support 256-, 512-, and 1024-byte sectors. *Mediainit* passes any supplied format option directly through to the device. The device then either accepts the format option if it is supported or rejects it if it is not supported. Refer to device operating manuals for additional information. The default format option is **0.**

–i *interleave* The interleave factor, *interleave*, refers to the relationship between sequential logical records and sequential physical records. It defines the number of physical records on the media that lie between the beginning points of two consecutively numbered logical records. The choice of interleave factor can have a substantial impact on disk performance. For CS/80 and SS/80 drives, consult the appropriate device operating manual for details. For Amigo drives, see DEPENDENCIES.

pathname *Pathname* is the path name to the character (raw) device special file associated with the device unit or volume that is to be initialized. *Mediainit* aborts if you lack either read or write permission to the device special file, or if the device is currently open for any other process. This prevents accidental initialization of the root device or any mounted volume. See DEPENDENCIES for additional Series 800 requirements. *Mediainit* locks the unit or volume being initialized so that no other processes can access it.

When a given CS/80 or SS/80 device contains multiple units or a given unit contains multiple volumes as defined by the drive controller, any available unit or volume associated with that controller can be initialized, independent of other units and volumes that share the same controller. Thus, you can initialize one unit or volume to any format or interleave factor without affecting formats or data on companion units or volumes. However, be aware that the entire unit or volume (as defined by the drive controller) is initialized without considering the possibility that it may be subdivided into smaller structures by the the operating software. When such structures exist, unexpected loss of data is possible.

Mediainit dominates controller resources and limits access by competing processes to other units or volumes sharing the same controller. If other simultaneous processes need access to the same controller, some access degradation can be expected until initialization is complete; especially if you are initializing a tape cartridge in a drive that shares the root disk controller. See Series 800 DEPENDENCIES for additional Series 800 information.

In general, *mediainit* attempts to carefully check any **−f** (format option) or **−i** (interleave options) supplied, and aborts if an option is out of range or inappropriate for the media being initialized. Specifying an interleave factor or format option value of **0** has the same effect as not specifying the option at all.

For disks that support interleave factors, the acceptable range is usually **1** (no interleave) through *N*−**1**, where *N* is the number of sectors per track. With SS/80 hard disks, the optimum interleave factor is usually determined by the speed (normal or high) of the HP-IB interface card used and whether DMA is present in the system. The optimum interleave factor for SS/80 flexible disk drives is usually a constant (often **2**), and is independent of the type of HP-IB interface used. The optimum interleave factor for CS/80 disks is usually **1** and is also usually not related to the type of HP-IB interface being used. In any case, refer to the appropriate device operating manual for recommended values.

If a disk being initialized requires an interleave factor but none is specified, *mediainit* provides an appropriate, though not necessarily optimum default. For CS/80 and SS/80 disks, *mediainit* uses whatever the device reports as its current interleave factor. SS/80 floppy drives report their minimum (usually best) interleave factor, if the currently installed media is unformatted.

When a given device supports format options, the allowable range of interleave factors may be related to the specified format option. In such instances, *mediainit* cannot check the interleave factor if one is specified.

Notes

Most types of mass storage media must be initialized before they can be used. HP hard disks, flexible disks, and cartridge tapes require some form of initialization, but 9-track tapes do not. Initialization usually involves formatting the media, writing and reading test patterns, then sparing any defective blocks. Depending upon the media and device type, none, some, or all of the initialization process may have been performed at the factory. *Mediainit* completes whatever steps are appropriate to prepare the media for error-free operation.

Most HP hard disks are formatted and exhaustively tested at the factory by use of a process more thorough but also more time-consuming than appropriate for *mediainit*. However, *mediainit* is still valuable for ensuring the integrity of the media after factory shipment, formatting with the correct interleave factor, and sparing any blocks which may have become defective since original factory testing was performed.

HP flexible disks are not usually formatted prior to shipment, so they must undergo the entire initialization process before they can be used.

All HP CS/80 cartridge tapes are certified and formatted prior to shipment from the factory, When a tape is certified, it is thoroughly tested and defective blocks are spared. *Mediainit* usually certifies a tape only if it has not been certified previously. If the tape has been previously

certified and spared, *mediainit* usually reorganizes the tape's spare block table, retaining any previous spares, and optimizing their assignment for maximum performance under sequential access. Reorganizing the spare block table takes only a few seconds, whereas complete certification takes about a half-hour for 150-foot tapes, and over an hour for 600-foot tapes.

HP CS/80 cartridge tape drives have a feature called "auto-sparing". If under normal usage the drive has trouble reading a block, the drive logs the fact and automatically spares out that block the next time data is written to it. Thus, as a tape is used any marginal blocks that were not spared during certification are spared automatically if they cause problems. This sparing is automatic within the device, and is totally independent of *mediainit*.

Reorganization of a tape's spare block table technically renders any existing data undefined, but the data is not usually destroyed by overwriting. To ensure that old tape data is destroyed, which is useful for security, complete tape re-certification can be forced with the **-r** option.

Some applications may require that a file system be placed on the media before use. *Mediainit* does not create a file system; it only prepares media for writing and reading. If such a file system is required, other utilities such as *newfs*(1M), *lifinit*(1), or *mkfs*(1M) must be invoked after running *mediainit*.

RETURN VALUE

Mediainit returns a value of **0** upon successful completion, a value of **1** if there was a device-related error, or a value of **2** if there was a syntax-related error.

ERRORS

Appropriate error messages are printed out to stderr during the execution of *mediainit*.

EXAMPLES

The following example formats an HP 9122 SS/80 3-1/2 inch flexible disk with an interleave factor of 2, 1024-byte sectors, and double-sided HP format:

mediainit -i 2 -f 3 /dev/rdsk/9122

WARNINGS

Aborting *mediainit* is likely to leave the medium in a corrupt state, even if it was previously initialized. To recover, the initialization must be restarted.

DEPENDENCIES

Series 300

Series 300 systems support various Amigo disk drives. Acceptable interleave factors for Amigo devices are as follows:

Device	Range	Default
HP 9895 SS/DS	1 - 29	2
HP 8290X	1 - 15	3
HP 9121	1 - 15	2
HP 9133V	na	9
HP 9133XV	na	9
HP 9134XV	na	9

Series 800

Pathname must be a device special file whose minor number of the device being initialized has the diagnostic bit set (the diagnostic bit is the most significant bit in the minor number). For device special files with the diagnostic bit set, the section number is meaningless. The entire device is accessed.

For a device that contains multiple units on a single controller, each unit can be initialized independently from any other unit. It should be noted, however, that *mediainit* requires that there be no other processes accessing the device before initialization begins,

regardless of which unit is being initialized. If there are accesses currently in progress, *mediainit* aborts. During the initialization process, *open*(2) rejects all other accesses to the device being initialized, producing the error **EACCES**.

AUTHOR

Mediainit was developed by HP.

SEE ALSO

mkfs(1M), newfs(1M), lifinit(1).

NAME

merge – three-way file merge

SYNOPSIS

merge [**–p**] file1 file2 file3

DESCRIPTION

Merge combines two files that are revisions of a single original file. The original file is *file2*, and the revised files are *file1* and *file3*. *Merge* identifies all changes that lead from *file2* to *file3* and from *file2* to *file1*, and deposits the merged text into *file1*. If the **–p** option is used, the result goes to standard output instead of *file1*.

An overlap occurs if both *file1* and *file3* have changes in the same place. *Merge* prints how many overlaps occurred, and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, the user should edit the result in *file1* and delete one of the alternatives.

This command is particularly useful for revision control, especially if *file1* and *file3* are the ends of two branches that have *file2* as a common ancestor.

EXAMPLES

A typical use for *merge* would be as follows:

To merge an RCS branch into the trunk, first check out the three different versions from RCS (see *co*(1)) and rename them for their revision numbers: 5.2, 5.11, and 5.2.3.3. File 5.2.3.3 is the end of an RCS branch that split off the trunk at file 5.2. File 5.11 is the latest version on the trunk. Therefore, it is also a revision of the "original" file, 5.2. Now, merge the branch into the trunk with the command:

```
merge 5.11 5.2 5.2.3.3
```

File 5.11 now contains all changes done on the branch and the trunk and indicates all overlapping changes. The overlaps will have to be corrected by using an editor, and then file 5.11 can be checked back in (see *ci*(1)).

WARNINGS

Merge uses the system editor *ed*(1). Therefore, the file size limits of *ed*(1) apply to *merge*.

AUTHOR

Merge was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

diff3(1), diff(1), rcsmerge(1), co(1).

NAME

mesg – permit or deny messages to terminal

SYNOPSIS

mesg [**n**] [**y**]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

STANDARDS CONFORMANCE

mesg: SVID2, XPG2, XPG3

NAME

mkdir – make a directory

SYNOPSIS

mkdir [*–p*] *dirname* ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

If the –p option is specified, intermediate directories are created as necessary. Otherwise, the full path prefix of *dirname* must already exist.

Mkdir requires write permission in the parent directory.

EXAMPLES

The following command creates the directory **gem** beneath directory **raw:**

mkdir raw/gem

Note that directory **raw** must already exist, and it must be a subdirectory of the current working directory.

SEE ALSO

rm(1), sh(1), umask(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *mkdir* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

mkdir: SVID2, XPG2, XPG3

NAME

mkfifo – make FIFO (named pipe) special files

SYNOPSIS

mkfifo filename ...

DESCRIPTION

Mkfifo creates the FIFO special files named by its operand list. The operands are taken sequentially in the order specified and, if the user has write permission in the appropriate directory, the FIFO is created with permissions 0666 modified by the user's file mode creation mask (see *umask*(2)).

The specific actions performed are equivalent to calling

mkfifo(*filename***, 0666)**

for each filename in the operand list (see *mkfifo*(2)).

EXAMPLES

The following command creates a FIFO special file named **peacepipe** in the current directory:

mkfifo peacepipe

SEE ALSO

mkfifo(2), mknod(1M), umask(1).

DIAGNOSTICS

Mkfifo returns exit code zero if invoked with at least one operand and if all FIFO special files were created successfully. Otherwise, it prints a diagnostic message and returns a non-zero exit code.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *mkfifo* behaves as if all internationalization variables are set to "C". See *environ*(5).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

mkfifo: XPG3

NAME

mkmf – make a makefile

SYNOPSIS

mkmf [**–acdil**] [**–f** *makefile*] [**–F** *template*] [**–M** *language*] [*macroname=value* ...]

DESCRIPTION

Mkmf creates a makefile that informs the *make*(1) command how to construct and maintain programs and libraries. After gathering up all the source code file names in the current working directory and inserting them into the makefile, *mkmf* scans source code files for included files and generates dependency information that is appended to the makefile. Source code files are identified by their file name suffixes. *Mkmf* recognizes the following suffixes:

.c	C
.f	Fortran
.h	Include files
.i	Pascal include files
.l	Lex or Lisp
.o	Object files
.p	Pascal
.r	Ratfor
.s	Assembler
.y	Yacc

Mkmf checks for an existing makefile before creating one. If no **–f** option is present, *mkmf* tries the makefiles **makefile** and **Makefile**, respectively.

After the makefile has been created, arbitrary changes can be made using a text editor. *Mkmf* can also be used to re-edit the macro definitions in the makefile, regardless of changes that may have been made since it was created.

By default, *mkmf* creates a program makefile. To create a makefile that handles libraries, the **–l** option must be used.

Make Requests

Given a makefile created by *mkmf, make* recognizes the following requests:

all	Compile and load a program or library.
clean	Remove all object and core files.
clobber	Remove all files that can be regenerated.
depend	Update included file dependencies in a makefile.
echo	List the names of the source code files on standard output.
extract	Extract all the object files from the library and place them in the same directory as the source code files. The library is not altered.
index	Print an index of functions on standard output.
install	Compile and load the program or library and move it to its destination directory.
print	Print source code files on standard output.
tags	Create a tags file for the *ex*(1) editor, for C, Pascal, and Fortran source code files.
update	Recompile only if there are source code files that are newer than the program or library, link and install the program or library.

Several requests may be given simultaneously. For example, to compile and link a program, move the program to its destination directory, and remove any unnecessary object files:

make install clean

Macro Definitions

Mkmf understands the following macro definitions:

CFLAGS	C compiler flags. After searching for included files in the directory currently being processed, *mkmf* searches in directories named in **−I** compiler options and then in the **usr/include** directory.
DEST	Directory where the program or library is to be installed.
EXTHDRS	List of included files external to the current directory. *Mkmf* automatically updates this macro definition in the makefile if dependency information is being generated.
FFLAGS	Fortran compiler flags. After searching for included files in the directory currently being processed, *mkmf* searches in directories named in **−I** compiler options and then in the **/usr/include** directory.
HDRS	List of included files in the current directory. *Mkmf* automatically updates this macro definition in the makefile.
INSTALL	Installation program name.
LD	Link editor name.
LDFLAGS	Link editor flags.
LIBRARY	Library name. This macro also implies the **−l** option.
LIBS	List of libraries needed by the link editor to resolve external references.
MAKEFILE	Makefile name.
OBJS	List of object files. *Mkmf* automatically updates this macro definition in the makefile.
PROGRAM	Program name.
SRCS	List of source code files. *Mkmf* automatically updates this macro definition in the makefile.
SUFFIX	List of additional file name suffixes for *mkmf* to know about.

Both these and any other macro definitions already within the makefile may be replaced by definitions on the command line in the form *macroname=value*. For example, to change the C compiler flags and the program name, type the following line:

mkmf "CFLAGS=−I../include −O" PROGRAM=mkmf

Note that macro definitions such as CFLAGS with blanks in them must be enclosed in double quote (") marks.

File Name Suffixes

Mkmf can recognize additional file name suffixes, or ignore ones that it already recognizes, by specifying suffix descriptions in the SUFFIX macro definition. Each suffix description takes the form *'.suffix:tI'* where *t* is a character indicating the contents of the file (**s** = source file, **o** = object file, **h** = header file, **x** = executable file) and *I* is an optional character indicating the include syntax for header files (C = C syntax, F = Fortran, and Ratfor syntax, P = Pascal syntax). The following table describes the default configuration for *mkmf:*

.c:sC C
.f:sF Fortran
.h:h Include files
.i:h Pascal include files
.l:sC Lex or Lisp
.o:o Object files
.p:sP Pascal
.r:sF Ratfor
.s:s Assembler
.y:sC
Yacc

For example, to change the object file suffix to **.obj**, undefine the Pascal include file suffix, and prevent Fortran files from being scanned for included files, the SUFFIX macro definition could be:

"SUFFIX = .obj:o .i: .f:s"

Include Statement Syntax

The syntax of include statements for C, Fortran, and Pascal source code are of the form:

C: #include "*filename*"
#include <*filename*>
where # must be the first character in the line.

Fortran:
$include '*filename*'$
$INCLUDE '*filename*'$
where $ must be the first character in the line. Alternatively, the $ may be omitted if the include statement starts in column 7. In either case the trailing $ can be omitted.

Pascal:
$include '*filename*'$
$search '*filename*'$
$INCLUDE '*filename*'$
$SEARCH '*filename*'$
where $ must be the first character in the line and the trailing $ is optional.

User-Defined Templates

If *mkmf* cannot find a makefile within the current directory, it normally uses one of the standard makefile templates, 'C.p' or 'C.l', in **/usr/lib/mf** unless the user has alternative 'C.p' or 'C.l' template files in a directory $PROJECT/lib/mf where $PROJECT is the absolute path name of the directory assigned to the PROJECT environment variable.

Options

−a Include source files beginning with a . in the makefile.

−c Suppress 'creating *makefile* from ...' message.

−d Turn off scanning of source code for **include** files. Old dependency information is left untouched in the makefile.

−f *makefile*
Specify an alternative *makefile* file name. The default file name is **Makefile**.

−i Prompt the user for the name of the program or library and the directory where it is to be installed. If a carriage return is typed in response to each of these queries, *mkmf* assumes that the default program name is **a.out** or the default library name is **lib.a**, and the destination directory is the current directory.

-l Force the makefile to be a library makefile.

-F *template*
Specify an alternative makefile template path name. The path name can be relative or absolute.

-M *language*
Specify an alternative *language*-specific makefile template. The default language is C and the corresponding program and library makefile templates are 'C.p' and 'C.l', respectively. *Mkmf* looks for these templates in **/usr/lib/mf** or **$PROJECT/lib/mf**.

DIAGNOSTICS

Exit status 0 is normal. Exit status 1 indicates an error.

WARNINGS

The name of the makefile is included as a macro definition within the makefile and must be changed if the makefile is renamed.

Since executable files are dependent on libraries, standard library abbreviations must be expanded to full path names within the LIBS macro definition in the makefile.

Generated dependency information appears after a line in the makefile beginning with **###**. This line must not be removed, nor must any other information be inserted in the makefile below this line.

The name of a program or library must not conflict with any predefined target names in a makefile. It is especially important to avoid the the name **update**. Otherwise, *make*(1) recursively executes itself an infinite number of times.

FILES

/usr/lib/mf/C.p	Standard program makefile template
/usr/lib/mf/C.l	Standard library makefile template
$PROJECT/lib/mf/C.p	User-defined program makefile template
$PROJECT/lib/mf/C.l	User-defined library makefile template

SEE ALSO

ar(1), ctags(1), ld(1), make(1).

"Make: A Program for Maintaining Computer Programs", *HP-UX Concepts and Tutorials*, Feldman, S.I., vol. 2 (Program Development and Maintenance).

"Automatic Generation of Make Dependencies", *Software–Practice and Experience*, Walden, K., vol. 14, no. 6, pp. 575-585, June 1984.

AUTHOR

Mkmf was developed by the University of California, Berkeley.

NAME

mkstr – extract error messages from C source into a file

SYNOPSIS

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr examines a C program and creates a file containing error message strings used by the program. Programs with many error diagnostics can be made much smaller by referring to places in the file, and reduce system overhead in running the program.

Mkstr processes each of the specified *file*s, placing a revised version of each in a file whose name consists of the specified *prefix* concatenated in front of the original name. A typical usage of *mkstr* would be

```
mkstr mystrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *mystrings* and revised copies of the source for these files to be placed in files whose names are prefixed with *xx*.

When processing the error messages in the source for transfer to the message file, *mkstr* searches for the string **error("** in the input file. Each time it is encountered, the C string starting after the leading quote is placed in the message file, followed by a null character and a new-line character. The null character terminates the message so that it can be easily used when retrieved, and the new-line character makes it possible to conveniently *cat* the error message file to review its contents.

The modified copy of the input file is identical to the original, except that each occurrence of any string that was moved to the error message file is replaced by an offset pointer usable by *lseek* to retrieve the message.

If the command line includes the optional –, extracted error messages are placed at the end of the specified message file instead of overwriting it. This enables you to process individual files that are part of larger programs that have been previously processed by *mkstr* without reprocessing all the files.

All functions used by the original program whose names end in "error" that also can take a constant string as their first argument should be rewritten so that they search for the string in the error message file.

For example, a program based on the previous example usage would resemble the following:

```
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>

char errfile[] = "mystrings";

error(offset, a2, a3, a4)
int offset, a1, a2, a3;
{
    char msg[256];
    static int fd = -1;

        if (fd < 0) {
            fd = open(errfile, O_RDONLY);
```

```
            if (fd < 0) {
                perror(errfile);
                exit(1);
            }
        }

        if (lseek(fd, (off_t) offset, 0) | | read(fd, msg, 256) <= 0) {
            printf("? Can't find error message in %s:\n", errfile);
            perror(errfile);
            exit(1);
        }

        printf(msg, a1, a2, a3);
}
```

SEE ALSO

lseek(2), perror(3C), xstr(1).

BUGS

Strings in calls to functions whose names end in 'error', notably *perror*(3C), may be replaced with offsets by *mkstr*.

Calls to error functions whose first argument is not a string constant are left unmodified without warning.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of comments and string literals as single- and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *mkstr* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported within file names, comments, and string literals.

NAME

mktemp – make a name for a temporary file

SYNOPSIS

mktemp [**–c**] [**–d** *directory_name*] [**–p** *prefix*]

DESCRIPTION

Mktemp makes a name that is suitable for use as the pathname of a temporary file, and writes that name to the standard output. The name is chosen such that it does not duplicate the name of an existing file. If the **–c** option is specified, a zero length file is created with the generated name.

The name generated by *mktemp* is the concatenation of a directory name, a slash (/), the value of the LOGNAME environment variable truncated to {NAME_MAX} – 6 characters, and the process id of the invoking process.

The directory name is chosen as follows:

(1) If the **–d** option is specified, *directory_name* is used.

(2) Otherwise, if the TMPDIR environment variable is set and a string that would yield a unique name can be obtained by using the value of that variable as a directory name, this value is used.

(3) Otherwise, if a string that would yield a unique name can be obtained using **/tmp** as the directory, **/tmp** is used.

(4) Otherwise, **.** (current directory) is used.

If the **–p** option is specified, *prefix* is used instead of the value of the LOGNAME environment variable for name generation.

SEE ALSO

mktemp(3C), umask(1).

DIAGNOSTICS

Mktemp returns exit code 0 on successful completion and non zero if syntax, file access, or file creation errors were encountered or a unique pathname could not be generated.

NAME

mm, osdd − print documents formatted with the mm macros

SYNOPSIS

mm [*options*] [*files*]

osdd [*options*] [*files*]

DESCRIPTION

Mm can be used to type out documents using *nroff*(1) and the *mm* text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn*(1) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff*(1) and MM are generated, depending on the options selected.

Osdd is equivalent to the command **mm -mosd**.

Options for *mm* are given below. Any other arguments or flags (e.g., **−rC3**) are passed to *nroff*(1) or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

−T*term* Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable **$TERM** from the environment (see *profile*(4) and *environ*(5)) as the value of *term*, if **$TERM** is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.

−12 Indicates that the document is to be produced in 12-pitch. May be used when **$TERM** is set to one of **300, 300s, 450,** and **1620.** (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)

−c Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620,** and **X**.

−e Causes *mm* to invoke *neqn*.

−t Causes *mm* to invoke *tbl*(1).

−E Invokes the **−e** option of *nroff*.

−y Causes *mm* to use the non-compacted version of the macros (see *mm*(5)).

As an example (assuming that the shell variable **$TERM** is set in the environment to **450**), the two command lines below are equivalent:

```
mm −t −rC3 −12 ghh*
tbl ghh* | nroff −cm −T450−12 −h −rC3
```

Mm reads the standard input when − is specified instead of any file names. (Mentioning other files together with − leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm −
```

HINTS

1. *Mm* invokes *nroff* with the **−h** flag. With this flag, *nroff* assumes that the terminal has tabs set every 8 character positions.
2. Use the **−o***list* option of *nroff* to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **−e**, **−t**, and − options, *together* with the **−o***list* option of *nroff* may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the **−s** option of *nroff* (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The **−s** option of *nroff* does not work with the **−c** option of *mm*, or if *mm* automatically invokes *col*(1) (see **−c** option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you'll get (often subtle) garbage; however, if you are redirecting output into a file, use the **−T37** option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

col(1), env(1), nroff(1), tbl(1), profile(4), mm(5), term(5).

MM–Memorandum Macros in *HP-UX Concepts and Tutorials: Text Formatters.*

DIAGNOSTICS

mm "mm: no input file" if none of the arguments is a readable file and *mm* is not used as a filter.

NAME

more, page – file perusal filter for crt viewing

SYNOPSIS

more [**−n**] [**−cdflsu**] [+linenumber] [+/pattern] [name ...]

page [more options]

REMARKS:

Pg(1) is preferred in some standards and has some added functionality, but does not support character highlighting.

DESCRIPTION

More is a filter which allows examination of continuous text, one screenful at a time, on a soft-copy terminal. It is quite similar to *pg*(1). and is retained for primarily for backward compatibility. *More* normally pauses after each screenful, printing – –**More**– – at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilites are enumerated later.

More supports the Basic Regular Expression syntax (see *regexp*(5)).

The command line options are:

−*n* An integer which is the size (in lines) of the window which *more* will use instead of the default.

−c *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

−d *More* will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

−f This causes *more* to count logical lines, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen postions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

−l Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

−s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

−u Normally, *more* will handle underlining and bold such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining, else stand-out mode, for underlined information in the source file. If the terminal can perform stand-out, more uses that mode for bold information. The **−u** option suppresses this processing, as do the "ul" and "os" terminfo flags.

+linenumber Start up at *linenumber.*

+pattern Start up two lines before the line containing the regular expression *pattern.*

If the program is invoked as *page,* then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

More uses terminfo descriptor files to determine terminal characteristics, and to determine the default window size, see *term*(4). On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the **−c** mode of operation, the shell command sequence *MORE='-c' ; export MORE* or the *csh* command *setenv MORE -c* would cause all invocations of *more*, including invocations by programs such as *man* and *msgs*, to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.profile* or *.cshrc* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the **−−More−−** prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space> display *i* more lines, (or another screenful if no argument is given).

^D display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d same as ^D (control-D).

*i*z same as typing a space except that *i*, if present, becomes the new window size.

*i*s skip *i* lines and print a screenful of lines.

*i*f skip *i* screenfuls and print a screenful of lines.

"q or Q" Exit from *more.*

= Display the current line number.

v Start up the editor *vi* at the current line.

h Help command; give a description of all the *more* commands.

i/expr search for the *i*-th occurrence of the regular expression *expr.* If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command invoke a shell with *command*. The characters "%" and "!" in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, "%" is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense).

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

":q or :Q" exit from *more* (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the – –**More**– –(*xx*%).

At any time when output is being sent to the terminal, the user can hit the quit key (normally control–\). *More* will stop sending output, and will display the usual – –**More**– – prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

nroff –ms +2 doc.n | more -s

EXAMPLES

The following command displays the file **stuff** in a fifteen line-window and converts multiple adjacent blank lines into only one blank line:

more -s -15 stuff

FILES

/usr/lib/more.help	help file
/usr/lib/terminfo/?/*	compiled terminal capability data base

VARIABLES

MORE	Default paging mode.

AUTHOR

More was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

csh(1), man(1), pg(1), sh(1), term(4), terminfo(4), environ(5), lang(5), regexp(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as printable, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *more* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**–t** *tapename*] *command* [*count*]

DESCRIPTION

Mt is used to give commands to the tape drive. If *tapename* is not specified, the environment variable TAPE is used; if TAPE is not defined, the default drive is used.

Mt winds the tape in the requested direction (forward or backward), stopping after the specified *count* EOF marks or records are passed. If *count* is not specified, one is assumed. Each EOF mark counts as one record. When winding backwards, the tape always stops at the BOT marker, regardless of the number remaining in *count*.

Mt accepts the following *commands*:

eof	Write *count* EOF marks.
fsf	Forward space *count* files.
fsr	Forward space *count* records.
bsf	Backward space *count* files.
bsr	Backward space *count* records.
rew	Rewind tape.
offl	Rewind tape and go offline.

Spacing operations (back or forward space file or record) leave the tape positioned past the object being spaced to in the direction of motion. That is, backspacing a file leaves the the tape positioned before the file mark, forward spacing a file leaves the tape positioned after the file mark. This is consistent with all classical usage on tapes.

FILES

/dev/rmt/*	Raw magnetic tape interface
/dev/rmt/0mn	Default tape interface

WARNINGS

Only raw, no rewind, Berkeley-style devices can be specified.

It is possible to wind the tape beyond the EOT marker and off the end of the reel.

AUTHOR

Mt was developed by the University of California, Berkeley.

SEE ALSO

dd(1), mt(7).

NAME

mv – move or rename files and directories

SYNOPSIS

mv [**–f**] [**–i**] *file1 new_file*
mv [**–f**] [**–i**] *file1* [*file2* ...] *dest_directory*
mv [**–f**] [**–i**] *directory1* [*directory2* ...] *dest_directory*

DESCRIPTION

Mv moves:

- *file1* to new or existing *new_file*,
- *file1* to existing *dest_directory*,
- *file1*, *file2*, ... to existing *dest_directory*,
- directory subtree *directory1*, to new or existing *dest_directory*. or
- multiple directory subtrees *directory1*, *directory2*, ... to new or existing *dest_directory*.

Moving *file1* to *new_file* is used to relocate a file within the file system or to rename a file within a directory. When destination is a directory, one or more files are moved into that directory. If two or more files are moved, the destination must be a directory. When moving a single file to a new file, if *new_file* exists, its contents are destroyed.

If the access permissions of the destination *dest_directory* or existing destination file *new_file* forbid writing, *mv* asks permission to overwrite the file. This is done by printing the mode (see *chmod*(2) and Access Control Lists below), followed by the first letters of the words *yes* and *no* in the current native language, prompting for a response, and reading one line from the standard input. If the response is affirmative and is permissible, the operation occurs; if not, the command proceeds to the next source file, if any.

If *file1* is a file and *new_file* is a link to another file with other links, the other links remain and *new_file* becomes a new file. If *file1* is a file with links or a link to a file, the existing file or link remains intact, but with the new name *new_file* which may or may not be in the directory where *file1* resided, depending on directory pathnames used in the *mv* command. The last access and modification times of the file or files being moved remain unchanged.

Options

–f Perform *mv* commands without prompting for permission. This option is assumed when the standard input is not a terminal.

–i Causes *mv* to write a prompt to standard output before moving a file that would overwrite an existing file. If the response from the standard input is affirmative, the file is moved if permissions allow the move.

Access Control Lists (ACLs)

If optional ACL entries are associated with *new_file*, *mv* displays a plus sign (+) after the access mode when asking permission to overwrite the file.

If *new_file* is a new file, it inherits the access control list of *file1*, altered to reflect any difference in ownership between the two files (see *acl*(5)).

EXAMPLES

Rename a file in the current directory:

mv *old_filename new_filename*

Rename a directory in the current directory:

mv *old_dirname new_dirname*

Move directory *sourcedir* and its contents to a new location (*targetdir*) in the file system.

mv *sourcedir targetdir*

This results in a subdirectory named *sourcedir* located in directory *targetdir*.

Move all files and directories (including links) in current directory to a new location underneath *targetdir*:

mv * *targetdir*

Move all files and directories (including links) in *sourcedir* to a new location underneath *targetdir* (*sourcedir* and *targetdir* are in separate directory paths):

mv *sourcedir*/* *targetdir*

WARNINGS

If *file1* and *new_file* exist on different file systems, *mv* copies the file and deletes the original. In this case the mover becomes the owner and any linking relationship with other files is lost. *Mv* cannot carry hard links across file systems.

The *mv* command cannot be used to perform the following operations:

- Rename either the current working directory or its parent directory using the "." or ".." notation,
- Rename a directory to a new name identical to the name of a file contained in the same parent directory.

DEPENDENCIES

RFA and NFS

Access control lists of networked files are summarized (as returned in *st_mode* by *stat*(2)), but not copied to the new file. When using *mv* on such files, a + is not printed after the mode value when asking for permission to overwrite a file.

AUTHOR

Mv was developed by AT&T, the University of California, Berkeley and HP.

SEE ALSO

cp(1), cpio(1), ln(1), rm(1), link(1M), lstat(2), readlink(2), stat(2), symlink(2), symlink(4), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG and LC_CTYPE determine the local language equivalent of y (for yes/no queries).

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *mv* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

mv: SVID2, XPG2, XPG3

NAME

nawk – pattern-directed scanning and processing language

SYNOPSIS

nawk [**–F***fs*] [*prog* | **–f** *file* ...] [*file* ...]

Remarks:

This is a more recent version of the *awk*(1) program.

DESCRIPTION

Nawk scans each input *file* for lines that match any of a set of patterns specified literally in *prog* or in a file specified as **–f** *file*. With each pattern there can be an associated action that is to be performed when a line in *file* matches the pattern. Each line is matched against the pattern portion of every pattern-action statement, and the associated action is performed for each matched pattern. The file name – means the standard input. Any *file* of the form *var=value* is treated as an assignment, not a filename.

An input line is made up of fields separated by white space, or by regular expression **FS**. The fields are denoted **$1**, **$2**, ...; **$0** refers to the entire line.

Statements

A pattern-action statement has the form:

pattern { *action* }

A missing { *action* } means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

An action is a sequence of statements. A statement can be one of the following:

if(*expression*) *statement* [**else** *statement*]
while(*expression* **)** *statement*
for(*expression* **;** *expression* **;** *expression* **)** *statement*
for(*var* **in** *array* **)** *statement*
do *statement* **while(** *expression* **)**
break
continue
{ [*statement* ...] }
expression # commonly *var=expression*
print [*expression-list*] [> *expression*]
printf *format* [, *expression-list*] [> *expression*]
return [*expression*]
next # skip remaining patterns on this input line.
delete *array* [*expression*] # delete an array element.
exit [*expression*] # exit immediately; status is *expression*.

Statements are terminated by semicolons, newlines or right braces. An empty *expression-list* stands for **$0**. String constants are quoted (" "), with the usual C escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators + - * / % ^ (exponentiation), and concatenation (indicated by a blank). The operators ++ -- += -= *= /= are also available in expressions. Variables can be scalars, array elements (denoted *x*[*i*]) or fields. Variables are initialized to the null string. Array subscripts can be any string, not necessarily numeric (this allows for a form of associative memory). Multiple subscripts such

as **[***i,j,k* are permitted. The constituents are concatenated, separated by the value of **SUBSEP**.

The **print** statement prints its arguments on the standard output (or on a file if *>file* or *>>file* is present or on a pipe if *|cmd* is present), separated by the current output field separator, and terminated by the output record separator. *file* and *cmd* can be literal names or parenthesized expressions. Identical string values in different statements denote the same open file. The **printf** statement formats its expression list according to the format (see *printf*(3)).

Built-In Functions

The built-in function **close(***expr***)** closes the file or pipe *expr*.

The customary functions **exp**, **log**, **sqrt**, **sin**, **cos**, **atan2** are built in. Other built-in functions are:

length	the length of its associated argument taken as a string, or of **$0** if no argument.
rand	random number on (0,1)
srand	sets seed for **rand**
int	truncates to an integer value
substr(*s, m, n***)**	the *n*-character substring of *s* that begins at position *m* counted from 1.
index(*s, t***)**	the position in *s* where the string *t* occurs, or 0 if it does not.
match(*s, r***)**	the position in *s* where the regular expression *r* occurs, or 0 if it does not. The variables **RSTART** and **RLENGTH** are set to the position and length of the matched string.
split(*s, a, fs***)**	splits the string *s* into array elements *a***[1]**, *a***[2]**, ..., *a***[***n***]**, and returns *n*. The separation is done with the regular expression *fs*, or with the field separator **FS** if *fs* is not given.
sub(*r, t, s***)**	substitutes *t* for the first occurrence of the regular expression *r* in the string *s*. If *s* is not given, **$0** is used.
gsub	same as **sub** except that all occurrences of the regular expression are replaced; **sub** and **gsub** return the number of replacements.
sprintf(*fmt, expr, ...* **)**	the string resulting from formatting *expr* ... according to the *printf*(3) format *fmt*
system(*cmd***)**	executes *cmd* and returns its exit status

The built-in function **getline** sets **$0 to** the next input record from the current input file; **getline** *<file* sets **$0** to the next record from *file*. **getline** *x* sets variable *x* instead. Finally, *cmd* **| getline** pipes the output of *cmd* into **getline**; each call of **getline** returns the next line of output from *cmd*. In all cases, **getline** returns 1 for a successful input, 0 for end of file, and −1 for an error.

Patterns

Patterns are arbitrary Boolean combinations (with **! || &&**) of regular expressions and relational expressions. Regular expressions are as in *egrep*(1). Isolated regular expressions in a pattern apply to the entire line. Regular expressions can also occur in relational expressions, using the operators ~ and !~. */re/* is a constant regular expression; any string (constant or variable) can be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern can consist of two patterns separated by a comma; in this case, the action is per-

formed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

where a relop is any of the six relational operators in C, and a matchop is either ~ (matches) or !~ (does not match). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns **BEGIN** and **END** can be used to capture control before the first input line is read and after the last. **BEGIN** and **END** do not combine with other patterns.

Variable Names

Variable names with special meanings are:

FS	regular expression used to separate fields; also settable by option **-F***fs*.
NF	number of fields in the current record
NR	ordinal number of the current record
FNR	ordinal number of the current record in the current file
FILENAME	the name of the current input file
RS	input record separator (default newline)
OFS	output field separator (default blank)
ORS	output record separator (default newline)
OFMT	output format for numbers (default **%.6g**)
SUBSEP	separates multiple subscripts (default 034)
ARGC	argument count, assignable
ARGV	argument array, assignable; non-null members are taken as filenames

Functions can be defined (at the position of a pattern-action statement) thus:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar, and by reference if array name. Functions can be called recursively. Parameters are local to the function; all other variables are global.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average:

```
        { s += $1 }
END     { print "sum is", s, " average is", s/NR }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Simulate *echo*(1):

```
BEGIN   {                                   # Simulate echo(1)
        for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
        printf "\n"
        exit }
```

SEE ALSO

lex(1), sed(1)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, *Awk – A Pattern Scanning and*

BUGS

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string, concatenate "" to it.
Scope rules for variables in functions are not well defined.

AUTHOR

Nawk was developed by AT&T.

NAME

neqn − format mathematical text for nroff

SYNOPSIS

neqn [−**d***xy*] [−**s***n*] [−**f***n*] [−**p***n*] [*file ...*]

DESCRIPTION

Neqn is a preprocessor for *nroff*(1) for typesetting mathematical text on typewriter-like terminals. Usage is almost always:

neqn files | nroff

or equivalent.

If no files are specified (or if − is specified as the last argument), *nroff* reads from the standard input. A line beginning with **.EQ** marks the start of an equation; the end of an equation is marked by a line beginning with **.EN**. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *neqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument −**d***xy* or (more commonly) with **delim** *xy* between **.EQ** and **.EN**. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between **.EQ** and **.EN** is passed through untouched.

Tokens within *neqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus:

x sub j produces x_j

and

a sub k sup 2 produces a_k^2,

while:

$e^{x^2+y^2}$ is made by typing *e sup {x sup 2 + y sup 2}*.

Fractions are produced by using **over**:

a over b yields $\frac{a}{b}$;

sqrt produces square roots:

1 over sqrt {ax sup 2+bx+c} results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits:

$\lim_{n\to\infty} \sum_{0}^{n} x_i$ is made with *lim from {n −> inf } sum from 0 to n x sub i*.

Left and right brackets, braces, etc., of proper height are made with **left** and **right**:

left [x sup 2 + y sup 2 over alpha right] ~=~ 1 produces $\left[\frac{x^2 + y^2}{\alpha}\right] = 1.$

Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and **""** for nothing at all (useful for a right-side-only bracket). A **left** *thing* need not have a matching **right** *thing*.

Vertical piles of *things* are made with **pile**, **lpile**, **cpile**, and **rpile**:

pile {a above b above c} produces $\begin{matrix} a \\ b \\ c \end{matrix}$.

Piles may have arbitrary numbers of elements; **lpile** left justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies.

Matrices are made with **matrix**:

matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } } produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$.

In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**:

x dot = f(t) bar is $\dot{x} = \overline{f(t)}$,
y dotdot bar ~=~ n under is $\bar{\ddot{y}} = \underline{n}$, and
x vec ~=~ y dyad is $\vec{x} = \overleftrightarrow{y}$.

Point sizes and fonts can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s***n* and **−f***n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **−p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define *thing % replacement %*

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** (∑), **int** (∫), **inf** (∞), and shorthands such as >= (≥), != (≠), and -> (→) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or **GAMMA** (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Nroff*(1) four-character escapes such as **\(dd** (‡) and **\(bu** (•) may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *nroff*(1) when all else fails. Details are given in the manuals cited below.

WARNINGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.
See also WARNINGS under *nroff*(1).

SEE ALSO

Typesetting Mathematics—User's Guide, by B. W. Kernighan and L. L. Cherry.
New Graphic Symbols for EQN and NEQN, by C. Scrocca.

mm(1), nroff(1), tbl(1), mm(5).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *neqn* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

newform − change or reformat a text file

SYNOPSIS

newform [−**i**tabspec] [−**o**tabspec] [−**l**n] [−**b**n] [−**e**n] [−**c**char] [−**p**n] [−**a**n] [−**f**] [−**s**] [files]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for −**s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "−**e**15 −**l**60" will yield results different from "−**l**60 −**e**15". Options are applied to all *files* on the command line.

−**i***tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be −−, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to −**8**. A *tabspec* of −**0** expects no tabs; if any are found, they are treated as −**1**.

−**o***tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for −**i***tabspec*. If no *tabspec* is given, *tabspec* defaults to −**8**. A *tabspec* of −**0** means that no spaces will be converted to tabs on output.

−**l***n* Set the effective line length to *n* characters. If *n* is not entered, −**l** defaults to 72. The default line length without the −**l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use −**i** to expand tabs to spaces).

−**b***n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see −**l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when −**b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform −l1 −b7 file-name
```

The −**l1** must be used to set the effective line length shorter than any existing line in the file so that the −**b** option is activated.

−**e***n* Same as −**b***n* except that characters are truncated from the end of the line.

−**c***k* Change the prefix/append character to *k*. Default character for *k* is a space.

−**p***n* Prefix *n* characters (see −**c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

−**a***n* Same as −**p***n* except characters are appended to the end of a line.

−**f** Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* −**o** option. If no −**o** option is specified, the line which is printed will contain the default specification of −**8**.

−**s** Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform −s −i −l −a −e file-name

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*Newform* was called with a bad option.
not −s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 − normal execution
1 − for any error

SEE ALSO

fspec(4), csplit(1), tabs(1).

BUGS

Newform normally only keeps track of physical characters; however, for the **−i** and **−o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

Newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **−i−−** or **−o−−**).

If the **−f** option is used, and the last **−o** option specified was **−o−−**, and was preceded by either a **−o−−** or a **−i−−**, the tab specification format line will be incorrect.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [group]

DESCRIPTION

Newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. Environment variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (**PS1**) other than **$** (default) and has not exported **PS1**. After an invocation of *newgrp*, successful or not, their **PS1** will now be set to the default prompt string **$**. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

If the first argument to *newgrp* is a –, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

DIAGNOSTICS

Sorry:	You didn't qualify as a group member.
Unknown group:	The group name was not in **/etc/group**.
Permission denied:	If a password must be given, it can only come from a teletype port.

If the *stdin* is a non-tty file, this message is given:

You have no shell:	Exec of the shell failed.

BUGS

There is no convenient way to enter a password into **/etc/group**.

Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

Any shell variables that are not exported are lost.

EXTERNAL INFLUENCES

International Code Set Support

Characters from the 7-bit USASCII code set are supported in group names (see *ascii*(5)).

STANDARDS CONFORMANCE

newgrp: SVID2, XPG2, XPG3

NAME

newmail – notify users of new mail in mailboxes

SYNOPSIS

newmail [**–i** *interval*] [**–w**] [*file-spec...*]

DESCRIPTION

Newmail monitors the user's incoming or specified mailbox. Without any options, *newmail* runs in the background at a default interval of 60 seconds to monitor the user's incoming mailbox. Other options are:

–i *interval* This option changes the time interval between mailbox checks to the value specified, in seconds.

–w This option runs the program within a window where it has a more succinct output format and also runs in foreground rather than background.

The basic operation is that the program checks the incoming or specified mailbox each *interval* seconds and lists any new mail that has arrived in any of the mailboxes, indicating the sender name, and the subject of the message.

The following message is produced when the program is initially started:

Newmail started: folder *foldername*

Each entry displayed can be in a number of different formats depending on the mode of the program and the status of the message. If *newmail* is running in a window (**–w** option), then the output will resemble:

New mail from *sender name – subject of message*

or

PRIORITY mail from *sender name – subject of message*

where *sender name* is either the name of the person sending it, if available (the ARPA 'From:' line), or some other brief indication of origin. If there is no subject, the message "(No Subject Specified)" is displayed. If *newmail* is checking more than one mailbox, output lines are prefixed by the *folder-name* or prefix string specified by *file-spec*.

If the message is a "priority" message (meaning that it has a field in the header "Priority:"), the line will read **PRIORITY mail** instead of **New**mail.

When running *newmail* without the **–w** option, the output format is modified so that it is suitable for display on an already active screen. *Newmail* messages are prefixed with a pair of pointer characters as follows:

>> **New mail from** *sender name – subject of message*

or

>> **PRIORITY mail from** *sender name – subject of message*

In this case, output lines are also prefixed when monitoring multiple mailboxes.

file-spec is made up of two components: the *folder name* and the *prefix-string*, the latter of which can always be omitted. The format is *foldername=prefix-string*. Metacharacters such as +, =, and % which indicate the folder directory in the *elm* mailer (see *elm*(1)) is available to specify the folder name (see EXAMPLES).

Newmail runs until the user logs out or explicitly kills it, and can internally reset itself if the mailbox shrinks in size and then grows again.

EXAMPLES

Here are some example invocations:

Check incoming mailbox every 60 seconds:

newmail

Check incoming mailbox every 15 seconds for new messages from "joe" or "root".

newmail −i 15 joe root

Monitor the incoming mailbox for new messages from user "mary" and the folder in your *maildir* called "postmaster". Prefix all messages from "mary" with the string "Mary", and messages from "postmaster" (see *elm*(1) with "POBOX". Also, monitor folder **/tmp/mbox**:

newmail mary=Mary +postmaster=POBOX /tmp/mbox

WARNING

If the interval is set to less than 10 seconds, *newmail* issues a warning message that such short intervals are not recommended.

Interval must be less than 2^{32}. seconds because *newmail* uses *sleep*(3C).

AUTHOR

newmail was developed by Hewlett-Packard Company.

NAME

news − print news items

SYNOPSIS

news [**−a**] [**−n**] [**−s**] [items]

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *News* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

Options

−a Causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

−n Causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

−s Causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If an interrupt is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/usr/news/*
$HOME/.news_time
/etc/profile

SEE ALSO

mail(1), profile(4), environ(5).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

news: SVID2, XPG2

NAME

nice – run a command at low priority

SYNOPSIS

nice [–increment] command [arguments]

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **––10**.

An *increment* larger than 19 is equivalent to 19.

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

Nice returns the exit status of the subject command.

NOTES

Nice is built into *csh*(1) with a slightly different syntax than described here. The form "nice +10" nices to positive nice, and "nice -10" can be used by the super-user to give a process more of the processor.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

nice: SVID2

NAME

nl – line numbering filter

SYNOPSIS

nl [–**b***type*] [–**h***type*] [–**f***type*] [–**p**] [–**v***start#*] [–**i***incr*] [–**s***sep*] [–**w***width*] [–**n***format*] [–**l***num*] [–**d***delim*] [file]

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless told otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

–**b***type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meanings are:

a	number all lines;
t	number lines with printable text only;
n	no line numbering;
p*string*	number only lines that contain the regular expression specified in *string*. Basic Regular Expression syntax is supported (see *regexp*(5)).

The default *type* for logical page body is **t** (text lines numbered).

–**h***type* Same as –**b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

–**f***type* Same as –**b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

–**p** Do not restart numbering at logical page delimiters.

–**v***start#* *Start#* is the initial value used to number logical page lines. Default is **1**.

–**i***incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

–**s***sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

–**w***width* *Width* is the number of characters to be used for the line number. Default *width* is **6**.

–**n***format* *Format* is the line numbering format. Recognized values are:

ln left justified, leading zeroes suppressed;
rn right justified, leading zeroes suppressed;
rz right justified, leading zeroes kept.

Default *format* is **rn** (right justified).

−l*num* *Num* is the number of blank lines to be considered as one. For example, **−l2** results in only the second adjacent blank being numbered (if the appropriate **−ha**, **−ba**, and/or **−fa** option is set). Default is **1**.

−d*xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **−d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl −v10 −i10 −d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are ! and +.

SEE ALSO

pr(1), environ(5), lang(5), regexp(5).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the characters matched by character class expressions in regular expressions.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *nl* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

nl: SVID2, XPG2, XPG3

NAME

nlio – control Native Language I/O

SYNOPSIS

nlio [**–c** [*flag*]] [**–n**] [**–s**] [**–t** [*tb*]]

DESCRIPTION

Nlio controls Native Language I/O for the current standard input device. Without arguments, *nlio* reports the settings. If the standard input is not a terminal or is inaccessible, *nlio* tries the current standard output. If both fail, it terminates with an error message. *Nlio* writes normal messages on the standard output and error messages on the standard error output.

Options

Nlio does all of the specified options one by one from left to right, even if some fail:

–c[*flag*] Specify the scope of 16-bit data handling (parsing data and conversion):

- **i** for input,
- **o** for output,
- **io** or **b** for both input and output,
- **n** for neither input nor output.

If *flag* is omitted, *nlio* prints the current status of **i**, **o**, **b**, or **n**.

–n Check whether Native Language I/O is available for the terminal. If it is available, *nlio* returns exit code 0; if not, 1. No messages are displayed.

–s Print the name of the Native Language I/O server running on the terminal.

–t[*tb*] Specify terminal behavior. If it is omitted, *nlio* prints the name of the current terminal behavior (see *nlioinit*(1)).

When the login shell is terminated, the settings are reset to **–cb -t**<initial tb>.

RETURN VALUE

Nlio returns exit code **0** in success and **1** in failure.

AUTHOR

Nlio was developed by HP.

SEE ALSO

nlioenv(1), nliostart(1), nlioinit(1M).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

nlioenv – set Native Language I/O environment

SYNOPSIS

nlioenv [−**p** [*files*]] [−**w** [*files*]] [−**l** *flag*]

DESCRIPTION

Nlioenv sets the Native Language I/O environment for the terminal device that is the current standard input. Without arguments, it reports the settings. If the standard input is not a terminal or is inaccessible, *nlioenv* tries the current standard output. If both fail, *nlioenv* terminates with an error message. It writes normal messages on the standard output and error messages on the standard error output. It tries to do all of the specified options one by one from left to right, even if some fail.

Options

−**p** [*files*] The *files* are absolute or relative path names of word dictionaries used for phrase conversion in addition to the phrase dictionary. Up to three dictionaries can be specified to be opened. The dictionaries previously in use are closed and replaced with the dictionaries specified by this option. The opened dictionaries will be accessed in the order they are specified, and then the phrase dictionary will be accessed. If *files* are omitted, *nlioenv* closes the currently opened dictionaries for phrase conversion and stops using them.

−**w** [*files*] The *files* are absolute or relative path names of word dictionaries to be used for word conversion. Up to three dictionaries can be specified to be opened. The dictionaries previously in use are closed and replaced with the dictionaries specified by this option. The opened dictionaries will be accessed in the order they are specified. If *files* are omitted, *nlioenv* closes the currently opened dictionaries for word conversion and stops using them.

−**l***flag* The *flag* specifies the setting to be listed. If it is **p**, *nlioenv* prints the absolute path names of the dictionaries currently in use for phrase conversion in the order they would be accessed. If *flag* is **w**, *nlioenv* prints the names of dictionaries currently in use for word conversion.

When the login shell is terminated, all of the opened dictionaries for the terminal device are closed.

RETURN VALUE

Nlioenv returns exit code **0** in success and **1** in failure.

WARNINGS

The length of the absolute path name of word dictionaries is limited to 119 bytes.

AUTHOR

Nlioenv was developed by HP.

SEE ALSO

nlio(1), nliostart(1), nlioinit(1M).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

nliostart – start Native Language I/O

SYNOPSIS

nliostart *tb* [*server*]

DESCRIPTION

Nliostart enables Native Language I/O by executing a NLIO server and a new shell. *Nliostart* finds an available pseudo terminal (see *pty*(7)) and executes an NLIO server between the master side of the pty and the user's terminal. It also executes a new shell on the slave side of the pty. The shell is determined by the **SHELL** environment variable. If it is not defined, **/bin/sh** is used as a default shell. Exiting the new shell terminates *nliostart*. If an NLIO server is already running for the current tty and either input or output conversion are enabled for the tty, *nliostart* does not execute a new NLIO server.

The argument *tb* specifies the terminal behavior for the I/O. The following choices are supported:

bj	Japanese
bk	Korean
bc	Simplified Chinese
bt	Traditional Chinese
hp35714a	Japanese, with 25 screen lines
atbj	Japanese, with 24 lines and a line used for host input conversion

The arguments **hp35714a** and **atbj** are for HP-16 terminals that require host input conversion.

The optional argument *server* identifies the NLIO server that *nliostart* executes to perform Native Language I/O. If the *server* is omitted, the default server **bserver** is executed.

Each server is capable of conversion between internal code (HP-15) and external code, although some features are not available on all servers. When the *stty*(1) command option **icanon** is set, entering a single erase character erases the preceding 16-bit or 8-bit input character.

bserver	Convert code between internal code (HP-15) and external code for the languages of Japanese, Korean, Simplified Chinese, and Traditional Chinese. **Bserver** is not capable of host input conversion.
j0server	Convert code for the Japanese language only. **J0server** is capable of host input conversion using phrase and word dictionaries. It uses the input window for 16-bit character input.

AUTHOR

Nliostart was developed by HP.

SEE ALSO

nlio(1), nlioenv(1), nlioinit(1M), pty(7), stty(7).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

nljust – justify lines, left or right, for printing

SYNOPSIS

nljust [**–aclnt**] [**–e** *seq*] [**–j** *just*] [**–m** *mode*] [**–o** *order*] [**–r** *margin*] [**–w** *width*] [**–x** *ck*] [*file ...*]

DESCRIPTION

Nljust formats for printing data written in languages with a right-to-left orientation. It is designed to be used with the *pr*(1) and the *lp*(1) commands.

Nljust reads the concatenation of input files (or standard input if none are given) and produces on standard output a right-to-left formatted version of its input. If "-" appears as an input file name, *nljust* reads standard input at that point. You can use "--" to delimit the end of options.

Nljust formats input files for all languages that are read from right to left. For languages that have a left-to-right orientation, the command merely copies input files to standard output.

Options

–a Justify data for all languages, including those having a left-to-right text orientation. By default only right-to-left language data is justified. For all other languages, input files are directly copied to standard output.

–c Select enhanced printer shapes for some Arabic characters. With this option, two character combinations of laam and alif are replaced by a single character.

–e *seq* Use *seq* as the escape sequence to select the primary character set. This escape sequence is used by languages that have too many characters to be accommodated by ASCII in a single 256-character set. In these cases, the *seq* escape sequence can be used to select the non-ASCII character set. The *escape* character itself (0x1b) is not given on the command line. Hewlett-Packard escape sequences are used by default.

–j *just* If *just* is **l**, left justify print lines. If *just* is **r**, right justify print lines starting from the (designated or default) print width column. The default is right justification.

–l Replace leading spaces with alternative spaces. Some right-to-left character sets have a non-ASCII or alternative space. This option can be useful when filtering **pr** **–n** output (see *pr*(1)). With right justification, the –l option causes line numbers to be placed immediately to the right of the tab character. Without the **–l** option, right justification causes line numbers to be placed at the print-width column. By default, leading spaces are not replaced by alternative spaces.

–m *mode* Indicate *mode* of any file to be formatted. Mode refers to the text orientation of the file when it was created (see *hpnls*(5) for more details). If *mode* is **l**, assume Latin mode. If *mode* is **n**, assume non-Latin mode. By default, mode information is obtained from the **LANGOPTS** environment variable.

–n Do not terminate lines containing printable characters with a new-line. By default, print lines are terminated by new-lines.

–o *order* Indicate data *order* of any file to be formatted. The text orientation of a file can affect the way its data is arranged (see *hpnls*(5) for more details). If *order* is **k**, assume keyboard order. If *order* is **s**, assume screen order. By default, order information is obtained from the **LANGOPTS** environment variable.

–t Truncate print lines that do not fit the designated or default line length. Print lines are folded (that is, wrapped to next line) by default.

−x *ck* Expand input tabs to column positions *k+1*, *2*k+1*, *3*k+1*, etc. Tab characters in the input are expanded to the appropriate number of spaces. If *k* is 0 or is omitted, default tab settings at every eighth position is assumed. If *c* (any non-digit character) is given, it is treated as the input tab character. The default for *c* is the tab character. *Nljust* always expands input tabs. This option provides a way to change the tab character and setting. If this option is specified, at least one of the parameters *c* or *k* must be given.

−r *margin* Designate a number as the print *margin*. The print margin is the column where truncation or folding takes place. The print margin determines how many characters appear on a single line and can never exceed the print width. The print margin is relative to the justification. If the print margin is 80, folding or truncation occurs at column 80 starting from the right during a right justification. Similarly, folding or truncation occurs at column 80 starting from the left during a left justification. By default, the print margin is set to column 80.

−w *width* Designates a number as the print *width*. The print width is the maximum number of columns in the print line. Print width determines the start of text during a right justification. The larger the print width, the further to the right the text will start. By default, an 80-column print width is used.

EXAMPLES

Right justify *file1* on a 132-column printer with a print margin at column 80 (the default).

nljust −w 132 file1 | lp

Right justify *pr* output of *file2* with line numbers on a 132-column printer with a print margin at column 132.

pr −n file2 | nljust −w

WARNINGS

If *pr*(1) with line numbers (**−n** option) is piped to *nljust*, the separator character must be a tab (0x09).

It is the user's responsibility to ensure that the **LANGOPTS** environment variable accurately reflects the status of the file.

Mode and justification must be consistent. Only non-Latin mode files can be right justified in a meaningful way. Similarly, only Latin mode files can be safely left justified. If mode and justification do not match, the results are undefined.

If present, alternative numbers always have a left-to-right orientation.

AUTHOR

Nljust was developed by HP.

SEE ALSO

forder(1), lp(1), pr(1), strord(3C), hpnls(5).

EXTERNAL INFLUENCES

Environment Variables

The LANGOPTS environment variable determines the mode and order of the file. The syntax of LANGOPTS is [*mode*][*_order*]. The *mode* describes the mode of a file where **l** represents Latin mode and **n** represents non-Latin mode. Non-Latin mode is assumed for values other than **l** and **n**. The *order* describes the data order of a file where **k** is keyboard and **s** is screen. Keyboard order is assumed for values other than **k** and **s**. Mode and order information in LANGOPTS can be overridden from the command line.

The LC_ALL environment variable determines the direction of a language (left-to-right or right-to-left) and whether context analysis of characters is necessary.

The LC_NUMERIC environment variable determines if a language has alternative numbers.

The LANG environment variable determines the language in which messages are displayed.

International Code Set Support

Single-byte character code sets are supported.

NAME

nlsinfo − display native language support information

SYNOPSIS

nlsinfo [**−acfhilmnstC**] [**−d** *n*] [**−e** *n*] [**−o** *n*] [**−r** *n1 n2*] [**−L** *language*]

DESCRIPTION

Nlsinfo displays a list of installed Native Language Support (NLS) international environment languages and displays information contained within the runtime locale associated with these languages (see *setlocale*(3C)). Commands and routines modified for NLS operation (see the EXTERNAL INFLUENCES section on applicable manual pages) use the information contained within locales to process and display data according to the local language and customs of a user (see *hpnls*(5)). Unless overridden by the **−L** option, the runtime locale is determined by the locale category environment variables (those that begin with **LC_**) and the LANG environment variable.

By default, non-printable characters are displayed as 2-digit hexadecimal numbers unless they are included within strings, in which case the hexadecimal numbers are preceded by \x.

Options

The following options are recognized:

−l Display the list of installed international environment languages.

−a Display the LC_ALL locale category which contains all parts of the selected locale. Equivalent to specifying options **−csntmh**.

−c Display the LC_CTYPE locale category that contains character classification and conversion information (see *ctype*(3C), *conv*(3C), and *nl_tools_16*(3C)).

−s Display the LC_COLLATE locale category that contains collating (sorting) information (see *strcoll*(3C) and *strxfrm*(3C)).

−n Display the LC_NUMERIC locale category which contains information regarding the radix character and thousands separator.

−t Display the LC_TIME locale category which contains such information as the format of the date and time and the names of the days of the week (see *strftime*(3C)).

−m Display the LC_MONETARY locale category which contains information for the formatting of monetary quantities.

−h Display all other local custom information not included in the LC_NUMERIC, LC_TIME and LC_MONETARY locale categories.

−i Display a list of abbreviated "from" and "to" code set names used by *iconv*(1) the code set conversion utility. Each "from" and "to" code set name pair represents a conversion supported by *iconv*(1).

−C Display all the characters defined in the selected NLS environment's code set. The list of characters may be quite long for some Asian NLS environments.

If none of the options **l**, **a**, **c**, **s**, **n**, **t**, **m**, **h**, **i**, or **C** is specified, **−l** is used as a default.

−f Place a form feed character before all category display headings.

−o *n* Display non-printable characters as numbers using the base specified by *n*. If *n* is *o* display non-printable characters as 3-digit octal numbers. If *n* is *x* display non-printable characters as 2-digit hex numbers. If non-printable characters are included within strings, the

octal or hex numbers are preceded by a backslash (\).

−e *n* Display all characters (including printable characters) as hexadecimal or octal numbers. If *n* is *o* display characters as 3-digit octal numbers. If *n* is *x* display characters as 2-digit hex numbers. If characters are included within strings, the octal or hex numbers are preceded by a backslash (\).

−d *n* Display *n* lines between headings. By default 16 lines are displayed between headings.

−r *n1 n2* Display information only about element number *n1* through element number *n2* of the requested table(s), rather than all elements of each table. The parameters *n1* and *n2* can be hexadecimal (preceded by "0x"), octal (preceded by "0") or decimal numbers. These parameters can also be characters in which case each character is mapped to its value in the code set of the user's current locale (not the locale, if any, specified via the **−L** option). Characters which are digits should be quoted with the backslash character (\) to prevent them from being interpreted as numbers.

−L *language* Display information about the locale associated with *language* instead of the locale specified by the setlocale category and LANG environment variables. The parameter *language* should be one of the language names from the output of "nlsinfo -l" (the valid language names are also given in *lang*(5)).

WARNINGS

The format and content of the display of a particular table may change if the underlying NLS implementation is changed to support new features. Applications should not use the information displayed by *nlsinfo* as the basis for their operation, but rather should use the interface routines provided in LIBC (see *conv*(3C), *ctype*(3C), *langinfo*(3C), *nl_tools_16*(3C), *setlocale*(3C), *string*(3C) and *buildlang*(1M)).

AUTHOR

Nlsinfo was developed by HP.

FILES

/usr/lib/nls/config
/usr/lib/nls/$LANG/locale.def

SEE ALSO

buildlang(1M), iconv(1),
conv(3C), ctype(3C), langinfo(3C), nl_tools_16(3C), setlocale(3C), string(3C),
environ(5), hpnls(5), langid(5).

EXTERNAL INFLUENCES

Environment Variables

The LC_CTYPE environment variable determines how option argument characters are mapped to a value in the code set.

The LANG environment variable determines the language in which messages and headings are displayed.

International Code Set Support

Single- and multi-byte-character code sets are supported with the exception of multi-byte-character file names.

NAME

nm – print name list of common object file.

SYNOPSIS

nm [*options*] [*file*]

REMARKS

This is a generic entry for a machine-dependent program. A specific entry is provided for each version. Refer to manual entry *nm_300*(1) for information about *nm* on Series 300 systems or *nm_800*(1) for information about *nm* on Series 800 systems.

SEE ALSO

nm_300(1), nm_800(1).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

nm: SVID2, XPG2, XPG3

NAME

nm – print name list of common object file

SYNOPSIS

nm [–**gnoprsu**] [file ...]

DESCRIPTION

The *nm* command prints the name list (symbol table) of each common object *file* in the argument list. If an argument is an archive, a listing for each common object file in the archive will be produced. If no *file* is given, the symbols in **a.out** are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. If the symbol is a secondary definition, the type letter is followed by the letter **S**. The output is sorted alphabetically.

The options are:

–**g**	Print only global (external) symbols.
–**n**	Sort numerically rather than alphabetically.
–**o**	Precede each output line with the file or archive element name, rather than printing the file or archive element name only once. This option can be used to make piping to *grep*(1) more meaningful.
–**p**	Do not sort; print in symbol-table order.
–**r**	Sort in reverse order.
–**s**	Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next highest value). This difference is the value printed. This flag turns on –**g** and –**n** and turns off –**u** and –**p**.
–**u**	Print only undefined symbols.

SEE ALSO

ar(1), a.out_300(4), a.out_800(4), ar(4).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

nm – print name list of object file

SYNOPSIS

nm [**–hnoprxuvTV**] file...

DESCRIPTION

The *nm* command displays the symbol table of each object file *file*. *File* may be relocatable or absolute common object file, or it may be an archive of relocatable or absolute common object files. For each symbol, at least the following information will be printed:

Name The name of the symbol.

Value Its value expressed as an offset or an address depending on its storage class.

Scope The scope of the symbol (undefined, static, sdef, or external). The "sdef" scope indicates an external symbol that is flagged as a secondary definition.

Type The type of the symbol (code, data, common, absolute, etc.).

Subspace The subspace to which the symbol belongs.

The output of *nm* may be controlled using the following options:

–e Print only external and static symbols. This is the normal behavior, so this option is ignored.

–f Produce full output. This is the normal behavior, so this option is ignored.

–h Do not display the output header data.

–n Sort symbols by *name*, in ascending collation order, before they are printed (see Environment Variables below).

–o Print the *value* and *size* of a symbol in octal instead of decimal.

–p Produce easily parsable, terse output. Each symbol *name* is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text symbol), **D** (data symbol), **B** (bss symbol), or **C** (common symbol). If the symbol is local (non-external), the type letter is in lowercase. If the symbol is a secondary definition, the type letter is followed by the letter **S**.

–r Prefix each output line with the name of the object file or archive.

–u Print undefined symbols only.

–v Sort symbols by *value* before they are printed.

–x Print the *value* and *size* of a symbol in hexadecimal instead of decimal.

–T By default, *nm* prints the entire name of the symbols listed. Since object files can have symbol names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **–T** option causes *nm* to truncate every name that would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

–V Print the version of the *nm* command executing on the standard error output.

SEE ALSO

cc(1), ld(1).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating order output by the *–n* option.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *nm* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single-byte character code sets are supported.

NAME

nohup – run a command immune to hangups, logouts, and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

Nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**; otherwise, *nohup* will fail.

If output from *nohup* is redirected to a terminal, or is not redirected at all, the output is sent to **nohup.out**.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

nohup sh file

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

nohup file &

An example of what the contents of *file* could be is:

tbl ofile | eqn | nroff > nfile

SEE ALSO

chmod(1), nice(1), sh(1), signal(5).

WARNINGS

Be careful to place punctuation properly, for example in the following command:

nohup command1; command2

nohup applies only to *command1*, and the following command is syntactically incorrect:

nohup (command1; command2)

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

nohup cpio –o <list >/dev/rmt/1m&

while

nohup cpio –o <list >/dev/rmt/1m 2>errors&

puts the error messages into file *errors*.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

nohup: SVID2, XPG2, XPG3

NAME

nroff − format text

SYNOPSIS

nroff [*options*] *file* ...

DESCRIPTION

Nroff formats text contained in *file* (standard input by default) for printing on typewriter-like devices and line printers. Its capabilities are described in the tutorial cited below.

Nroff is best not used directly, but rather via macro packages such as *mm*, which provides a high-level interface to document processing, as opposed to the very low level one provided directly in *nroff*.

An argument consisting of a minus (−) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order but must appear before the *file*, are:

−o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N*−*M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end. (See **WARNINGS** below.)

−n*N* Number first generated page *N*.

−s*N* Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, for example, with *mm*(1)). When *nroff* halts between pages, an ASCII **BEL** is sent to the terminal.

−r*aN* Set register *a* (which must have a one-character name) to *N*.

−i Read standard input after *files* are exhausted.

−q Invoke the simultaneous input-output mode of the **.rd** request.

−z Print only messages generated by **.tm** (terminal message) requests.

−m*name* Precede the input *files* with the non-compacted (ASCII text) macro file **/usr/lib/nls/**LANG**/tmac/tmac.***name*, where LANG is the value of the LANG environment variable. If LANG is not set or **/usr/lib/nls/**LANG**/tmac/tmac.***name* does not exist **/usr/lib/tmac/tmac.***name* is used instead.

−c*name* Precede the input *files* with the compacted macro files **/usr/lib/macros/cmp.[nt].[dt].***name* and **/usr/lib/macros/ucmp.[nt].***name*.

−k*name* Compact the macros used in this invocation of *nroff*, placing the output in files **[dt].***name* in the current directory.

−T*name* Prepare output for specified terminal. Known *name*s are **37** for the (default) TELETYPE Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, **2631** for the Hewlett Packard 2631 line printer, and **klp** for a (generic) 16-bit character printer having ratio of 2 to 3 in 8-bit and 16-bit character width.

−e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.

−h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character

widths.

−u*n* Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

EXAMPLES

The following command prints the first five pages of the document whose *nroff* source file is *filename*:

nroff -o-5 *filename*

Note that there should not be a space between the **o** and the − or the − and the **5**.

To print only pages 1, 3, and 4 type:

nroff -o1,3,4 *filename*

WARNINGS

When *nroff* is used with the **−o***list* option inside a pipeline, it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

FILES

/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers

SEE ALSO

mm(1), soelim(1).
nroff/troff tutorial in the *Text Formatters* volume of *HP-UX Concepts and Tutorials*.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LANG is used to determine the search path for the **−m** option. LANG also determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *nroff* behaves as if all internationalization variables are set to "C". See *environ*(5).

NAME

od, xd − octal and hexadecimal dump

SYNOPSIS

od [**−bcdosx**] [file] [[+][**0x**]offset[.][**b**]]
xd [**−bcdosx**] [file] [[+][**0x**]offset[.][**b**]]

DESCRIPTION

Od (*xd*) dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **−o** (**−x**) is the default. An offset field is inserted at the beginning of each line. For *od*, the offset is in octal, for *xd* the offset is in hexadecimal.

Options

The meanings of the format options are:

−b Interpret bytes in octal (hexadecimal).

−c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**0**, backspace=**b**, form-feed=**f**, new-line=**n**, return=**r**, tab=**t**; others appear as 3-digit octal numbers.

−d Interpret 16-bit words in decimal.

−o Interpret 16-bit words in octal.

−s Interpret 16-bit words in signed decimal.

−x Interpret 16-bit words in hexadecimal.

The *file* argument specifies which file is to be dumped. If no *file* argument is specified, the standard input is used.

The *offset* argument specifies the offset in the file where dumping is to commence, and is normally interpreted as octal bytes. Interpretation can be altered as follows:

offset must be preceded by + if the file argument is omitted.

offset preceded by **0x** is interpreted in hexadecimal.

offset followed by **.** is interpreted in decimal.

offset followed by **b** is interpreted in blocks of 512 bytes.

Dumping continues until end-of-file.

SEE ALSO

adb(1).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the range of printable characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used *instead* of LANG. If any internationalization variable contains an invalid setting, *od* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported. Multi-byte data is displayed as multi-byte values.

STANDARDS CONFORMANCE

od: SVID2, XPG2, XPG3

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [**–f**] *name* ...

pcat *name* ...

unpack *name* ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible, each input file *name* is replaced by a packed file *name*.**z** with the same ownership, modes, and access and modification times. The **-f** option forces packing of *name*. This is useful for causing an entire directory to be packed even if some of the files do not benefit. If *pack* is successful, *name* is removed. Packed files can be restored to their original form using *unpack* or *pcat*.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
the file is empty;
no disk storage blocks will be saved by packing;
a file called *name*.**z** already exists;
the **.z** file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.**z** use:

pcat name.z

or just:

pcat name

To make an unpacked copy, say *nnn*, of a packed file named *name*.**z** (without destroying *name*.**z**) use the command:

pcat name >nnn

Pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the **.z**) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

Unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.**z** (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the "unpacked" name already exists;
if the unpacked file cannot be created.

Access Control Lists (ACLs)

Pack retains all entries in a file's access control list when compressing and expanding it (see *acl*(5)).

DEPENDENCIES

Series 800

Access control lists are not implemented.

RFA and NFS

Optional access control list entries of networked files are summarized (as returned in st_mode by *stat*(2)), but not copied to the new file.

SEE ALSO

cat(1), compact(1), compress(1), acl(5).

STANDARDS CONFORMANCE

pack: SVID2, XPG2, XPG3

pcat: SVID2, XPG2, XPG3

unpack: SVID2, XPG2, XPG3

NAME

pam – Personal Applications Manager, a visual shell

SYNOPSIS

pam [**–c** *args* ...]

DESCRIPTION

Pam is a program that provides a friendlier, less intimidating means of communication between HP-UX and system users. It provides many of the traditional capabilities supported by other shell programs, such as executing commands as foreground processes (that is, where you must wait until one command has been completed before the system accepts the next command) or background processes (where the command runs in the background while you perform other tasks in the foreground). *Pam* also supports other useful capabilities such as using substituted files instead of standard input and standard output, pipelining several processes into a single command, and handling shell scripts and programs. *Pam* maintains a continuous display of the open folder (current directory), and makes use of windowing and mouse I/O facilities when they are available on the system.

Display

The *pam* display has two parts. The top two lines are called the command area, while the remainder of the display is the folder (directory) area. The first line in the command area displays messages (such as prompts and errors) from the system to the user, while the second line displays input commands and text from the user to the system. *Pam* maintains a buffer of 20 command lines. You can use shifted arrow keys, BACK SPACE, INSERT CHARACTER, and DELETE CHARACTER to access and edit any existing current or previous command line in the 20-line command buffer. For example, each time you press SHIFT–UP ARROW, the next previous command line in the buffer is displayed.

The folder (lower) area is used by *pam* to display files that reside in the currently open folder (that is, the current directory). One of the file names displayed in the folder area is highlighted. This highlighted file name identifies which file in the folder should be used as a file name parameter for *pam* commands that are invoked using the *pam* menu. The highlighted file name can be changed by using TAB, SHIFT–TAB and arrow keys.

Commands

A command is a sequence of words separated by blanks. In general, the first word is the name of the command, and the words that follow are passed as arguments to the invoked command. Two or more commands (together with their associated arguments, if any) separated by a vertical bar (|) form a pipeline. To pass data between commands in a pipeline, the standard output from one command is connected to the standard input of the next command in the pipeline.

To force *pam* to complete execution of the current command or pipeline before running another command, place a semicolon (;) at the end of the line. If you prefer to perform other tasks while the command or pipeline is being executed, run the first command as a "background" process by adding an ampersand (&) at the end of the command line. *Pam* then starts the command, and without waiting for completion, returns for your next instruction.

In a windowed system, interactive commands are treated as background processes (&) unless a semicolon is present at the end of the line. In non-windowed systems, commands taken from a script or from interactive commands are run to completion before the next command is accepted for execution (;) unless an ampersand is present at the end of the command line.

Sequences of more than one command or pipeline can be joined on a single command line by placing a semicolon or ampersand (but not both) between each adjacent pair of

commands or pipelines in the line. In such constructs, commands separated by semicolon (;) are executed in sequence (that is, the first command is run to completion before the next is begun). Commands separated by ampersand (&) are executed simultaneously on a timesharing basis. (However, this does not necessarily result in the most efficient use of computer resources due to timesharing overhead as sharing processes compete for processor time.) Note that when semicolon or ampersand is used to separate commands, standard output from one command is not automatically connected to standard input for the next. When data must be passed between successive commands or programs, use the pipeline connector (|) or redirect standard output and input to and from a specified file.

Pam runs commands based on the file type of the command name:

program (executable) The command name is run (exec'ed) or, if it is a shell script, the commands in the script are run.

folder (directory) The command name (folder) becomes the new open folder. This is equivalent to *cd* folder.

data (non-executable) The command name (data file) is displayed one page at a time.

Standard Input, Output, and Error Files

The standard input, output, and error of a command can be redirected using the following syntax:

< *name*	Use the file *name* as standard input for the command.
> *name*	Use the file *name* as standard output for the command.
>> *name*	Use the file *name* as standard output for the command, but concatenate the output to the end of the file.
^ *name*	Use the file *name* as standard error for the command.
^^ *name*	Use the file *name* as standard error for the command, but concatenate the output to the current end of the file, if it exists.
# *name*	[For windowed systems only] Use the named window as standard input, output and error for the command. If the window doesn't exist, a window is created. Specific redirection of I/O with >, >>, <, |, ^, or ^^ overrides any redirection specified with "#".

I/O redirection is possible only with an associated command. Multiple redirections of standard input, output, and error associated with a command are not allowed. The I/O redirection can be placed anywhere in the command.

If a command is followed by "&" (background process), the default standard input for the command is the empty file **/dev/null**.

Using Patterns to Represent Filenames

Each word in the command line (command name, parameter, redirection file name, window name) is scanned for the characters *, ?, and [. If one of these characters appears, the word is treated as a pattern that represents more than one filename. *Pam* replaces the pattern word with alphabetically sorted filenames corresponding to the pattern. If no file name is found that matches the pattern, the word is left unchanged. A period character (.) at the start of a filename or immediately following a /, as well as the character / itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by a hyphen (-) matches any character lexically between the pair, inclusive. A NOT operator (!) can be specified immediately following the left bracket to match any single character not enclosed in the brackets.

Quoting

Characters can be quoted on the *pam* command line to prevent *pam* from performing special processing on the characters (such as <, >, #, space, |, ;, &, *, ?, [). A pair of double (") or single (') quote characters can be used to enclose the character string being quoted. The \ character quotes only the single following character.

Scripts

A script is an executable file containing command lines and comments. A comment is a line that begins with "!" or "#"; comments are ignored by *pam*. The command lines in the script file are executed in sequence (unless non-sequential execution is explicitely specified using the "&" character).

Script arguments that are specified when a script is run can be accessed by script commands using the notation "$1" for the first argument, "$2" for the second argument, etc. All arguments can be accessed at once using "$*". The name of the script can be accessed using "$0".

Autost

If a file named **Autost** exists in the open folder when *pam* is started, it is automatically processed as a command. If **Autost** is a script file, it is run as **source Autost** (see Built-In Commands described later). Otherwise, it is processed as if it were entered as a command (for example, if **Autost** is a data file, it is viewed). *Pam* does not process any command input until processing of the **Autost** file is complete.

Environment

The *pam* environment is set up when *pam* is run, and can be reset at any time by using the command *getenv*. The environment variables are read from a file and are not sorted or checked for syntax by *pam*. *Pam* passes the current environment to commands that it starts and uses the following environment variables in running commands:

ACTION The ACTION variable specifies a command name (corresponding to an executable file), and is used whenever a data file is specified as a command. The ACTION command is run in this case and the data file is passed as the first argument. The default value for ACTION is "view".

HOME The HOME variable specifies a folder and is used whenever the "cd" command is run without an argument. The HOME folder specifies the directory to change to in this case. The default value for HOME is "/".

LANG The LANG variable specifies the language for which the system is localized. The default value for LANG is **n-computer**.

PATH The PATH variable specifies a list of folders. When *pam* runs a command it looks for it in the folders in the PATH list.

SCRSHELL The SCRSHELL variable specifies the shell to be used by *pam* in running scripts. If the specified name does not contain a "/" then *pam* searches for the shell using the PATH environment variable. If the SCRSHELL is undefined or the specified shell does not exist, the script is processed by *pam*.

Menu

The *pam* menu displays the following softkey menu labels corresponding to the indicated function keys:

[function key 1]	**open**, **view**, or **start** (a program), or **reread**
[f2]	**echo**
[f3]	**arrow** (toggle key)
[f4]	**move**
[f5]	**copy**
[f6]	**rename**
[f7]	**delete**
[f8]	**close**

The command associated with a menu item is run whenever the item is selected by pressing the corresponding function key. The highlighted file name in the folder area of the display is used as a parameter for the command.

The menu item associated with **f3** is used to toggle the semantics of the arrow keys and is available only in non-windowed systems. **f3** is initially set to use the arrow keys for manipulating the position of the file highlight in the folder area. **f3** alternately controls movement of the cursor on the command line.

Built-In Commands

Several commands are executed directly by *pam*:

cd [name]
: Make the named folder (directory) the open folder (current directory). If no folder is specified, the HOME environment variable is used to determine which folder to open.

close
: Closes the open folder and displays the parent folder.

copy *name1* [*name2*]
copy *name1* [*name2* ...] *folder_name*
: Copy *name1* to *name2*; if *name2* exists and is not a folder, it is overwritten. If only *name1* is specified, the copy is completed with the command *to*name2. If the last parameter specified is a folder, all the specified files are copied into that folder.

delete *name1* [*name2* ...]
: The named files and folders (if empty) are deleted.

echo [arg ...]
: Arguments are written to standard output. The echo menu item (menu item 2 and/or function key 2) writes the full pathname of the highlighted file in the folder area of the display to the command line.

getenv *name*
: The named file is read in and used as the active environment.

makefolder *name1* [*name2* ...]
: Folders are created and given the specified name(s).

move *name1* [*name2*]
move *name1* [*name2*] *folder_name*
: Rename file or folder *name1* to *name2*. If *name2* exists and is a file, it is overwritten. If only *name1* is specified, use the command *to name2* to complete the move. If the last parameter is a folder, all the specified files are moved into that folder.

rename *name1* [*name2*]
rename *file_name1* [*file_name2* ...] *folder_name*
: Same as **move.**

reread Reread the open folder and update the display. The keystroke CONTROL–L also does a reread.

source *name* [*arg* ...] Read command lines from the named script file and execute them. A shell is NOT forked to execute the commands. Parameter substitution for the arguments (*arg* ...) is handled the same way as during regular script execution.

to [*name*] Complete a pending copy, move or rename. The file or folder *name* identifies the destination for a preceding **copy**, **move**, or **rename** command that had no destination specified. If *name* is omitted, the destination defaults to the current open folder.

view *name1* [*name2* ...]
Copy the specified file(s) to standard output. If standard output is the screen (default), the file is displayed one page at a time.

Signals

Pam ignores INTERRUPT and QUIT signals if the command is followed by an "&"; otherwise *pam* uses default signal handling when running commands.

Invoking *Pam*

Pam can be invoked as a keyboard command or from a program. When *pam* is invoked without **-c** as the first argument, *pam* acts as an interactive, display-oriented command interpreter.

When *pam* is invoked with **-c** as the first argument (either from the keyboard or from a running program), the remaining arguments in the command are interpreted as command inputs intended for processing by *pam*. The list of arguments intended as commands for *pam* must not exceed a total of 160 characters. When the **-c** option is used, *pam* executes the list of command arguments (built-in commands, redirection, pipes, and most other *pam* facilities can be used), then exits.

Exiting from *Pam*

To terminate *pam* and return to normal HP-UX operation, press CONTROL–D or CONTROL–C.

NETWORKING FEATURES

RFA

The following command is executed directly by *pam*:

netunam *pathname* [*string*]
Initiate a network connection to the specified system (as indicated by *pathname*) using the specified login (as indicated by *string*). If *string* is omitted, the network connection to the specified system, if currently active, is disconnected.

DEPENDENCIES

With a non-windowed *pam* running on certain terminals, the shifted right and left arrow keys cannot be used to move the cursor on the command line.

The environment variable LANG is not set up by *pam*.

AUTHOR

Pam was developed by HP.

FILES

$HOME/.environ
$HOME/Autost

/usr/lib/nls/C/pam.cat
/dev/null

NAME

passwd – change login password

SYNOPSIS

passwd [**-f** *file*] [*name*]

DESCRIPTION

This command changes or installs a password associated with the login *name*. If *name* is omitted, *passwd* uses *getlogin*(3C) to determine the invoking user's user name. An alternate password file can be chosen with the **-f** option. The user must have read and write permission for the file given with the **-f** option. The default password file is **/etc/passwd.**

Ordinary users may change only the password which corresponds to their login *name*.

Passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has "aged" sufficiently. If "aging" is insufficient, the new password is rejected and *passwd* terminates; see *passwd*(4).

Assuming "aging" is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time the two copies of the new password are compared. If the two copies differ, *passwd* repeats the cycle of prompting for the new password, at most twice.

Passwords must be constructed to meet the following requirements:

Each password must have at least six characters. Only the first eight characters are significant.

Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means uppercase and lowercase letters.

Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

One whose effective user ID is zero is called a superuser; see *id*(1), and *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

id(1), login(1), su(1), crypt(3C), passwd(4).

EXTERNAL INFLUENCES

International Code Set Support

Characters from single-byte character code sets are supported in passwords.

STANDARDS CONFORMANCE

passwd: SVID2, XPG2

NAME

paste − merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste *file1 file2*
paste −**d***list file1 file2*
paste −**s** [−**d***list*] *file1 file2*

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns in a table and pastes them together horizontally (parallel merging). In other words, it is the horizontal counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the −**s** option form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to standard output, so *paste* can be used as the start of a pipe, or as a filter if − is used instead of a file name.

Options are:

−**d**	Without this option, the new-line characters of all but the last file (or last line in case of the −**s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).
list	One or more characters immediately following −**d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (that is, no −**s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **n** (new-line), **t** (tab), \\ (backslash), and **0** (empty string, not a null character). Quoting may be necessary if characters have special meaning to the shell. (For example, to get one backslash, use −*d*"\\\\".).
−**s**	Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with −**d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
−	Can be used in place of any file name to read a line from the standard input. (There is no prompting).

EXAMPLES

ls | paste −d" " −	list directory in one column
ls | paste − − − −	list directory in four columns
paste −s −d"\t\n" file	combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1).

NOTES

pr −**t** −**m**... works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS

line too long	Output lines are restricted to 1023 characters.
too many files	Except for −**s** option, no more than OPEN_MAX minus 3 input files can be specified (see *limits*(5)).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

NAME

pathalias – electronic address router

SYNOPSIS

pathalias [**–icv**] [**–i** host] [**–d** arg] [**–t** arg] [files]

DESCRIPTION

Pathalias computes the shortest paths and corresponding routes from one host (computer system) to all other known, reachable hosts. *Pathalias* reads host-to-host connectivity information on standard input or in the named *files*, and writes a list of host-route pairs on the standard output.

Options are:

–i Ignore case: map all host names to lowercase. By default, case is significant.

–c Print costs: print the path cost (see below) before each host-route pair.

–v Verbose: report some statistics on the standard error output.

–l *host*
Set local host name to *host*. By default, *pathalias* discovers the local host name in a system-dependent way.

–d *arg*
Declare a dead link, host, or network (see below). If *arg* is of the form "host1!host2," the link from host1 to host2 is treated as an extremely high cost (*i.e.*, DEAD) link. If *arg* is a single host name, that host is treated as dead and is used as an intermediate host of last resort on any path. If *arg* is a network name, the network requires a gateway.

–t *arg*
Trace input for link, host or network on the standard error output. The form of *arg* is as above.

The public domain version of *pathalias* includes two undocumented options, that are briefly described in the Special Options section below.

Input Format

A line beginning with white space continues the preceding line. Anything following '#' on an input line is ignored.

A list of host-to-host connections consists of a "from" host in column 1, followed by white space, followed by a comma-separated list of "to' hosts, called *links*. A link may be preceded or followed by a network character to use in the route. Valid network characters are '!' (default), '@', ':', and '%'. A link (and network character, if present) may be followed by a "cost" enclosed in parentheses. Costs may be arbitrary arithmetic expressions involving numbers, parentheses, '+', '–', '*', and '/'. The following symbolic costs are recognized:

LOCAL	25	(local-area network connection)
DEDICATED	95	(high speed dedicated link)
DIRECT	200	(toll-free call)
DEMAND	300	(long-distance call)
HOURLY	500	(hourly poll)
EVENING	1800	(time restricted call)
DAILY	5000	(daily poll, also called POLLED)
WEEKLY	30000	(irregular poll)

In addition, DEAD is a very large number (effectively infinite), and HIGH and LOW are –5 and +5 respectively, for baud-rate or quality bonuses/penalties. These symbolic costs represent an imperfect measure of bandwidth, monetary cost, and frequency of connections. For most mail traffic, it is important to minimize the number of intermediaries in a route, thus, *e.g.*, HOURLY is

far greater than DAILY / 24. If no cost is given, a default of 4000 is used.

For the most part, arithmetic expressions that mix symbolic constants other than HIGH and LOW make no sense. For example, if a host calls a local neighbor whenever there is work, and additionally polls every evening, the cost is DIRECT, **not** DIRECT+EVENING.

Some examples:

down	princeton!(DEDICATED), tilt, %thrash(LOCAL)
princeton	topaz!(DEMAND+LOW)
topaz	@rutgers(LOCAL)

If a link is encountered more than once, the least-cost occurrence dictates the cost and network character. Links are treated as bidirectional, to the extent that a DEAD reverse link is assumed unless better information is available.

The set of names by which a host is known by its neighbors is called its *aliases*. Aliases are declared as follows:

name = alias, alias ...

The name used in the route to or through aliased hosts is the name by which the host is known to its predecessor in the route.

Fully connected networks, such as the ARPANET or a local-area network, are declared as follows:

net = {host, host, ...}

The host-list may be preceded or followed by a routing character, and may be followed by a cost:

princeton-ethernet = {down, up, princeton}!(LOCAL)
ARPA = @{sri-unix, mit-ai, su-score}(DEDICATED)

The routing character used in a route to a network member is the one encountered when "entering" the network. See also the sections on *gateways* and *domains*.

Connection data may be given while hiding host names by declaring

private {host, host, ...}

Pathalias will not generate routes for private hosts, but may produce routes through them. The scope of a private declaration extends from the declaration to the end of the input file in which it appears. It is best to put private declarations at the beginning of the appropriate input file.

Output Format

A list of host-route pairs is written to the standard output, where route is a string appropriate for use with *printf*(3S), *e.g.*,

rutgers princeton!topaz!%s@rutgers

The "%s" in the route string should be replaced by the user name at the destination host. (This task is normally performed by a mailer.)

Except for *domains* (see below), the name of a network is never used in expansions. Thus, in the earlier example, the path from down to up would be "up!%s," not "princeton-ethernet!up!%s."

Gateways

A network is represented by a pseudo-host and a set of network members. Links from the members to the network have the weight given in the input, while the cost from the network to the members is zero. If a network is declared dead on the command line (with the **−d** option), the member-to-network links are marked dead, which discourages paths to members by way of

the network.

If the input also shows a link from a host to the network, then that host will be preferred as a gateway. Gateways need not be network members.

For example, suppose CSNET is declared dead on the command line and the input contains

```
CSNET = {...}
csnet-relay          CSNET
```

Then routes to CSNET hosts will use csnet-relay as a gateway.

Domains

A host or network whose name begins with '.' is called a domain. Domains are presumed to require gateways, *i.e.*, they are DEAD. The route given by a path through a domain is similar to that for a network, but here the domain name is tacked onto the end of the next host. Subdomains are permitted. For example,

```
harvard              .EDU
.EDU = {.BERKELEY}
.BERKELEY            ernie
```

yields

```
ernie                ...!harvard!ernie.BERKELEY.EDU!%s
```

Output is given for the nearest gateway to a domain, *e.g.*, the example above gives

```
.EDU                 ...!harvard!%s
```

Special Options

The public domain version of *pathalias* includes two undocumented options that rewrite named files with intermediate data of limited usage. Here are brief descriptions:

−g *file* Dump graph edges into *file* in the form "host>host" for simple connections and "host@<tab>host" for network connections (from hosts to networks only).

−s *file* Dump shortest path tree into *file* in the form "host<tab>[@]host[!](cost)", including both connections from hosts to networks and from networks to hosts. This data may be useful for generating lists of one-way connections.

BUGS

The **−i** option should be the default.

The order of arguments is significant. In particular, **−i** and **−t** should appear early.

Pathalias can generate hybrid (*i.e.* ambiguous) routes, which are abhorrent and most certainly should not be given as examples in the manual entry.

Multiple '@'s in routes are prohibited by many mailers, so *pathalias* resorts to the "magic %" rule when appropriate. This convention is not documented anywhere, including here.

AUTHOR

Pathalias was developed by Peter Honeyman and Steven M. Bellovin.

FILES

newsgroup mod.map Likely location of some input files.

NAME

pc – Pascal compiler

SYNOPSIS

pc [*options*] *file ...*

REMARKS

This manual page describes the generic HP Pascal compiler; implementation dependencies for different machines are noted as needed.

DESCRIPTION

Pc is the HP standard Pascal compiler. It accepts several types of file arguments:

(1) Arguments whose names end with **.p** are taken to be Pascal source files. They are each compiled, and each corresponding object program or module(s) is left in the current directory in a file whose name is that of the source, with **.o** substituted for **.p**. The **.o** file will be immediately deleted (leaving only the linked executable file) if only a single source is compiled and linked, if the **–C** option is specified, or if the source fails to compile correctly.

(2) All other file arguments, including those whose names end with **.o** or **.a**, are passed on to the linker (*ld*(1)) to be linked into the final program.

Options and other arguments can be passed to the compiler through the PCOPTS environment variable as well as on the command line. The compiler picks up the value of PCOPTS and places its contents before any arguments on the command line. For example (in *sh*(1) notation),

```
$ PCOPTS=-v
$ export PCOPTS
$ pc -L prog.p
```

is equivalent to

```
$ pc -v -L prog.p
```

When set, the TMPDIR environment variable specifies a directory to be used for temporary files, overriding the default directories **/tmp** and **/usr/tmp**.

Options

The following options are recognized:

–A Produce warnings for the use of non-ANSI-Pascal features. (Same as **$ANSI ON$**).

–C Suppress code generation. No **.o** files will be created and linking will be suppressed. This is effectively a request for syntax/semantic checking only (same as **$CODE OFF$**).

–c Suppress linking and only produce object (**.o**) files from source files.

–g Generate additional information needed by a symbolic debugger, and ensure that the program is linked as required. See the appropriate implementation reference manual for more information on symbolic debugging support.

–G Prepare object files for profiling with *gprof* (see *gprof*(1)).

–I*dir* Add *dir* to the list of directories that are searched for **$INCLUDE** files whose names do not begin with **/**. The directory containing the source file is always searched first, followed by any directories specified with the **–I** option, then the current working directory, and finally the standard directory */usr/include*.

–L Write a program listing to *stdout* (see the DEPENDENCIES section below for exceptions).

−l*x* Cause the linker to search first in the library named **/lib/lib***x***.a** and then in **/usr/lib/lib***x***.a** (see *ld*(1)).

−N Cause the output file from the linker to be marked as unshareable (see **−n**). For details and system defaults, refer to the linker documentation (*ld*(1)).

−n Cause the output file from the linker to be marked as shareable (see **−N**). For details and system defaults, refer to the linker documentation (*ld*(1)).

−o *outfile* Name the output file from the linker *outfile* instead of **a.out**.

−p Prepare object files for profiling with *prof* (see *prof*(1)).

−P *lines* Specifies the number of lines (including any header or trailer) which should be listed per page of generated listing (same as **$LINES n$**).

−Q Cause the output file from the linker to be marked as not demand loadable (see **−q**). For details and system defaults, refer to the linker documentation (*ld*(1)).

−q Cause the output file from the linker to be marked as demand loadable (see **−Q**). For details and system defaults, refer to the linker documentation (*ld*(1)).

−s Cause the output of the linker to be *stripped* of symbol table information (see *ld*(1) and *strip*(1)). (This option is incompatible with symbolic debugging.)

−t *c,name* Substitute or insert subprocess *c* with *name* where *c* is one or more of an implementation-defined set of identifiers indicating the subprocess(es). This option works in two modes: 1) if *c* is a single identifier, *name* represents the full pathname of the new subprocess; 2) if *c* is a set of (more than one) identifiers, *name* represents a prefix to which the standard suffixes are concatenated to construct the full path name of the new subprocesses.

The values *c* can assume are:

c compiler body (standard suffix is *pascomp*)
0 same as *c*
l linker (standard suffix is *ld*)

−v Enable verbose mode, producing a step-by-step description of the compilation process on *stderr*.

−w Suppress warning messages (same as **$WARN OFF$**).

−W *c,arg1*[*,arg2,...,argN*]

Cause *arg1* through *argN* to be handed off to subprocess *c*. The *argi* are of the form *−argoption*[*,argvalue*], where *argoption* is the name of an option recognized by the subprocess and *argvalue* is a separate argument to *argoption* where necessary. The values that *c* can assume are those listed under the **−t** option, as well as the value *d* (driver program), which has a special meaning explained below.

For example, the specification to pass the **-r** (preserve relocation information) option to the linker would be:

-W l,-r

The **−W d** option specification allows additional, implementation-specific options to be recognized and passed through the compiler driver to the appropriate compiler subprocesses. For example, on Series 300:

-W d,-U

will send the option −**U** to the driver and compiler. Furthermore, a shorthand notation for this mechanism can be used by prepending + to the option name; as in

+U

which is equivalent to the previous option expression. Note that for simplicity this shorthand is applied to each implementation-specific option individually, and that the *argvalue* is no longer separated from the *argoption* by a comma (see −**W**).

−**y** Generate additional information needed by static analysis tools, and ensure that the program is linked as required for static analysis. This option is incompatible with optimization.

−**Y** Enable 16-bit Native Language Support when parsing string literals and comments (same as **$NLS_SOURCE ON$**). Note that 8-bit parsing is always supported.

+**a** Cause the compiler to generate archived object (**.a**) files instead of simple object (**.o**) files. This allows source files containing multiple HP Pascal modules to be compiled such that each module can be linked independently. Use of this option is discouraged for portability reasons. It is provided to facilitate migration from Series 200 HP-UX releases prior to Release 5.1 (where **.a** files were always generated), to 5.1 and subsequent releases. Otherwise, it is recommended that modules needing separate linkability be placed in separate source files. To facilitate use of previously existing makefiles and scripts that depended on the archive generation, the PCOPTS environment variable can be used (e.g., set PCOPTS="+a $PCOPTS").

+**Q** *dfile* Cause *dfile* to be read before compilation of each source file. *Dfile* may contain only compiler directives.

+**R** Inhibit range checking (Same as **$RANGE OFF$**).

DIAGNOSTICS

The diagnostics produced by *pc* are intended to be self-explanatory. Occasional messages may be produced by the linker.

A list of all compiler errors may be found in **/usr/lib/paserrs**.

If a listing is requested (−**L** option), errors are written to the listing file (*stdout*). If a listing is requested and either or both of *stdout/stderr* has been redirected to something other than a terminal, errors will also be written to *stderr*. If no listing is requested (no −**L** option), errors are written to *stderr*. This effectively guarantees that *stderr* will always receive error messages, unless that would result in duplication of error messages printed on the terminal.

DEPENDENCIES

Series 300

The −**I** and −**y** options are not supported on the Series 300.

The −**L** option writes a program listing to the file given in the **$LIST filename$** option in the source, instead of to *stdout*.

The following options are implemented only on the Series 300:

+**A** Cause the compiler to always use 2-byte data and stack alignment rules instead of default 4-byte alignment rules for stacks and data objects exceeding four bytes. Refer to the reference manual for more details.

+M Cause the compiler not to generate inline code for the MC68881 floating-point coprocessor. Library routines will be referenced for math and intrinsic operations.

+bfpa Generate code that uses the HP 98248A floating point accelerator card, if it is installed at run-time. If the card is not installed, floating point operations are done on the MC68881 math coprocessor.

+ffpa Generate code for the HP 98248A floating point accelerator card. This code does not run unless the card is installed.

+U Allow certain packed fields and elements to be passed by **VAR** (Same as **$ALLOW_PACKED ON$**).

Series 800

The following options are implemented on the Series 800:

−O Turn on optimization. The compiler performs full code optimization.

−S Output an assembly file. This file is named *filename*.**s**, where *filename* is the basename of the source file.

+C Convert MPE-style file names (*file[.group[.account]]*) into HPUX-style filenames (*[[account/]group/]file*) in compiler directives like **$INCLUDE** and **$SYSINTR**. This option expects a strictly MPE-like directory structure - i.e. "group"-level directories under "account"-level directories, and source files only in "group"-level directories. See the *HP Pascal Programmers' Guide* for the Series 800 for fuller details on the semantics of this option. Same as **$CONVERT_MPE_NAMES ON$**.

+N Turn off notes (same as **$NOTES OFF$**). Notes are also automatically turned off when **−w** is specified.

+O*opt* Invoke optimizations selected by *opt*. If *opt* is '1', then only level 1 optimizations are handled. If *opt* is '2', then all optimizations are performed. The option **+O2** is the same as **−O**.

The following Series 300 options are ignored with warnings: **+M, +bfpa, +ffpa.**

FILES

file.p	input file (Pascal source file)
file.o	compiler-generated or other object file that is to be relocated at link time
a.out	linked executable output file
/usr/lib/pascomp	compiler
/usr/lib/paserrs	compiler error message file
/lib/libc.a	HP-UX system library (C-language library)
/usr/tmp/*	temporary files used by the compiler

Series 300

file.a	optionally generated object archive file
/usr/lib/escerrs	Pascal escape codes
/usr/lib/syserrs	HP-UX system messages
/usr/lib/ioerrs	Pascal I/O results
/lib/crt0.o	runtime startup
/lib/libpc.a	Pascal run-time library
/lib/libp/libpc.a	profilable Pascal run-time library
/lib/libm.a	HP−UX math library
/usr/lib/libheap2.a	Pascal run-time library, alternate heap manager

/lib/libp/libheap2.a profilable Pascal run-time library, alternate heap manager

Series 800

/usr/lib/nls/$LANG/pc_msgs.cat
error message catalog
/usr/include/pasesc.ph
symbolic definitions of library escape codes
/lib/crt0.o runtime startup
/usr/lib/libcl.a Pascal run-time library

SEE ALSO

Programming in Pascal with Hewlett-Packard Pascal, by Peter Grogono.

Series 300

HP Pascal Reference Manual.

Series 800

HP Pascal Reference Manual.
HP Pascal Programmers' Guide.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported only in strings and comments.

NAME

pg – file perusal filter for soft-copy terminals

SYNOPSIS

pg [*–number*] [**–p** *string*] [**–cefns**] [*+linenumber*] [*+/pattern*] [*file...*]

REMARKS

The decryption facilities provided by this software are under control by the United States Government and cannot be exported without special licenses. These capabilities can be sold only to domestic customers at this time.

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. (The file name – and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

–number An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

–p *string* Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

–c Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

–e Causes *pg not* to pause at the end of each file.

–f Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *–f* option inhibits *pg* from splitting lines.

–n Normally, commands must be terminated by a <*newline*> character. This option causes an automatic end of command as soon as a command letter is entered.

–s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+linenumber Start up at *linenumber*.

+/pattern/ Start up at the first line containing the regular expression pattern.

Pg looks in the environment variable *PG* to pre-set any flags desired. For example, if you prefer to view files using the **–c** mode of operation, the shell command sequence *PG='-c' ; export PG* or the *csh* command *setenv PG -c* would cause all invocations of *pg*, including invocations by programs such as *man* and *msgs*, to use this mode. Normally, the user will place the command sequence which sets up the *PG* environment variable in the *.profile* or *.cshrc* file.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
: This causes one page to be displayed. The address is specified in pages.

(+1) **l**
: With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or ^**D**
: Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or ^**L**
: Typing a single period causes the current page of text to be redisplayed.

$
: Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The Basic Regular Expression syntax (see *regexp*(5)) is supported. Regular expressions must always be terminated by a <*newline*>, even if the −*n* option is specified.

i/*pattern*/
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^
i?*pattern*?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n**
: Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p**
: Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w**
: Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
: Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <*newline*>, even if the −*n* option is specified.

h
: Help by displaying an abbreviated summary of available commands.

q or **Q**
: Quit *pg*.

!*command*

Command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the *−n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *pg* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/*	Terminal information data base
/tmp/pg*	Temporary file when input is from a pipe

SEE ALSO

crypt(1), grep(1), terminfo(4), environ(5), lang(5), regexp(5).

WARNINGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options (e.g., *crypt*(1)), terminal settings may not be restored correctly.

STANDARDS CONFORMANCE

pg: SVID2, XPG2, XPG3

NAME

pr – print files

SYNOPSIS

pr [*options*] [*file(s)*]

DESCRIPTION

Pr prints the named files on the standard output. If *file* is –, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **–s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

The following *options* can be used singly or combined in any order:

+*k*	Begin printing with page *k* (default is 1).
–*k*	Produce *k*-column output (default is 1). This option should not be used with **–m**. The options **–e** and **–i** are assumed for multi-column output.
–a	Print multi-column output across the page. This option is appropriate only with the -*k* option.
–m	Merge and print all files simultaneously, one per column (overrides the –*k*, and **–a** options).
–d	Double-space the output.
–e*ck*	Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
–i*ck*	In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).
–n*ck*	Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **–m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
–w*k*	Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise). Width specifications are only effective for multi-columnar output.
–o*k*	Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
–l*k*	Set the length of a page to *k* lines (default is 66). If *k* is less than what is needed for the page header and trailer, then the option **–t** is in effect; that is, header and trailer lines are suppressed in order to make room for text.
–h	Use the next argument as the header to be printed instead of the file name.

−p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

−f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on failure to open files.

−t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

−s*c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

RETURN VALUE

Exit values are:

0 Successful completion.
>0 One or more of the input *file(s)* do not exist or cannot be opened.

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr −3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr −e9 −t <file1 >file2
```

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1), lp(1), ul(1).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text and the arguments associated with the **−e**, **−i**, **−n**, and **−s** options as single and/or multi-byte characters.

LC_TIME determines the format and contents of date and time strings.

LANG determines the language in which messages are displayed.

If LC_CTYPE or LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *pr* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

pr: SVID2, XPG2, XPG3

NAME

prealloc – preallocate disk storage

SYNOPSIS

prealloc *name size*

DESCRIPTION

Prealloc preallocates at least *size* bytes of disk space for an ordinary file *name*, creating the file if *name* does not already exist. The space is allocated in an implementation-dependent fashion for fast sequential reads and writes of the file.

Prealloc fails and no disk space is allocated if *name* already exists and is not an ordinary file of zero length, if insufficient space is left on disk, or if *size* exceeds the maximum file size or the file size limit of the process (see *ulimit*(2)). The file is zero-filled.

DIAGNOSTICS

Upon successful completion, *prealloc* exits with a 0 status. Exit status is 1 if *name* already exists and is not an ordinary file of zero length, 2 if there is insufficient room on the disk, or 3 if *size* exceeds file size limits.

EXAMPLES

The following example preallocates 50000 bytes for the file *myfile*:

```
prealloc myfile 50000
```

WARNINGS

The allocation of the file space is highly dependent on the current disk usage. A successful return does not indicate how fragmented the file actually might be if the disk is approaching its capacity.

AUTHOR

Prealloc was developed by HP.

SEE ALSO

prealloc(2), ulimit(2).

NAME

printenv – print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

If a *name* is specified and it is not defined in the environment, *printenv* returns exit status 1, else it returns status 0.

SEE ALSO

sh(1), environ(5), csh(1).

NAME

prmail – print out mail in the post office

SYNOPSIS

prmail [user ...]

DESCRIPTION

Prmail prints the mail which waits for you, or the specified user, in the post office. The mail is not disturbed.

FILES

/usr/mail/* post office

AUTHOR

Prmail was developed by the University of California, Berkeley.

SEE ALSO

from(1), mail(1).

NAME

prof − display profile data

SYNOPSIS

prof [−**tcan**] [−**ox**] [−**g**] [−**z**] [−**h**] [−**s**] [−**m** *mdata*] [*prog*]

DESCRIPTION

Prof interprets a profile file produced by the *monitor*(3C) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t, c, a,** and **n** determine the type of sorting of the output lines:

−**t** Sort by decreasing percentage of total time (default).

−**c** Sort by decreasing number of calls.

−**a** Sort by increasing symbol address.

−**n** Sort by symbol name in ascending collation order (see Environment Variables below).

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

−**o** Print each symbol address (in octal) along with the symbol name.

−**x** Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

−**g** Include non-global symbols (static functions).

−**z** Include all symbols in the profile range (see *monitor*(3C)), even if associated with zero number of calls and zero time.

−**h** Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

−**s** Print a summary of several of the monitoring parameters and statistics on the standard error output.

−**m** *mdata* Use file *mdata* instead of **mon.out** as the input profile file.

A program creates a profile file if it has been loaded with the −**p** option of *cc*(1). This option to the *cc* command arranges for calls to *monitor*(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the −**p** option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, **mon.out** is produced in the directory current when the program terminates. If PROFDIR=string, "string/pid.progname" is produced, where *progname* consists of argv[0] with any path prefix removed, and *pid* is the program's process id. If PROFDIR=nothing, no profiling output is produced.

WARNINGS

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely, however.

Only programs that call *exit*(2) or return from *main* will cause a profile file to be produced, unless a final call to *monitor*(3C) is explicitly coded.

The use of the **−p** option *cc*(1) to invoke profiling imposes a limit of 600 functions that may have call counters established during program execution. For more counters you must call *monitor*(3C) directly. If this limit is exceeded, other data will be overwritten and the **mon.out** file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

cc(1), exit(2), profil(2), monitor(3C).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating order output by the *−n* option.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *prof* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

STANDARDS CONFORMANCE

prof: SVID2, XPG2

NAME

prs − print and summarize an SCCS file

SYNOPSIS

prs [−**d**[*dataspec*]] [−**r**[*SID*]] [−**e**] [−**l**] [−**c**] [−**a**] *file* ...

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of − is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

−**d**[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see **Data Keywords** below) interspersed with optional user supplied text.

−**r**[*SID*] Used to specify the SCCS *ID*entification (*SID*) string of a delta for which information is desired. If no *SID* is specified, the *SID* of the most recently created delta is assumed. If an *SID* is specified, it must agree exactly with an *SID* in the file (that is, the *SID* structure used by *get*(1) does not work here).

−**e** Requests information for all deltas created *earlier* than and including the delta designated via the −**r** keyletter or the date given by the −**c** option.

−**l** Requests information for all deltas created *later* than and including the delta designated via the −**r** keyletter or the date given by the −**c** option.

−**c**[*cutoff*] Cutoff date-time, in the form

YY[MM[DD[HH[MM[SS]]]]]

Units omitted from the date-time default to their maximum possible values. Thus, −**c7502** is equivalent to −**c750228235959**. One or more non-numeric characters can be used to separate the various 2-digit segments of the cutoff date (for example **-c77/2/2 9:22:25**).

−**a** Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, that is, delta type = *D*, deltas. If the −**a** keyletter is not specified, information for existing deltas only is provided.

If no option letters (or only −**a**) are given, *prs* prints the file name using the default *dataspec* and the −**e** option. This produces information on all deltas.

Data Keywords

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords. Escapes may be used as follows:

tab	\t
new-line	\n
colon	\:
backspace	\b
carriage-return	\r
form feed	\f
backslash	\\
single quote	\'

The default *dataspec* is:

":Dt:\t:DL:**nMRs:**\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	Data Item	File Sect.	Value	Fmt
:Dt:	Delta information	Delta Tbl	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:.:**Tm**:.:**Ts:**	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq # of deltas incl, excl, ign	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Nm	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes* or *no*	S

:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

If no option letters (or only **-a**) are given, *prs* prints the file name, using the default *dataspec*, and the **-e** option; thus, information on all deltas is produced.

DIAGNOSTICS

Use *help*(1) for explanations.

EXAMPLES

```
prs -d"Users and/or user IDs for :F: are:\n:UN:" s.file
```

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

```
prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file
```

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case* (when no specifications for selecting or printing are given)

```
prs s.file
```

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the **-a** keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).

SCCS User's Guide in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of dataspec as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *prs* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte character file names are not supported.

STANDARDS CONFORMANCE

prs: SVID2, XPG2, XPG3

NAME

ps, cps – report process status

SYNOPSIS

ps [**–edafl**] [**–t** *termlist*] [**–p** *proclist*] [**–u** *uidlist*] [**–g** *grplist*]

cps [**–edafl**] [**–t** *termlist*] [**–p** *proclist*] [**–u** *uidlist*] [**–g** *grplist*]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

In the HP Clustered environment, *cps* can be used to obtain a cluster wide process listing. A seperate listing is produced for each member of the cluster preceded by the cluster member's name. *Cps* only reports on other members of the HP Cluster when the **e**, **d**, **a**, **t**, **p**, **u** or **g** options have been specified.

Options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

–e Print information about all processes.

–d Print information about all processes, except process group leaders.

–a Print information about all processes, except process group leaders and processes not associated with a terminal.

–f Generate a *full* listing. (See below for meaning of columns in a full listing.)

–l Generate a *long* listing. See below.

–m When the -f option is given, print the current argument information from the address space of the process itself, rather than the argument information at the time of exec.

–t *termlist* Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., **tty04**) or if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).

–p *proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.

–u *uidlist* Restrict listing to data about processes whose real user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the **–f** option is used, in which case the login name will be printed.

–g *grplist* Restrict listing to data about processes whose process group leaders are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear. **All** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:

0	swapped;
1	in core;
2	system process;
4	locked in core (e.g., for physical I/O);
10	being swapped;

20 being traced by another process;
40 another tracing flag;

S (l) The state of the process:
0 non-existent;
S sleeping;
W waiting;
R running;
I intermediate;
Z terminated;
T stopped;
X growing.

UID (f,l) The real user ID number of the process owner; the login name is printed under the **−f** option.

PID (all) The process ID of the process; it is possible to kill a process if you know this datum.

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

PRI (l) The priority of the process; higher numbers mean lower priority.

NI (l) Nice value; used in priority computation.

ADDR (l) The memory address of the process, if resident; otherwise, the disk address.

SZ (l) The size in blocks of the core image of the process.

WCHAN (l) The event for which the process is waiting or sleeping; if blank, the process is running.

STIME (f) Starting time of the process. The starting date is printed instead if the elapsed time is greater than 24 hours.

TTY (all) The controlling terminal for the process.

TIME (all) The cumulative execution time for the process (reported in the form "min:sec").

COMD (all) The command name; the full command name and its arguments are printed under the **−f** option. This field is renamed **COMMAND** except when the **−l** option is specified.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked <**defunct**> (see "zombie process" in *exit*(2)).

The time printed in the **STIME** field is the time when the process was forked, *not* the time when it was modified by *exec*.

Under the **−f** option, *ps* tries to determine the command name and arguments given when the process was created. If the **−m** option is given, ps makes an educated guess as to the command and arguments currently residing within the address space of the process. It does this by examining memory and the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the command line cannot be counted on too much. If the **−m** option is not given, ps presents the command and arguments given at the time of process creation.

Since process information may be changing as ps retrieves it, and a process performing an exec(2) may pass faulty command name information, the command line data may be inconsistent. If such a situation is detected, the command name as it would appear without the **−f** option is printed in square brackets. This is the actual pathname of the file read by exec(2) or copied from the parent process by fork(2).

To make *ps* output safer to display and easier to read, all control characters in the **COMD** field are mapped to "visible" equivalents. These are of the form ^*C*, where the original character was in the range 0 - 037 and ^*C* is that value plus 040.

EXAMPLES

To generate a full listing of all processes currently running on your machine, type:

ps -ef

To check if a certain process exists on the machine for example, *cron*(1M) (a clock daemon), check the far right column for **cron**.

FILES

/etc/passwd	supplies UID information
/etc/ps_data	internal data structure
/dev	searched to find terminal ("tty") names
/dev/mem	memory - used with -m option
/dev/kmem	virtual memory - used with -m option
/dev/swap	the default swap device - used with -m option

SEE ALSO

kill(1), nice(1), acctcom(1M), exec(2), fork(2), exit(2).

WARNINGS

Things can change while *ps* is running; the picture it gives is only a snapshot in time. Some data printed for defunct processes are irrelevant.

If two special files for terminals are located at the same select code, they are reported in the order in which they appear in **/dev**, not in alphabetical order.

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ps* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

ps: SVID2, XPG2, XPG3

NAME

psqlc, psqlpas, psqlfor, psqlcbl – preprocess ALLBASE/HP-UX HP SQL source programs written in C, Pascal, FORTRAN and COBOL

SYNOPSIS

psqlc **−s** [**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.c*]
psqlc *DBEnvirnomentName* [**−o** *ownername*] [**−m** *modulename*] [**−d**[**−r**]]
[**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.c*]
psqlpas **−s** [**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.p*]
psqlpas *DBEnvirnomentName* [**−o** *ownername*] [**−m** *modulename*] [**−d**[**−r**]]
[**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.p*]
psqlfor **−s** [**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.f*]
psqlfor *DBEnvirnomentName* [**−o** *ownername*] [**−m** *modulename*] [**−d**[**−r**]]
[**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.f*]
psqlcbl **−s** [**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.cbl*]
psqlcbl *DBEnvirnomentName* [**−o** *ownername*] [**−m** *modulename*] [**−d**[**−r**]]
[**−i** *sourcefilename.sql*] [**−p** *sqloutfilename.cbl*]

REMARKS

The ALLBASE/HP-UX product must be previously installed on the system for *psqlc*, *psqlpas*, *psqlfor*, or *psqlcbl* to function.

DESCRIPTION

Psqlc, *psqlpas*, *psqlfor*, and *psqlcbl* invoke the C, Pascal, FORTRAN, and COBOL preprocessors, respectively, for programmatically accessing an ALLBASE/HP-UX relational DataBase Environment (DBEnvironment). *Psqlc*, *psqlpas*, *psqlfor*, and *psqlcbl* can be executed by all system users.

Options

−s Specify that the preprocessor checks only the syntax of embedded SQL commands.

DBEnvironmentName
Identify the DBEnvironment in which a module is stored.

−o *ownername* Associate the stored module with a user's login name, a classname, or a groupname. You can specify an *ownername* for the module only if you have DBA authority in the DBEnvironment where the module is to be stored. If not specified, the default *ownername* is your login name.

−m *modulename*
Assign a name to the stored module. *Modulenames* must follow the rules governing HP SQL basic names as described in the *ALLBASE/HP-UX SQL Reference Manual*. If a *modulename* is not specified, the preprocessor uses the PROGRAM Statement name as the *modulename*.

−d Delete any module currently stored in the DBEnvironment by the *modulename* and *ownername* specified in the options list. If the **−d** option is not specified, the module is retained and all existing run authorities for that module are preserved.

−r Revoke all existing run authorities for a module when a program being preprocessed already has a stored module. The **−r** option cannot be specified unless **−d** is also specified. If the **−r** option is not specified, all existing run authorities for that module are preserved.

−i *sourcefilename.sql*
Identify the name of the input file containing the source code being preprocessed. If *sourcefilename.sql* is not specified, a default file **sqlin** is assumed. It is recommended but not required that the source file name have a file

extension of **.sql**.

-p *sqloutfilename*

Identify the name of the output file containing the preprocessor generated code. If *sqloutfilename* is not specified, the preprocessor generated code is written to a file with the same name as the source file name but with an appended file extension indicating the language of the source code: **.c** (C), **.p** (Pascal), **.f** (FORTRAN), or **.cbl** (COBOL).

DEPENDENCIES

Series 300

The *psqlcbl* preprocessor is not currently supported.

AUTHOR

Psqlc, *psqlpas*, *psqlfor*, and *psqlcbl* were developed by Hewlett-Packard.

FILES

/usr/bin/hpdbdaemon	Cleanup daemon program file.
/usr/bin/psqlc	C preprocessor program file.
/usr/bin/psqlpas	Pascal preprocessor program file.
/usr/bin/psqlfor	FORTRAN preprocessor program file.
/usr/bin/psqlcbl	COBOL preprocessor program file.
/usr/lib/hpsqlproc	HP SQL program file.
/usr/lib/hpsqlcat	HP SQL message catalog file.
/usr/lib/libsql.a	Run time routine library file.
/usr/lib/nls/$LANG/hpsqlcat	Localized HP SQL message catalog file.

SEE ALSO

ALLBASE/HP-UX HP SQL C Application Programming Guide.
ALLBASE/HP-UX HP SQL Pascal Application Programming Guide.
ALLBASE/HP-UX HP SQL FORTRAN Application Programming Guide.
ALLBASE/HP-UX HP SQL COBOL Application Programming Guide.

EXTERNAL INFLUENCES

Environment Variables

For preprocessors *psqlc*, *psqlpas*, and *psqlfor*, LANG determines the language in which messages are displayed.

International Code Set Support

For preprocessors *psqlc*, *psqlpas*, and *psqlfor*, single- and multi-byte character code sets are supported.

NAME

ptx – permuted index

SYNOPSIS

ptx [options] [input [output]]

DESCRIPTION

Ptx generates the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted (see *sort*(1) and Environment Variables below). Finally, the sorted lines are rotated so the keyword comes at the middle of each line. *Ptx* output is in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where **.xx** is assumed to be an *nroff* or *troff* macro provided by the user, or provided by the *mptx* macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

–f Fold uppercase and lowercase letters for sorting.

–t Prepare the output for the phototypesetter by using a line length of 100.

–w *n* Use the next argument, *n*, as the length of the output line. The default line length is 72 characters for *nroff* and 100 for *troff*.

–g *n* Use the next argument, *n*, as the number of characters that *ptx* will reserve in its calculations for each gap among the four parts of the line as finally printed. The default gap is 3.

–o *only* Use as keywords only the words given in the *only* file.

–i *ignore* Do not use as keywords any words given in the *ignore* file. If the **–i** and **–o** options are missing, use **/usr/lib/eign** as the *ignore* file.

–b *break* Use the characters in the *break* file to separate words. Tab, new-line, and space characters are *always* used as break characters. Punctuation characters are treated as part of the word in the absence of this option.

–r Take any leading non-blank characters of each input line to be a reference identifier (as to a page or chapter), separate from the text of the line. Attach that identifier as a 5th field on each output line.

FILES

/usr/lib/eign
/bin/sort
/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), mm(5).

BUGS

Line length counts do not account for overstriking or proportional spacing.
Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

LC_CTYPE determines the default break characters.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *ptx* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

International Code Set Support

Single-byte character code sets are supported.

NAME

pwd – working directory name

SYNOPSIS

pwd [*-H*]

DESCRIPTION

Pwd prints the path name of the working (current) directory. The **–H** option causes *pwd* to reveal hidden directories (context-dependent files) in the path name and append a plus sign (+) to them. See *cdf*(4).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to the system manager.

EXAMPLES

This command lists the path of the current working directory. If your home directory were **/mnt/staff** and the command **cd camp/nevada** were executed from the home directory, typing **pwd** would produce the following:

/mnt/staff/camp/nevada

AUTHOR

Pwd was developed by AT&T and HP.

SEE ALSO

cd(1), cdf(4).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *pwd* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

pwd: SVID2, XPG2, XPG3

NAME

pwget, grget − get password and group information

SYNOPSIS

pwget [**−n** *name* | **−u** *uid*]

grget [**−n** *name* | **−g** *gid*]

DESCRIPTION

Pwget and *grget* locate and display information from **/etc/passwd** and **/etc/group**.

The standard output of *pwget* contains lines of colon-separated password information of the format used in the **/etc/passwd** file (see *passwd*(4)).

The standard output of *grget* contains lines of colon-separated group information of the format used in the **/etc/group** file (see *group*(4)).

With no options, *pwget* and *grget* get all entries with *getpwent*(3C) or *getgrent*(3C), respectively, and output a line for each entry found.

Options

When an option is given, only a single entry is printed.

The options for *pwget* are:

−n *name* Output the first entry that matches *name* using *getpwnam* (on *getpwent*(3C)).

−u *uid* Output the first entry that matches *uid* using *getpwuid* (on *getpwent*(3C)).

The options for *grget* are:

−n *name* Output the first entry that matches *name* using *getgrnam* (on *getgrent*(3C)).

−g *gid* Output the first entry that matches *gid* using *getgrgid* (on *getgrent*(3C)).

NETWORKING FEATURES

NFS

If Yellow Pages (YP) are in use, these commands provide password and group information based on the YP version of the password and group databases in addition to the local password and group files.

RETURN VALUE

These commands return **0** upon success, **1** when a specific search fails, and **2** upon error.

DEPENDENCIES

NFS:

WARNINGS

If the Yellow Pages network database is in use and the YP client daemon (*ypbind*) is not bound to a YP server daemon (see *ypserv*(1M)), these utilities will wait until such a binding is established. These commands can be terminated in this condition by sending a SIGINT signal to the process (see *kill*(1)).

See *ypmatch*(1), and *ypserv*(1M) in the *Networking Reference*.

AUTHOR

Pwget and *grget* were developed by HP.

FILES

/etc/group local group data file

/etc/passwd local password data file

SEE ALSO

getgrent(3C), getpwent(3C), group(4), passwd(4).

NAME

ratfor – rational Fortran dialect

SYNOPSIS

ratfor [*options*] [*file* ...]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:

```
{ statement; statement; statement }
```

decision-making:

```
if (condition) statement [ else statement ]
switch (integer value) {
        case integer:    statement
        ...
        [ default: ]     statement
}
```

loops:

```
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [ until (condition) ]
break
next
```

and some syntactic sugar to make programs easier to read and write:

free form input:
multiple statements per line and automatic continuation of lines

comments:
this is a comment.

compiler directives:
directives beginning with a dollar sign (**$**) in column one are passed through to the compiler unchanged.

translation of relationals:
>, >=, etc., become **.GT.**, **.GE.**, etc.

return expression to caller from function:
return (expression)

define:
define *name replacement*

include:
include *file*

Options

The following options are supported:

−h Cause quoted strings to be turned into Hollerith constructs as, for example, **27H.**

−C Copy comments to the output and format them neatly.

−6*c* Normally, continuation lines are marked with an **&** in column 1. The **−6***c* option makes the continuation character *c* and places it in column 6.

Ratfor is best used with *f77*(1).

DEPENDENCIES

Series 300

Options can be passed to *ratfor* through *f77*(1) by using the **−W** option specifier.

Series 800

The **−6x** option must be used for successful automatic line continuation.

SEE ALSO

f77(1).

Ratfor: A Preprocessor for a Rational FORTRAN, tutorial in *HP-UX Concepts and Tutorials: Programming Environment.*

B. W. Kernighan and P. J. Plauger, *Software Tools,* Addison-Wesley, 1976.

NAME

rcs – change RCS file attributes

SYNOPSIS

rcs [*options*] *file* ...

DESCRIPTION

Rcs creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For *rcs* to work, the caller's login name must be on the access list, except if the access list is empty, if the caller is the owner of the file or the superuser, or if the **–i** option is present.

The caller of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. *Rcs* creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, *rcs* always creates a new file. On successful completion, *rcs* deletes the old one and renames the new one. This strategy makes links to RCS files useless.

Files ending in **",v"** are RCS files, all others are working files. If a working file is given, *rcs* tries to find the corresponding RCS file first in directory **./RCS** and then in the current directory, as explained in *rcsintro*(5).

Options

–a*logins* Appends the login names appearing in the comma-separated list *logins* to the access list of the RCS file.

–A*oldfile* Appends the access list of *oldfile* to the access list of the RCS file.

–c*string* Sets the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword Log during check out (see *co*(1)). This is useful for programming languages without multi-line comments. During *rcs* **–i** or initial *ci*(1), the comment leader is guessed from the suffix of the working file.

–e[*logins*] Erases the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, the entire access list is erased.

–i Creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, *rcs* tries to place it first into the subdirectory **./RCS**, and then into the current directory. If the RCS file already exists, an error message is printed.

–l[*rev*] Locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with *ci*(1) or *rcs* **–u** (see below).

–L Sets locking to *strict*. Strict locking means that the owner of an RCS file is not exempt from locking for check in. This option should be used for files that are shared.

–n*name*[:[*rev*]] Associates the symbolic name *name* with the branch or revision *rev*. *Rcs* prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic name is associated with the latest revision on the trunk. If *:rev* is omitted, the symbolic name is deleted.

–N*name*[:[*rev*]] Same as **–n**, except that it overrides a previous assignment of *name*.

–o*range* Deletes ("obsoletes") the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form

rev1−*rev2* means revisions *rev1* to *rev2* on the same branch, −*rev* means from the beginning of the branch containing *rev* up to and including *rev*, and *rev*− means from revision *rev* to the head of the branch containing *rev*. None of the outdated revisions may have branches or locks.

−q Quiet mode; diagnostics are not printed.

−s*state*[:*rev*] Sets the state attribute of the revision *rev* to *state*. If *rev* is omitted, the latest revision on the trunk is assumed. If *rev* is a branch number, the latest revision on that branch is assumed. Any identifier is acceptable for *state*. A useful set of states is **Exp** (for experimental), **Stab** (for stable), and **Rel** (for released). By default, *ci*(1) sets the state of a revision to **Exp**.

−t[*txtfile*] Writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, *rcs* prompts the user for text supplied from the standard input, terminated with a line containing a single "." or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the **−i** option is present, descriptive text is requested even if **−t** is not given. The prompt is suppressed if the standard input is not a terminal.

−u[*rev*] Unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is omitted, the latest lock held by the caller is removed. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single "." or control-D.

−U Sets locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for check in. This option should NOT be used for files that are shared. The default (**−L** or **−U**) is determined by your system administrator.

Access Control Lists (ACLs)

Do not add optional ACL entries to an RCS file, because they are deleted when the file is updated. The resulting access modes for the new file might not be as desired.

DIAGNOSTICS

The RCS filename and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 if unsuccessful.

EXAMPLES

The command:

rcs −ajane,mary,dave,jeff vision

adds the names **jane**, **mary**, **dave**, and **jeff** to the access list of the RCS file **vision,v**.

The command:

rcs −c'*tab****' vision**

sets the comment leader to *tab** for the file **vision**.

The command:

rcs −Nsso/6_0:38.1 vision

associates the symbolic name **sso/6_0**, with revision **38.1** of the file **vision**.

The command:

rcs −l38.1 vision,v

locks revision **38.1** of the file **vision,v** so that only the locker is permitted to check in (see *ci*(1)) the next revision of the file. This command prevents two or more people from simultaneously revising the same file and inadvertently overwriting each other's work.

AUTHOR

Rcs was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

co(1), ci(1), rcsdiff(1), rcsintro(5), rcsmerge(1), rlog(1), rcsfile(4), acl(5).

NAME

rcsdiff – compare RCS revisions

SYNOPSIS

rcsdiff [**−bcefhn**] [**−r**rev1] [**−r**rev2] file ...

DESCRIPTION

Rcsdiff compares two revisions of each given RCS file and creates output very similar to *diff*(1). A file name ending in "**,v**" is an RCS file name, otherwise it is a working file name. *Rcsdiff* derives the working file name from the RCS file name and vice versa, as explained in *rcsintro*(5). Pairs consisting of both an RCS and a working file name may also be specified.

The options **−b**, **−e**, **−f**, and **−h** have the same effect as described in *diff*(1); option **−n** generates an edit script of the format used by RCS.

The option **−c** generates a diff with lines of context. The default is to present 3 lines of context, but may be changed, for example, to 10, by **−c10**. With **−c** the output format is modified slightly from the normal *diff*(1) output. The "context" output begins with identification of the files involved and their creation dates and then each change is separated by a line with a dozen "*" (stars). The lines removed from file1 are marked with "-" (dashes); those added to file2 are marked with "+" (pluses). Lines that are changed from one file to the other are marked in both files with "!" (exclamation marks).

If both *rev1* and *rev2* are omitted, *rcsdiff* compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what you changed since the last check in.

If *rev1* is given, but *rev2* is omitted, *rcsdiff* compares revision *rev1* of the RCS file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, *rcsdiff* compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* may be given numerically or symbolically.

EXAMPLES

The command:

```
rcsdiff f.c
```

compares the latest trunk revision of RCS file **f.c,v** and the contents of working file **f.c**.

AUTHOR

Rcsdiff was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

ci(1), co(1), diff(1), ident(1), rcs(1), rcsintro(5), rcsmerge(1), rlog(1), rcsfile(4).

NAME

rcsmerge − merge RCS revisions

SYNOPSIS

rcsmerge −**r**rev1 [−**r**rev2] [−**p**] file

DESCRIPTION

Rcsmerge incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If **−p** is given, the result is printed on the standard output, otherwise the result overwrites the working file.

A file name ending in "**,v**" is an RCS file name, otherwise it is a working file name. *Rcsmerge* derives the working file name from the RCS file name and vice versa, as explained in *rcsintro*(5). A pair consisting of both an RCS and a working file name may also be specified.

Rev1 may not be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* may be given numerically or symbolically.

Rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained for the −**j** option of *co*(1). The command is useful for incorporating changes into a checked-out revision.

EXAMPLES

Suppose you have released revision 2.8 of **f.c**. Assume furthermore that you just completed revision 3.4, when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file **f.c** and execute:

```
rcsmerge −p −r2.8 −r3.4 f.c >f.merged.c
```

Then examine f.merged.c. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute: **co −j**:

```
ci −r2.8.1.1 f.c
co −r3.4 −j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in **f.c**:

```
rcsmerge −r2.8 −r2.4 f.c
```

Note the order of the arguments, and that **f.c** will be overwritten.

WARNINGS

Rcsmerge does not work for files that contain lines with a single ".".

AUTHOR

Rcsmerge was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

ci(1), co(1), merge(1), ident(1), rcs(1), rcsdiff(1), rlog(1), rcsfile(4).

NAME

readmail – read mail from specified mailbox

SYNOPSIS

readmail [**–p**] [**–n**] [**–f** *filename*] [**–h**]
readmail [**–p**] [**–n**] [**–f** *filename*] [**–h**] *number-list*
readmail [**–p**] [**–n**] [**–f** *filename*] [**–h**] *pattern*

DESCRIPTION

Readmail is a program that gives you the functionality of the **mailx(1)** "~r" command from the editor of your choice. There are three different ways of using the program.

First off, if you are actually creating a reply to a message from within **elm(1)** then *readmail* without any arguments will include a summary of the headers and the body of the message being replied to. If the you aren't currently editing a message the program will return an error.

Secondly, if you want to include certain messages, you can specify them by listing their ordinal locations in the mail file (that is, their "message numbers") up to 25 at a time. The *meta*-number '$' is understood to mean the last message in the mailfile. Similarly, '*' is understood to represent every message in the file (that is, 1 2 3 4 5 ... $)

Finally, you can also specify a pattern that occurs in one of the messages as a way of including it. This pattern can be typed in directly (no quotes) if the words are separated by a single space in the actual message. The pattern matching is case sensitive, so "Hello" and "hello" are not the same thing.

Other options are:

–f *folder* — This indicates that you would rather use the file specified for the operations specified rather than the incoming mailbox.

–h — This instructs the program to include the entire header of the matched message or messages when displaying their text. (default is to display the *From: Date:* and *Subject:* lines only)

–n — This instructs the program to exclude all headers. This is used mostly for extracting files mailed and such.

–p — This indicates that the program should put form-feeds (control-L) between message headers. This is very useful for printing sets of messages.

EXAMPLES

First off, to use this from within **vi(1)** to include the text of the current message at the end of the edit buffer, you could use the command;

```
!!readmail
```

(when you hit the second '!', the editor will put you at the bottom of the screen with the '!' prompt)

Another convenient use of the command is via a **csh(1)** alias similar to:

```
alias rd        'readmail $ | page'
```

which then makes it very easy, in conjunction with a program like **newmail(1)** to peruse your mail as it arrives without necessarily spending the time involved in invoking a mail system.

AUTHOR

readmail was developed by Hewlett-Packard Company.

FILES

/usr/mail/*username*
The incoming mailbox

$HOME/.elm/readmail
The temporary file for **elm(1)**

SEE ALSO

elm(1), mailx(1), newmail(1), vi(1).

NOTE

For performance reasons when the program is given a list of message numbers to display, it will sort them in ascending order, so "1 3 2" would output identically to "1 2 3".

NAME

rev – reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single- and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *rev* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

rlog − print log messages and other information on RCS files

SYNOPSIS

rlog [options] file ...

DESCRIPTION

Rlog prints information about RCS files. Files ending in **",v"** are RCS files, all others are working files. If a working file is given, *rlog* tries to find the corresponding RCS file first in directory **./RCS** and then in the current directory, as explained in *rcsintro*(5).

Rlog prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, *rlog* prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, *rlog* prints complete information. The options below restrict this output.

Options

−d*dates* Prints information about revisions with a check in date/time in the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2* (inclusive). A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d*, *d1*, and *d2* are in the free format explained in *co*(1). Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

−h Prints only RCS file name, working file name, head, access list, locks, symbolic names, and suffix.

−l[*lockers*] Prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.

−L Ignores RCS files that have no locks set; convenient in combination with **−R**, **−h**, or **−l**.

−r*revisions* Prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1*−*rev2* means revisions *rev1* to *rev2* on the same branch, −*rev* means revisions from the beginning of the branch up to and including *rev*, and *rev*− means revisions starting with *rev* to the head of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

−R Prints only the name of the RCS file; convenient for translating a working file name into an RCS file name.

−s*states* Prints information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

−t Prints the same as **−h**, plus the descriptive text.

−w[*logins*] Prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

Rlog prints the intersection of the revisions selected with the options **−d**, **−l**, **−s**, **−w**, and **−r**.

EXAMPLES

The following command prints the names of all RCS files in the subdirectory named **RCS** that have locks:

```
rlog −L −R RCS/*,v
```

The following command prints the headers of those files:

```
rlog −L −h RCS/*,v
```

The following command prints the headers plus the log messages of the locked revisions:

```
rlog −L −l RCS/*,v
```

The following command prints complete information:

```
rlog RCS/*,v
```

DIAGNOSTICS

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 if unsuccessful.

AUTHOR

Rlog was developed by Walter F. Tichy, Purdue University, West Lafayette, IN 47907.
Revision Number: 3.0; Release Date: 83/05/11.

SEE ALSO

ci(1), co(1), ident(1), rcs(1), rcsdiff(1), rcsintro(5), rcsmerge(1), rcsfile(4).

NAME

rm, rmdir − remove files or directories

SYNOPSIS

rm [−**fri**] *file* ...
rmdir [−**fi**] *dir* ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a user has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input (see Access Control Lists below). If that line begins with **y** the file is deleted; otherwise, the file remains. No questions are asked when the −**f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument −**r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the −**i** (interactive) option is in effect, *rm* and *rmdir* ask whether to delete each file or directory. If −**r** is specified with −**i**, *rm* asks whether to examine each directory before interactively removing files.

Rmdir removes entries for the named directories, which must be empty.

Access Control Lists (ACLs).

If a file has optional ACL entries, *rm* displays a plus sign (+) after the file's permissions. The permissions shown summarize the file's *st_mode* value returned by *stat*(2). See also *acl*(5).

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file .. merely to avoid the consequences of inadvertently doing something like:

rm −**r** .*

EXAMPLES

To remove files with a prompt for verification:

rm -i *filenames*

To remove an empty directory, type:

rmdir *directoryname*

To remove all the files in a directory type:

rm -i *directoryname*/*

Note that the above command removes files only, and leaves any directories in *directoryname* alone.

A powerful and dangerous command to remove a directory would be:

rm -rf *directoryname*

This command removes all files and directories from *directoryname* without prompting for verification to remove the files nor the directories. Therefore, this command should only be used when you are absolutely sure that all the files and directories in *directoryname* and *directoryname* itself are to be removed.

DEPENDENCIES

RFA and NFS

Rm does not display a plus sign (+) to indicate the existence of optional access control list

entries, when asking for confirmation before removing a networked file.

SEE ALSO

stat(2), unlink(2), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of filenames as single and/or multi-byte characters for the *rmdir* command.

LANG determines the language in which messages are displayed.

LANG and LC_CTYPE determine the local language equivalent of y (for yes/no) queries.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, rm and rmdir behave as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

rm: SVID2, XPG2, XPG3

rmdir: SVID2, XPG2, XPG3

NAME

rmb, rmbhil, rmbkbd, rmbtmr, rmbxfr – HP BASIC/UX environment

SYNOPSIS

rmb [*options*] [*autostart_file*]

Remarks:

This command requires installation of optional BASIC/UX software (not included with the standard HP-UX operating system) before it can be used.

The keywords **disk** and **disc** are parsed as equivalents by *rmb* and related commands and programs.

DESCRIPTION

This command invokes the HP BASIC/UX interpreter which can be used to execute BASIC commands or run BASIC programs.

The BASIC **EXECUTE** command is used to temporarily exit the BASIC environment and spawn a new Bourne shell from which any number of HP-UX commands can be executed. Ctrl-D terminates the shell and returns to BASIC.

Options

The command line options are:

–b This option causes *rmb* to print a screen on startup that resembles the bootrom screen. This screen contains information about hardware and software configurations. Several additional fields have been added to the standard bootrom screen:

Field	Contents
workspace	size of the *rmb* workspace
swap	information about swap device locations
mounted disks	information about mounted disk locations
HP-UX	information about the HP-UX version

In addition, interface card lines provide information about whether a swap device is on the interface (which precludes BURST I/O), and which device file (if any) is used to access the interface.

–c *file* Specifies that the file *file* should be used as the configuration file rather than **$HOME/.rmbrc**.

–e Enables "ignore compatibility errors" mode. This causes incompatible statements from the BASIC workstation to be ignored rather than flagged as errors.

–i Causes *rmb* to run **/usr/lib/rmb/rmbclean** at startup time to reclaim orphaned lockfiles and IPC resources.

–k Causes *rmb* to run **/usr/lib/rmb/rmbconfig –b** at startup time to update the kernel configuration file (**/usr/lib/rmb/rmbbootinfo**).

–l *opt* This option specifies for *rmb* to attempt to lock the indicated program segments into memory. This can result in an increase in program performance at the expense of other programs on the machine. The *opt* parameter can be any combination of the characters:

char	segment	size
t	text	2Mb
d	data	200Kb
w	workspace	configurable

The parameter **all** can also be used for *opt* to indicate locking all the segments. Note the approximate size of physical RAM consumed for each segment locked. Note also that each of the *rmb* daemons attempts to lock itself into memory. Program locking requires either superuser(root) capabilities, or that the user be a member of the privgrp MLOCK (see *setprivgrp*(1M)).

−n Specifies that the global configuration file **/usr/lib/rmb/rmbrc** is not to be read.

−r *num* This option specifies for *rmb* to attempt to run at the real-time priority specified by *num*. Valid values for *num* are 0 to 127, where 0 is the highest priority. Note that each of the *rmb* daemons will also attempt to run at this priority. Real-time priority requires either superuser capabilities, or that the user be a member of the privgrp RTPIO (see *setprivgrp*(1M)).

−t Enables terminal keymappings as described in *Using the BASIC/UX System*.

−w *num*[KM] Specifies the *rmb* workspace to be *num* bytes in size. The optional K suffix can be added to represent Kilobytes, or M for Megabytes. The default workspace size is 1Mb.

The following standard **X** command line options along with some additional options are supported by *rmb* when running in **X**.

−bd *color* This option specifies the color to use for the border of the window. The default is the foreground color used in the window. The corresponding resource name is **borderColor** (class **BorderColor**).

−bg *color* This option specifies the color to use for the background of the window. The default is "black." The corresponding resource name is **background** (class **Background**).

−buf *number* This option specifies the number of lines to save in the alpha buffer. The default is 52. The corresponding resource name is **bufferSize** (class **BufferSize**).

−bw *number* This option specifies the width in pixels of the border surrounding the window. The corresponding resource name is **borderWidth** (class **BorderWidth**).

−cm *mode* This option specifies the type of alpha cursor to be used in the window. The valid modes are "Underscore" (default) and "Block". The corresponding resource name is **cursorMode** (class **CursorMode**).

−display *display* This option specifies that *display* is to be used as the **X** windows display server for the *rmb* window.

−fg *color* This option specifies the color to use for displaying text. The default is "white." The corresponding resource name is **foreground** (class **Foreground**).

−fn *font* This option specifies a font to be used when displaying alpha text. The corresponding resource name is **font** (class **Font**).

−geometry *geometry* This option specifies the preferred size and position of the *rmb* window. The corresponding resource name is **geometry** (class **Geometry**).

−iconic This option indicates that *rmb* should be placed on the display in icon form. The corresponding resource name is **iconic** (class **Iconic**).

+iconic This option indicates that *rmb* should not be placed on the display in icon form. The corresponding resource name is **iconic** (class **Iconic**).

−retain This option indicates that the *rmb* window should be retained. The corresponding resource name is **retainedWindow** (class **RetainedWindow**).

+retain This option indicates that the *rmb* window should be not be retained. The corresponding resource name is **retainedWindow** (class **RetainedWindow**).

−title This option specifies a window title for the *rmb* window. This string may be used by the window manager when displaying the window or icon. The corresponding resource name is **windowName** (class **WindowName**).

−x *display* This option specifies that *display* is to be used as the **X** windows display server for the *rmb* window. This option is provided for compatibility with previous versions of *rmb*. It may not be supported in future releases as the **−display** option accomplishes the same task.

−xrm *resourcestring* This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

AUTHOR

Rmb was developed by HP

FILES

/dev/rmb/*	location of device files used by rmb
/usr/bin/rmb	rmb executable
/usr/bin/rmbbuildc	program for building CSUBs
/usr/include/csubdecl.h	include file for compiling CSUBs
/usr/lib/librmb.a	rmb library for linking CSUBs
/usr/lib/rmb/.lock/lock*	resource information lockfile
/usr/lib/rmb/demos/*	demonstration programs and manual examples
/usr/lib/rmb/fonts/*	default bitmapped character fonts
/usr/lib/rmb/ipcclean	ipcclean program
/usr/lib/rmb/newconfig/*	example configuration files
/usr/lib/rmb/rmbbootinfo	rmb configuration information file
/usr/lib/rmb/rmbclean	rmbclean script
/usr/lib/rmb/rmbconfig	rmbconfig program
/usr/lib/rmb/rmbdfile	kernel configuration file scanner
/usr/lib/rmb/rmbhil	HIL interface daemon
/usr/lib/rmb/rmbkill	program to kill EXECUTE processes after RESET
/usr/lib/rmb/rmbkbd	keyboard daemon
/usr/lib/rmb/rmbrc	global configuration file
/usr/lib/rmb/rmbtmr	timer daemon
/usr/lib/rmb/rmbxfr	TRANSFER statement daemon
/usr/lib/rmb/utils/*	rmb utilities and CSUBs
$HOME/.rmbrc	the local user configuration file

SEE ALSO

Using the BASIC/UX System, BASIC Language Reference.

INTERNATIONAL SUPPORT

rmb: messages

NAME

rmbbuildc – generate a CSUB library

SYNOPSIS

rmbbuildc [memory_size]

Remarks:

This command requires installation of optional BASIC/UX software (not included with the standard HP-UX operating system) before it can be used.

The keywords **disk** and **disc** are parsed as equivalents by *rmb* and related commands and programs.

DESCRIPTION

Rmbbuildc is a program which generates a CSUB library in a BASIC PROG file, which can then be loaded by *rmb*.

Rmbbuildc interactively prompts the user with the necessary information about the interface of each CSUB. Each prompt is explained in detail in the manual "Developing CSUBs for BASIC/UX". The program takes an optional parameter which specifies the number of bytes to allocate at run-time. If this parameter is omitted, the program will allocate 1,000,000 bytes, by default.

AUTHOR

Rmbbuildc was developed by HP.

SEE ALSO

Developing CSUBs for BASIC/UX.

NAME

rmbclean, ipcclean – clean up RMB lock files and IPC resources

SYNOPSIS

rmbclean

Remarks:

This command requires installation of optional BASIC/UX software (not included with the standard HP-UX operating system) before it can be used.

The keywords **disk** and **disc** are parsed as equivalents by *rmb* and related commands and programs.

DESCRIPTION

Rmbclean is a script to find orphaned *rmb* lock files and clean them up. This involves removing any IPC resources remaining for the process that created the lockfile, and removing the lockfile itself. *Rmbclean* does not touch lockfiles for which an *rmb* program is still running. *Rmbclean* calls the program *ipcclean* to do the actual reclamation of IPC resources and removal of the lockfile.

The lockfile is removed if the IPC resource reclamation is successful. It is also removed if it is not readable or if it is an out-of-date version. The only time it should not be removed is when resources are not reclaimed due to permissions violations.

The *rmb* installation process should add an invocation of *rmbclean* to the **/etc/rc** script. This is recommended as a means of cleaning up all lockfiles each time HP-UX is booted. *Rmbclean* can also be invoked manually from *rmb* with the **–i** option to *rmb*. *Rmb* also automatically invokes *rmbclean* to attempt to reclaim IPC resources when no more are available.

AUTHOR

Rmbclean was developed by HP

FILES

/usr/lib/rmb/.lock/lock*	rmb lock files
/usr/lib/rmb/ipcclean	a program to scan a lockfile for IPC resources
/usr/lib/rmb/rmbclean	the rmbclean script

INTERNATIONAL SUPPORT

ipcclean: messages

NAME

rmbconfig – generate and view RMB configuration information

SYNOPSIS

rmbconfig [**–bfs**]

Remarks:

This command requires installation of optional BASIC/UX software (not included with the standard HP-UX operating system) before it can be used.

The keywords **disk** and **disc** are parsed as equivalents by *rmb* and related commands and programs.

DESCRIPTION

Rmbconfig is a program to extract system information from HP-UX and store this information in a file accessible to *rmb*. *Rmbconfig* also automatically generates device files for all cards found in the system that are supported by *rmb*.

Since *rmbconfig* creates device files and reads **/dev/kmem**, it usually requires root capabilities. It is recommended that *rmbconfig* be installed with *setuid root* capabilities. This allows *rmb* to call it for updated system information.

The *rmb* installation process should add an invocation of *rmbconfig* to the **/etc/rc** script. This is recommended as a means of updating system configuration information each time HP-UX is booted. *Rmbconfig* can also be invoked manually from *rmb* with the **-k** option to *rmb*. *Rmb* also automatically invokes *rmbconfig* if the file **/usr/lib/rmb/rmbbootinfo** is not found or is the wrong revision.

Options

The command line options are:

–b This option specifies that device files are not to be created.

–f This option forces scanning of the computer back-plane. Normally *rmbconfig* does not scan the backplane if an HP 98577 card is found.

–s Specifies that the extracted system information is to be displayed.

Displayed information

There are several types of information that *rmbconfig* extracts from the system. In order of display they are:

HP-UX Information about HP-UX is determined via the *uname* system call. This includes information on the node name, computer number, and operating system revision.

bootrom The bootrom is scanned through a temporary iomap device file to determine the bootrom revision.

hardware Hardware capabilities are determined from the kernel. These include processor, coprocessors and other assists. Recognized hardware includes:

Device	Description
MC68020	Model 320, 330, and 350 processor
MC68030	Model 332, 340, 360 and 370 processor
HP 98248A	Floating-point accelerator
MC68881	Floating-point coprocessor
MC68882	Floating-point coprocessor
HP 98635	Floating-point card
HP 98286A	DOS coprocessor
HP 98620	DMA card

console The system console address and type are read from **/dev/kmem**. A device file for the console is created (or linked) as **/dev/rmb/crt**.

ram Information about the system RAM. This includes 3 values:

physical amount of physical RAM
available amount available for processes
free currently unused RAM

swap The location and sizes of all swap devices configured into the kernel are determined. This includes those swap devices which have **not** been *swapon*ed. This information is extracted from **/dev/kmem**.

disks The location of all mounted disks is read from **/dev/kmem**.

drivers The kernel is scanned for all recognized drivers. This includes the SYS V IPC code, disk drivers, interface drivers and card drivers. Recognized drivers (including permanent drivers) are:

Driver	Abbr.	Description
amigo	AMIGO	Amigo-protocol disk driver
ciper	CIPER	CIPER-protocol printer driver
console	CONS	Console driver
cs80	CS80	CS/80 disk driver
diskless	DSKLESS	diskless server driver
dos	DOS	HP 98286 MS-DOS card driver
ether	ETHER	Ethernet driver
gpio	GPIO	HP 98622 GPIO driver
graphics	-none-	Graphics driver
hil	HIL	HP-HIL loop driver
hpib	HPIB	HP-IB interface driver
ieee802	802	IEEE-802 (LAN) driver
iomap	IOMAP	Iomap file driver
mem	MEM	Mem/kmem driver
messages	-none-	System V messages
minifloppy	MF	Internal minifloppy (obsolete)
nimitz	-none-	HP 9836-style-keyboard driver
nfs	-none-	Network File System driver
plotter	PLOT	Old plotter driver
printer	PR	Line printer driver
ptym	PTYM	Pty master driver
ptys	PTYS	Pty slave driver
ramdisk	RAMDISC	RAM disk driver
rdu	RDU	RDU driver
r8042	R8042	8042 keyboard driver
rfa	-none-	remote file access driver
rje	RJE	HP 98641 RJE driver
scsi	SCSI	SCSI disk interface driver
semaphores	-none-	System V semaphores
shared memory	-none-	System V shared memory
sna	SNA	SNA link driver
srm	SRM	HP 98629 SRM driver
stp	STP	streaming tape driver
stealth	STEALTH	HP 98577 VME backplane driver
swap	SWAP	disk swap space driver
tp	TP	magnetic tape driver
tty	TTYsy	virtual (/dev/tty) driver
ttyxx	TTYxx	physical tty driver
vme	VME	HP 98646 VME extender driver
98624	-none-	hpib card driver
98625	-none-	disk interface card driver
98626	-none-	RS-232 card driver
98628	-none-	datacomm card river
98642	-none-	RS-232 mux driver

variables Various configurable kernel variables of interest are extracted from **/dev/kmem**. These include:

Variable	Description
maxdsiz	Maximum program data size
maxssiz	Maximum program stack size
maxtsiz	Maximum program text size
maxuprc	Maximum number of user processes
msgtql	System message queue limit
nproc	System process limit
shmmin	Minimum shared memory segment size
shmmax	Maximum shared memory segment size
shmmaxaddr	Highest address for shared memory

interfaces I/O interfaces are determined from two sources in **/dev/kmem**. First the kernel's internal device table is searched to find interfaces supported by the kernel. Then the physical I/O space of the machine is searched for interfaces not recognized by the kernel (unless a *STEALTH* VME card is found). Interfaces recognized by *rmbconfig* and the device files created (if any) include:

Interface	Dev_file	Description
HP 50962	srm	Serial SRM link card
HP 98253	eprom	EPROM programmer card
HP 98259	bubble	Bubble memory card
HP 98265	-none-	SCSI disk interface card
HP 98287	gbox	Hi-res display controller interface
HP 98577	vme	VME backplane adapter
HP 98622	gpio	GPIO card
HP 98623	bcd	BCD interface card
HP 98624	hpib	HP-IB card
HP 98625	-none-	HP-IB disk interface card
HP 98626	serial	RS-232 card
HP 98627	moon	RGB interface card
HP 98628	serial	Datacomm interface card
HP 98629	srm	SRM interface card
HP 98633	double	Multi-programmer interface
HP 98640	adc	ADC card
HP 98641	rje	RJE card
HP 98642	mux	RS-232 mux card
HP 98643	-none-	LAN interface card
HP 98644	serial	low-cost RS-232 card
HP 98646	vme	VME extender card
HP 98649	sna	SDLC card for IBM SNA
HP 98691	pdi	programmable datacomm card
HP 98695	3270.	3270 emulator
proto-sngl	iomap	prototype standard size card
proto-dbl	double	prototype double size card
proto-quad	quad	prototype quad size card

Device files are created as **/dev/rmb/**<*dev_file*><*sc*> where <*dev_file*> is the name specified in the above table, and <*sc*> is the select code at which the card is mapped. Interfaces with -none- indicated for their <*dev_file*> are either not supported for direct access by *rmb*, or do not require device files for access (HP 98248,

HP 98635). If an appropriate device file is found in **/dev**, a link is made to it in **/dev/rmb**. Otherwise a new device file is created. If the interface is supported by HP-UX, the device file is created with permissions 000 for security, and the system administrator is required to change them to allow access to the card.

The display of interfaces includes the card number (name), select code, card interrupt level, appropriate device file name (for *rmb*), and abbreviations for all drivers (from the driver table above) for which a device file was found for this select code. Device files are searched only in directories **/dev** and **/dev/rmb**.

HIL Files **/dev/hil*** are checked to determine the types of devices on the HP-HIL loop. Devices locked by running programs (such as window managers) cannot be determined and are reported as busy.

AUTHOR

Rmbconfig was developed by HP

FILES

/dev	searched for relevant device files
/dev/hil*	device files scanned for HIL info
/dev/kmem	device file used to access kernel info
/dev/rmb/*	location of created/linked device files
/tmp/rmbconf*	temp iomap file for accessing the bootrom
/usr/lib/rmb/rmbbootinfo	rmb configuration information file
/usr/lib/rmb/rmbconfig	the rmbconfig executable

INTERNATIONAL SUPPORT

rmbconfig: messages

NAME

rmbdfile – kernel configuration file scanner for rmb (HP BASIC/UX)

SYNOPSIS

rmbdfile [**-v**] [*dfile*]

Remarks:

This command requires installation of optional BASIC/UX software (not included with the standard HP-UX operating system) before it can be used.

The keywords **disk** and **disc** are parsed as equivalents by *rmb* and related commands and programs.

DESCRIPTION

Rmbdfile is a program to scan a kernel configuration file and modify it such that it will generate a kernel capable of running *rmb*. If *dfile* is specified, it is scanned. Otherwise the input is taken from **stdin**. The modified file is output to **stdout**.

Options

–v Specifies that *rmbdfile* scan a kernel configuration file and indicate whether or not the file is adequate for generating a kernel that can support running *rmb*. The modified file is not output.

Operation

rmbdfile examines two types of kernel parameters:

Drivers *Rmb* requires that the following drivers be configured into the kernel. If not present in the input file they are added by *rmbdfile*. Other drivers are passed through to the output.

Driver Name	Description
98624	HP 98624 HP-IB card driver
gpio	Device I/O Library GPIO driver
hpib	Device I/O Library HP-IB driver
mesg	message queues
sema	semaphores
shmem	shared memory

Variables *Rmb* requires a minimum or a maximum value for certain kernel variables. Some of these variables have an adequate default value; others are added by *rmbdfile* if not explicitly specified. The value of all specified variables are checked against the minimum (or maximum) value and if less (or greater), the variable is assigned this value. The table below indicates the scanned variables, whether the default value is adequate, whether a minimum or maximum is tested, and the boundary value. Variables not in the table are passed through unchanged.

Variable	Default ok	Requirement	Value
maxdsiz	yes	min	1310720 (1.25 Mbytes)
maxssiz	yes	min	524288 (.5 Mbytes)
maxtsiz	yes	min	2097152 (2 Mbytes)
maxuprc	no	min	64
msgtql	no	min	256
ndilbuffers	yes	min	5
nproc	no	min	128
shmmax	yes	min	1048576 (1 Mbytes)
shmmaxaddr	yes	min	4194304 (4 Mbytes)
shmmin	yes	max	65536 (64 Kbytes)

Any other kernel parameters are passed through to the output.

EXIT CODES

0 Normal execution

1 Input file not adequate (for **−v** option)

AUTHOR

Rmbdfile was developed by HP

INTERNATIONAL SUPPORT

rmbdfile: messages

NAME

rmbkernel – kernel building script for rmb (HP BASIC/UX)

SYNOPSIS

rmbkernel [*dfile*]

Remarks:

This command requires installation of optional BASIC/UX software (not included with the standard HP-UX operating system) before it can be used.

The keywords **disk** and **disc** are parsed as equivalents by *rmb* and related commands and programs.

DESCRIPTION

Rmbkernel is a script to generate a new kernel consistent with the requirements of *rmb*. If an input *dfile* is specified, then that file is used as the starting point for generating the new kernel. Otherwise, a dfile is extracted from the current kernel. The dfile that is used to build the new kernel will be saved as **/etc/conf/dfile.rmb**. If an old version of this file exists it is saved as **/etc/conf/dfile.rmb.old**. The new dfile **/etc/conf/dfile.rmb** is also copied back to *dfile* if one is specified.

The kernel that is built by this script is left in **/etc/conf/hp-ux**. Instructions are then given for installing the new kernel and rebooting.

Root capabilities are required to run *rmbkernel*.

DIAGNOSTICS

Rmbkernel prints messages as it runs to indicate what it is doing. If any failures occur in the script an error message is printed and the script aborts.

AUTHOR

Rmbdfile was developed by HP

FILES

/etc/config	kernel configuration program
/etc/conf/dfile.rmb	dfile corresponding to the new kernel
/tmp/dfile*	temporary working copies of the dfiles
/tmp/sysdef	temporary program for extracting kernel information
/usr/lib/rmb/rmbkernel	the rmbkernel script

NAME

rmbkill – kill a process and all its descendents

SYNOPSIS

rmbkill [**–q**] [-signo] *process_number*

Remarks:

This command requires installation of optional BASIC/UX software (not included with the standard HP-UX operating system) before it can be used.

The keywords **disk** and **disc** are parsed as equivalents by *rmb* and related commands and programs.

DESCRIPTION

Rmbkill is a program that kills a process and all its descendents without them having to be in their own process group. This program is called by *rmb* to kill all EXECUTE processes during a RESET.

Rmbkill first reads the process list from the kernel through the device file **/dev/kmem**. It then searches the process list for the process specified by *process_number*. If *process_number* is not found, the program terminates with an error. If found, the process and all its descendents are sent the signal SIGHUP. The process number and status of each process is printed with indentation showing the process tree structure.

Rmbkill exits if process numbers 0, 1 or 2 are specified. The program runs with setuid root capabilities in order to read **/dev/kmem**, but changes identity to that of the real user before attempting to kill any processes so that user permissions are not violated.

Options

–q Specifies quiet mode. The process tree structure with status is not printed.

–signo Indicates that signal signo should be sent instead of SIGHUP. Valid values include the range 1 through 32. Some signals are not implemented on all systems and produce an error message if incorrect.

AUTHOR

Rmbkill was developed by HP

FILES

/dev/kmem	device file for accessing kernel data structures
/usr/lib/rmb/rmbkill	the rmbkill program

INTERNATIONAL SUPPORT

rmbkill: messages

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel –r*SID file* ...

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the SID specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get*(1)) exists for the named SCCS file, the SID specified must *not* appear in any entry of the *p-file*).

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

DIAGNOSTICS

Use *help*(1) for explanations.

FILES

x.file	See *delta*(1).
z.file	See *delta*(1).

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(4).
SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LANG determines the local language equivalent of the affirmative string ("yes"). LANG also determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *rmdel* behaves as if all internationalization variables are set to "C". See *environ*(5).

STANDARDS CONFORMANCE

rmdel: SVID2, XPG2, XPG3

NAME

rmnl – remove extra new-line characters from file

SYNOPSIS

rmnl

DESCRIPTION

Rmnl removes all blank lines from a file (except at beginning of file as explained below), and is useful for removing excess white space from files for display on a CRT terminal. Groups of two or more successive \n (new-line) characters are reduced to a single \n character, effectively eliminating all blank lines in the file *except* that one or more blank lines at the beginning of a file remain as a single blank line. *Rmnl* is used by the *man* command.

To remove redundant blank lines rather than all blank lines, use *ssp*(1).

To remove all blank lines from a file including beginning of file, use *rmnl* piped to *ssp*, or *ssp* piped to *rmnl*.

Note: *Man*(1) and *fixman*(1M) do not use *rmnl* to remove excess blank lines as previously documented. Both commands use *more*(1) with the **–s** option.

SEE ALSO

man(1), ssp(1).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

rtprio − execute process with realtime priority

SYNOPSIS

rtprio *priority command* [*arguments*]
rtprio *priority* −*pid*
rtprio -t *command* [*arguments*]
rtprio -t −*pid*

DESCRIPTION

Rtprio executes *command* with a realtime priority, or changes the realtime priority of currently executing process *pid*. Realtime priorities range from zero (highest) to 127 (lowest). Realtime processes are not subject to priority degradation and are all of greater (scheduling) importance than non−realtime processes. See *rtprio*(2) for more details.

If **-t** is specified instead of a realtime *priority*, *rtprio* executes *command* with a timeshare (non-realtime) priority, or changes the currently executing process *pid* from a possibly realtime priority to a timeshare priority. The former is useful to spawn a timeshare priority command from a realtime priority shell.

If −**t** is not specified, *command* is not scheduled, or *pid*'s realtime priority is not changed, if the user is not a member of a group having PRIV_RTPRIO access and is not the super−user. When changing the realtime priority of a currently executing process, the effective user ID of the calling process must be super-user, or the real or effective user ID must match the real or saved user ID of the process to be modified.

RETURN VALUE

Rtprio returns exit status 0 if *command* is successfully scheduled or if *pid*'s realtime priority is successfully changed, 1 if *command* is not executable or *pid* does not exist, and 2 if *command* (*pid*) lacks realtime capability, or the invoker's effective user ID is not super-user, or the real or effective user ID does not match the real or saved user ID of the process being changed.

EXAMPLES

The following example executes the *a.out* file at a real-time priority of 100:

```
rtprio 100 a.out
```

The following example sets the currently running process with pid 24217 to a real-time priority of 40:

```
rtprio 40 -24217
```

AUTHOR

Rtprio was developed by HP.

SEE ALSO

setprivgrp(1M), getprivgrp(2), rtprio(2).

NAME

sact – print current SCCS file editing activity

SYNOPSIS

sact *file* ...

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **–e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (i.e., executed a *get* for editing).
Field 4	contains the date that **get –e** was executed.
Field 5	contains the time that **get –e** was executed.

DIAGNOSTICS

Use *help*(1) for explanations.

SEE ALSO

delta(1), get(1), unget(1).

SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

STANDARDS CONFORMANCE

sact: SVID2, XPG2, XPG3

NAME

sar – system activity reporter

SYNOPSIS

sar [–**ubdycwaqvmA**] [–**o** file] t [n]

sar [–**ubdycwaqvmA**] [–**s** time] [–**e** time] [–**i** sec] [–**f** file]

DESCRIPTION

In the first form above, the *sar* command samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the –**o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second form, with no sampling interval specified, *sar* extracts data from a previously recorded *file*, either the one specified by –**f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the –**s** and –**e** *time* arguments of the form *hh*[:*mm*[:*ss*]] The –**i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

–**u** Report CPU utilization (the default):
%usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

–**b** Report buffer activity:
bread/s, bwrit/s – transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s – accesses of system buffers;
%rcache, %wcache – cache hit ratios, e.g., 1 – bread/lread;
pread/s, pwrit/s – transfers via raw (physical) device mechanism.

–**d** Report activity for each block device, e.g., disk or tape drive:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or to device, number of bytes transferred in 512-byte units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

–**y** Report TTY device activity:
rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.

–**c** Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls.

–**w** Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins (including initial loading of some programs) and swapouts;
pswch/s – process switches.

–**a** Report use of file access system routines:
iget/s, namei/s, dirblk/s.

−q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc − run queue of processes in memory and runnable;
swpq-sz, %swpocc − swap queue of processes swapped out but ready to run.

−v Report status of text, process, inode and file tables:
text-sz, proc-sz, inod-sz, file-sz − entries/size for each table, evaluated once at sampling point;
text-ov, proc-ov, inod-ov, file-ov − overflows occurring between sampling points.

−m Report message and semaphore activities:
msg/s, sema/s − primitives per second.

−A Report all data. Equivalent to **−udqbwcayvm**.

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar −o temp 60 10
```

To later review disk and tape activity from that period:

```
sar −d −f temp
```

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sa1(1M).

STANDARDS CONFORMANCE

sar: SVID2

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

sccsdiff –r*SID1* –r*SID2* [–**p**] [–**s**n] *file* ...

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

–r*SID?* *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given. The SID's accepted, and the corresponding version retrieved for the comparison are the same as for *get*(1).

–**p** pipe output for each file through *pr*(1).

–**s***n* *n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

DIAGNOSTICS

"*file*: No differences" if the two versions are the same.

Use *help*(1) for explanations.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), diff(1), get(1), help(1), pr(1).
SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

NAME

script − make typescript of terminal session

SYNOPSIS

script [−**a**] [*file*]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. It starts a shell named by the SHELL environment variable, or by default **/bin/sh**, and silently records a copy of output to your terminal from that shell or its descendents, using a pseudo-terminal device (see *pty*(7)).

All output is written to *file*, or appended to *file* if the −**a** option is given. If no file name is given, the output is saved in a file named **typescript**. The recording can be sent to a line printer later with *lp*(1), or reviewed safely with the −**v** option of *cat*(1).

The recording ends when the forked shell exits (or the user ends the session by typing "exit") or the shell and all its descendents close the pseudo-terminal device.

This program is useful when operating a CRT display and a hard-copy record of the dialog is desired. It can also be used for a simple form of session auditing.

Script respects the convention for login shells, as described in *su*(1), *sh*(1), and *ksh*(1). Thus, if it is invoked with a command name beginning with a hyphen (−) (that is, −**script**), *script* passes to the shell a basename that is also preceded by a hyphen.

EXAMPLES

The command:

script scott

saves everything printed thereafter on the user's screen into the file **scott**.

The command:

script -a temp

appends a copy of everything printed to the user's screen to the file **temp**.

WARNINGS

A command such as **cat scott**, which displays the contents of the destination file, should not be issued while executing *script* because it would cause *script* to log the output of the *cat*(1) command to itself until all available disk space is filled. Other commands, such as *more*(1), can cause the same problem, to a lesser extent.

Script records all received output in the *file*, including typing errors, backspaces, and cursor motions. Note that it does not record typed characters, only echoed characters. Thus passwords are not recorded in the *file*. Responses other than simple echoes (such as output from screen oriented editors and *ksh* command editing) are recorded as they appeared in the original session.

AUTHOR

Script was developed by the University of California, Berkeley and HP.

NAME

sdfchmod – change mode of an SDF file

SYNOPSIS

sdfchmod mode device:file ...

DESCRIPTION

Sdfchmod is intended to mimic *chmod*(1).

An SDF file name is recognized by the embedded colon (:) delimiter (see *sdf*(4) for SDF file naming conventions).

The permissions of each named file are changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see *chmod*(2)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others.

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, which is the default if *who* is omitted.

Op can be + to add *permission* to the file's mode, – to take away *permission*, or = to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save text – sticky); **u**, **g** or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g**; **t** only works with **u**.

EXAMPLES

The examples that follow assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c1d0s3**.

The first example denies write permission to others for the SDF directory **/bin**:

sdfchmod o–w /dev/rdsk/c1d0s3:/bin

The second example makes the SDF file **/users/fred/a.out** executable and readable by everyone:

sdfchmod a=rx /dev/rdsk/c1d0s3:/users/fred/a.out

The third example adds read permission for the group associated with the SDF file **/last.boot.rev**:

sdfchmod g+r /dev/rdsk/c1d0s3:/last.boot.rev

The fourth example assigns read and execute permission to everybody, and sets the set-user-id bit for the SDF file **/usr/local/hoo**:

sdfchmod 4555 /dev/rdsk/c1d0s3:/usr/local/hoo

In the fifth example, the two commands perform the same function, namely to give read, write, and execute permission to the owner and read and execute permissions to everybody else for the SDF file **/users/debbie/script**:

sdfchmod a=rx,u+w /dev/rdsk/c1d0s3:/users/debbie/script

sdfchmod 755 /dev/rdsk/c1d0s3:/users/debbie/script

AUTHOR

Sdfchmod was developed by the Hewlett-Packard Company.

SEE ALSO

sdf(4), chmod(1), chmod(2).

NAME

sdfchown, sdfchgrp – change owner or group of an SDF file

SYNOPSIS

sdfchown owner device:file ...

sdfchgrp group device:file ...

DESCRIPTION

Sdfchown and *sdfchgrp* are intended to mimic *chown*(1) and *chgrp*(1).

An SDF file name is recognized by the embedded colon (:) delimiter (see *sdf*(4) for SDF file naming conventions).

Sdfchown changes the owner of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

Sdfchgrp changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

EXAMPLES

The examples that follow assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c9d1d5**.

The first example sets the owner of the SDF file **/users/abc/phone.num** to **adm**:

sdfchown adm /dev/rdsk/c9d1d5:/users/abc/phone.num

The second example sets the group ID of the SDF file **/tmp/b.date** to the decimal number **105**:

sdfchgrp 105 /dev/rdsk/c9d1d5:/tmp/b.date

AUTHOR

Sdfchown was developed by the Hewlett-Packard Company.

FILES

/etc/passwd
/etc/group

SEE ALSO

chown(1), chgrp(1), group(4), passwd(4), sdf(4).

NAME

sdfcp, sdfln, sdfmv – copy, link, or move files to/from an SDF volume

SYNOPSIS

sdfcp file1 [file2 ...] target
sdfln file1 [file2 ...] target
sdfmv file1 [file2 ...] target

DESCRIPTION

Sdfcp, *sdfln*, *sdfmv* are intended to mimic *cp*(1).

An SDF file name is recognized by the embedded colon (:) delimiter (see *sdf*(4) for SDF file naming conventions).

Sdfcp copies an HP-UX file to an SDF file, or an SDF file to either an SDF or HP-UX file. It also copies a list of HP-UX files to an SDF directory, or copies a list of SDF files to either an SDF or HP-UX directory.

Sdfln creates links to *target* if, and only if, all *files* referenced on the command line are on the same SDF volume.

Sdfmv behaves the same way as *sdfcp*, except that it moves files instead of copying them.

The last name on the argument list is the target file or directory. If two or more files are specified in the command line, not counting *target*, then *target* must be a directory. Under no circumstances may any argument other than *target* be a directory.

The file name "-" (dash) is interpreted to mean standard input or standard output, depending on the position in the argument list. The use of the file name "-" makes no sense for *sdfln* and *sdfmv*.

EXAMPLES

The examples that follow assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c2d0s2**.

The first example copies the HP-UX file **mydata** to the SDF file **/users/old/mike/olddata**:

```
sdfcp mydata /dev/rdsk/c2d0s2:/users/old/mike/olddata
```

The second example copies the SDF file **/users/gary/.cshrc** to the SDF directory **/tmp** (on the same SDF volume):

```
sdfcp /dev/rdsk/c2d0s2:/users/gary/.cshrc /dev/rdsk/c2d0s2:/tmp
```

The third example copies the SDF files **/a/b** and **/a/c** to the HP-UX directory **/users/dave**:

```
sdfcp /dev/rdsk/c2d0s2:/a/b /dev/rdsk/c2d0s2:/a/c /users/dave
```

The fourth example copies standard input to the SDF file **/users/craig/memo**:

```
sdfcp - /dev/rdsk/c2d0s2:/users/craig/memo
```

The fifth example copies the SDF file **/etc/rc** to the SDF file **/etc/rc.old** on another SDF volume residing in the HP-UX device file **/dev/rdsk/c2d1s0**:

```
sdfcp /dev/rdsk/c2d0s2:/etc/rc /dev/rdsk/c2d1s0:/etc/rc.old
```

The sixth example shows how you can implement a *cat*(1) program for concatenating SDF files using *sdfcp* in a shell script:

```
if [ $# -lt 1 ]
then
        echo "Usage: sdfcat file ..."
        exit 1
fi
```

```
for i in $*
do
        sdfcp $i -
done
```

The seventh example links the SDF file **/tmp/x** to **/users/gary/x1**:

sdfln /dev/rdsk/c2d0s2:/tmp/x /dev/rdsk/c2d0s2:/users/gary/x1

The eighth example moves the HP-UX file **/etc/rc.backup** to the SDF file **/etc/rc**:

sdfmv /etc/rc.backup /dev/rdsk/c2d0s2:/etc/rc

Assuming that the current HP-UX directory contains only regular files, the ninth example shows how to move all files in an HP-UX directory to the SDF directory **/savestuff**:

sdfmv * /dev/rdsk/c2d0s2:/savestuff

AUTHOR

Sdfcp was developed by the Hewlett-Packard Company.

SEE ALSO

sdf(4), cp(1).

NAME

sdffind – find files in an SDF system

SYNOPSIS

sdffind path-name-list expression

DESCRIPTION

Sdffind is intended to mimic *find*(1).

An SDF file name is recognized by the embedded colon (:) delimiter (see *sdf*(4) for SDF file naming conventions).

Sdffind recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below.

–name *pattern* True if *pattern* matches the current file name.

–perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

(flags&onum)==onum

–type *c* True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file.

–type *n* True if the current file being examined by *sdffind* is a network special file.

–links *n* True if the file has *n* links.

–user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file (on the local system, not the SDF file system), it is taken as a user ID.

–group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file (on the local system, not the SDF file system), it is taken as a group ID.

–size *n* True if the file is *n* blocks long.

–exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.

–ok *cmd* Like **–exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

–print Always true; causes the current path name to be printed. This option must be included on the *sdffind* command line anytime you want *sdffind* to print the path names it has found on the standard output. If **–print** is not specified, *sdffind* locates the files, but fails to tell you about them!

When **–print** is specified as the only *expression*, *sdffind* prints the absolute path names of all files it finds, beginning at each directory in the *path-name-list*. If **–print** is included as the last component of an *expression*, *sdffind* prints the absolute path names of only those files which satisfy the other primaries in the *expression*.

–inum *n* True if the file has inode number *n*.

EXAMPLES

The examples that follow assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c3d0s0**.

The first example prints the names of all files on the SDF volume **/dev/rdsk/c3d0s0**:

sdffind /dev/rdsk/c3d0s0: -print

The second example prints the name of all the subdirectories under **/usr/lib** on the SDF file system:

sdffind /dev/rdsk/c3d0s0:/usr/lib −type d −print

The third example gives a long listing of every ordinary file under /users on the SDF file system:

sdffind /dev/rdsk/c3d0s0:/users −type f −exec sdfls -l {} **';'**

The fourth example finds all the files on the SDF volume by the name of "core" and asks whether they should be removed:

sdffind /dev/rdsk/c3d0s0: -name core -ok sdfrm {} **';'**

AUTHOR

Sdffind was developed by the Hewlett-Packard Company.

FILES

/etc/passwd
/etc/group

SEE ALSO

sdf(4), find(1), stat(2), chmod(1).

NAME

sdfls, sdfll − list contents of SDF directories

SYNOPSIS

sdfls [−**AadlpFi**] [names]
sdfll [**sdfls options**] [names]

DESCRIPTION

Sdfls is intended to mimic *ls*(1). *Sdfll* is equivalent to **sdfls -l**.

An SDF file name is recognized by the embedded colon (:) delimiter (see *sdf*(4) for SDF file naming conventions).

For each SDF directory named, *sdfls* lists the contents of that SDF directory; for each SDF file named, *sdfls* repeats its name and the information requested.

If you are the super-user, *sdfls* defaults to listing all files except . (current directory) and .. (parent directory).

There are several options to *sdfls*:

−a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.

−A The same as **−a**, except that the current directory "." and parent directory ".." are not listed. For the super-user, this flag defaults to ON, and is turned off by **−A**. Due to the internal data representation of the SDF directory format, the **−A** and **−a** options perform the same function.

−d If argument is a directory, list only its name; often used with **−l** to get the status of a directory.

−l List in long format giving mode, number of links, owner, group, size in bytes, and time of last modification for each file.

−p Do not use **/etc/passwd** and **/etc/group** to interpret user and group ownership, but rather print out the numeric form.

−F If the entry is a directory or SRM special file, print a '/' character after the entry, or if the entry is executable, print a '*' character after the entry.

−i Print the inode number of each entry before the listing the entry names.

EXAMPLES

The examples that follow assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c7s0s1**.

The first example will list all the files in the root directory of the SDF directory structure:

sdfls −a /dev/rdsk/c7s0s1:

The second example gives (in long format) all the information about the SDF directory **/users/root** itself (but not the files in the directory):

sdfls −ld /dev/rdsk/c7s0s1:/users/root

The third example will print (in long form) all the information about every file in the SDF directory **/etc**, printing numbers instead of names for user and group IDs.

sdfls −ap /dev/rdsk/c7s0s1:/etc

The previous example is useful if the SDF directory structure was not created on your system but brought in from another series 500 system.

DEPENDENCIES

On the Series 500, network special files are supported. With the **−F** option, *sdfls* will print a '/'

character after the entry for a network special file.

AUTHOR

Sdfls was developed by the Hewlett-Packard Company.

FILES

/etc/passwd	to get user ids.
/etc/group	to get group ids.

SEE ALSO

sdf(4), ls(1).

NAME

sdfmkdir – make an SDF directory

SYNOPSIS

sdfmkdir device:dirname ...

DESCRIPTION

Sdfmkdir is intended to mimic *mkdir*(1).

An SDF file name is recognized by the embedded colon (:) delimiter (see *sdf*(4) for SDF file naming conventions).

Sdfmkdir creates specified directories in mode 777, masked with the current value of *umask*.

RETURNS

Sdfmkdir returns exit code **0** if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

EXAMPLES

The following example assumes that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c0d1s5**.

This example will create an empty subdirectory named **sysmods** under the directory **/usr/lib**:

sdfmkdir /dev/rdsk/c0d1s5:/usr/lib/sysmods

AUTHOR

Sdfmkdir was developed by the Hewlett-Packard Company.

SEE ALSO

sdf(4), mkdir(1).

NAME

sdfrm, sdfrmdir – remove SDF files or directories

SYNOPSIS

sdfrm [**–fri**] device:file ...

sdfrmdir device:dir ...

DESCRIPTION

Sdfrm and *sdfrmdir* are intended to mimic *rm*(1) and *rmdir*(1).

An SDF file name is recognized by the embedded colon (:) delimiter (see *sdf*(4) for SDF file naming conventions).

Sdfrm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed.

If a designated file is a directory, an error comment is printed (unless the optional argument **–r** has been used, see below).

The options are:

–f Remove a file with no questions asked, even if the file has no write permission.

–r Cause *sdfrm* to recursively delete the entire contents of a directory, and then the directory itself. *Sdfrm* can recursively delete up to 17 levels of directories.

–i Cause *sdfrm* to ask whether or not to delete each file. If **–r** is also specified, *sdfrm* asks whether to examine each directory encountered.

Sdfrmdir removes entries for the named directories, which must be empty.

EXAMPLES

The following examples assume that an SDF directory structure exists on the HP-UX device file **/dev/rdsk/c6d0s1**.

The first example recursively combs through the SDF directory **/tmp** and asks if each SDF file should be removed (forced, with no file mode checks):

sdfrm –irf /dev/rdsk/c6d0s1:/tmp

The second example removes the SDF directory **/users/doug**:

sdfrmdir /dev/rdsk/c6d0s1:/users/doug

AUTHOR

Sdfrm was developed by the Hewlett-Packard Company.

SEE ALSO

sdf(4), rm(1), rmdir(1).

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

Sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

The following options exist:

–w *n* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

–l Only print the left side of any lines that are identical.

–s Do not print identical lines.

–o *output* Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed – stream text editor

SYNOPSIS

sed [**−f** *sfile*] [**−e** *script*] [**−n**] [*file ...*]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **−f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **−e** option and no **−f** options, the flag **−e** may be omitted. The **−n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **−n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, that is, a */regular expression/* in the style of *ed*(1) modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line embedded in the pattern space.

A period **.** matches any character except the terminal new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Sed supports the Basic Regular Expression syntax (see *regexp*(5)).

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**
text Append. Place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the **:** command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) **c**\
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**\
text Insert. Place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output if the default output has not been suppressed (by the **−n** option on the command line or the **#n** command in the *script* file). Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) **p** Print. Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of **/**. For a fuller description see *ed*(1). *Flags* is zero or more of:

n n= 1 - 512. Substitute for just the n th occurrence of the *regular expression.*

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made and the default output has been suppressed (by the **−n** option on the command line or the **#n** command in the *script* file).

w *wfile*
Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label*
Test. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfile*
Write. Append the pattern space to *wfile*.

(2) **x**
Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)**!** *function*
Don't. Apply the *function* (or group, if *function* is {} only to lines *not* selected by the address(es).

(0) **:** *label*
This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1)=
Place the current line number on the standard output as a line.
(2){
Execute the following commands through a matching } only when the pattern space is selected. The syntax is:

```
{ cmd1
cmd2
cmd3
.
.
.
}
```

(0)
An empty command is ignored.
(0)#
If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and the characters matched by character class expressions in regular expressions.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *sed* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

SEE ALSO

awk(1), ed(1), grep(1), environ(5), lang(5), regexp(5).

The *sed: A Non-Interactive Streaming Editor* tutorial in the *Text Editors and Processors* volume of *HP-UX Concepts and Tutorials*.

WARNINGS

There is a limit of 100 commands in the script.

STANDARDS CONFORMANCE

sed: SVID2, XPG2, XPG3

NAME

sh, rsh − shell, the standard/restricted command programming language

SYNOPSIS

sh [−**acefhiknrstuvx**] [*args*]
rsh [−**acefhiknrstuvx**] [*args*]

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See **Invocation** below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, **#**, **?**, −, **$**, and **!**.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(5) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by | (or, for historical compatibility, by ^). The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or ||, and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and ||. The symbols **&&** and || also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**
Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see **Parameter Substitution** below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ... **)** *list* **;;**] ... **esac**
A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the Pattern Matching Notation as qualified for the **case** command (see *regexp*(5)).

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if**

command returns a zero exit status.

while *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a sub-shell.

{ *list*;} *list* is simply executed.

name () { *list*;} Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see **Execution**).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments
A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

Command Substitution
The standard output from a command enclosed in a pair of grave accents (` `) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution
The character **$** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${***parameter***}** The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or all the positional parameters, starting with **$1, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter***:-***word***}**
If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter***:=***word***}**
If *parameter* is not set or is null set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, then the message "parameter null or not set" is printed.

${*parameter***:+***word***}**
If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
-	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME	The default argument (home directory) for the *cd* command.
PATH	The search path for commands (see **Execution** below). The user may not change **PATH** if executing under *rsh*.
CDPATH	The search path for the *cd* command.
MAIL	If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.
MAILCHECK	This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.
MAILPATH	A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by **%** and a message that will be printed when the modification time changes. The default message is *you have mail*.
PS1	Primary prompt string, by default "**$** ".
PS2	Secondary prompt string, by default "> ".
IFS	Internal field separators, normally **space**, **tab**, and **new-line**.
SHACCT	If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1M) and *acctcms*(1M) can be used to analyze the data collected.
SHELL	When the shell is invoked, it scans the environment (see **Environment** below) for this name. If it is found and there is an 'r' in the file name part of its value, the shell becomes a restricted shell. **SHELL** is also used by some processors to determine which command interpreter to run.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (**""** or **''**) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is processed as a pattern for file name expansion. The form of the patterns is the Pattern Matching Notation defined by *regexp*(5).

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (' '), except a single quote, are quoted. Inside double quote marks (" "), parameter and command substitution occurs and \ quotes the characters \, `, ", and **$**. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2" ....**

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of **PS2**) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<word	Use file *word* as standard input (file descriptor 0).
>word	Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.
>>word	Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<<[-]word	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\new-line** is ignored, and \ must be used to quote the characters \, **$**, `, and the first character of *word*. If **-** is appended to **<<**, then all leading tabs are stripped from *word* and from the document.
<&digit	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&digit**. (See *dup*(2)).
<&-	The standard input is closed. Similarly for the standard output using **>&-**.

If any of the above is preceded by a digit, then the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1. Note that this type of I/O redirection is necessary if you want to *synchronously* collect stdout and stderr output in the same file. Redirecting stdout and stderr separately will cause asynchronous collection of data at the destination (i.e. things written to stdout can subsequently be over-written by things written to stderr, and vice-versa).

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by **&** then the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see *environ*(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                    and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **−k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set −k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the **Special Commands** listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a **/** then the search path is not used; such commands will not be executed by the restricted shell.

Otherwise, each directory in the path is searched for an executable file. If the file has execute permissions but is not a directory or an executable object code file, it is assumed to be a script file, which is a file of data for an interpreter. If the first two characters of the script file are "#!", *exec*(2) expects an interpreter path name to follow. *Exec*(2) then attempts to execute the specified interpreter as a separate process to read the entire script file. If a call to *exec*(2) fails, **/bin/sh** is spawned to interpret the script file.

A parenthesized command is also executed in a sub-shell. The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

The following commands are executed in the shell process. Input/output redirection is permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*. Note that this command does not spawn another shell to execute *file*, and thus differs in behavior and output from executing *file* as a shell script.

break [*n*] Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.

continue [*n*] Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*] Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <**null**> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

echo [*arg* ...] Echo arguments. See *echo*(1) for usage and description.

eval [*arg* ...] The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...] The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*] Causes a shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, then a list of all names that are exported in this shell is printed. Function names may *not* be exported.

hash [**−r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the

work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. A **−H option to** *pwd* is supported. It makes hidden directories (see *cdf*(4)) in the path visible. See *pwd*(1) for usage and description.

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, then a list of all *readonly* names is printed.

return [*n*] Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**−−aefhkntuvx** [*arg* ...]]

- **−a** Mark variables which are modified or created for export.
- **−e** Exit immediately if a command exits with a non-zero exit status.
- **−f** Disable file name generation
- **−h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- **−k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **−n** Read commands but do not execute them.
- **−t** Exit after reading and executing one command.
- **−u** Treat unset variables as an error when substituting.
- **−v** Print shell input lines as they are read.
- **−x** Print commands and their arguments as they are executed.
- **−−** Do not change any of the flags; useful in setting **$1** to −.

Using + rather than − causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given then the values of all names are printed.

shift [*n*] The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test*(1) for usage and description. Note that **"[** ... **]"** in an **if** *list* is interpreted the same as **"test** ...". There must be blanks around the brackets.

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
The command *arg* is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) or

signal 18 (death of child) will produce an error. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *n* is 0 then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [**−f** [*n*]]
If the **−f** *n* option is used, a size limit of *n* blocks is imposed on files written by child processes (files of any size may be read). With no argument, the current limit is printed. If no option is given, **−f** is assumed.

umask [*nnn*] The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH, PS1, PS2, MAILCHECK** and **IFS** cannot be unset.

wait [*n*] Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is −, commands are initially read from **/etc/profile** and then from **$HOME/.profile**, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as **/bin/sh**. The flags below are interpreted by the shell on invocation only; Note that unless the **−c** or **−s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c *string* If the **−c** flag is present then commands are read from *string*.

−s If the **−s** flag is present or if no arguments remain then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

−i If the **−i** flag is present or if the shell input and output are attached to a terminal, then this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

−r If the **−r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **$PATH**,
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes

that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating pattern matching notation for file name generation.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as letters, and the characters matched by character class expressions in pattern matching notation.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *sh* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

RETURN VALUE

The error codes returned by the shell are:

0 – success;
1 – a built-in command failure (see **Special Commands**);
2 – syntax error;
3 – signal received that is not trapped.

If the shell is non-interactive, it will terminate and pass one of the above as its exit status. If it is interactive, it will not terminate, but $? will be set to one of the above values.

Whenever a child process of the shell dies due to a signal, the shell returns an exit status of 80 hexadecimal + the number of the signal.

WARNINGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

When the shell encounters >>, it does not open the file in append mode. Instead, it opens the file for writing and seeks to the end.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

The command **readonly** (without arguments) produces the same output as the command **export**.

Failure (non-zero exit status) of a special command preceding a || symbol prevents the *list* following || from executing.

In an international environment, character ordering is determined by the setting of LC_COLLATE, rather than by the binary ordering of character values in the machine collating sequence. This brings with it certain attendant dangers, particulary when using range

expressions in file name generation patterns. For example, the command,

rm [a-z]*

might be expected to match all file names beginning with a lower-case alphabetic character. However, if dictionary ordering is specified by LC_COLLATE, it would also match file names beginning with an upper-case character (as well as those beginning with accented letters). Conversely, it would fail to match letters collated after 'z' in languages such as Norwegian.

The correct (and safe) way to match specific character classes in an international environment is to use a pattern of the form:

rm [[:lower:]]*

This uses LC_CTYPE to determine character classes and will work predictably for all supported languages and codesets. For shell scripts produced on non-internationalized systems (or without consideration for the above dangers), it is recommended that they be executed in a non-NLS environment. This requires that LANG, LC_COLLATE, etc., be set to "C" or not set at all.

DEPENDENCIES

Series 800

The signal SIGSEGV should not be blocked when executing *sh*.

AUTHOR

Sh was developed by AT&T and HP.

FILES

$HOME/.profile
/dev/null
/etc/profile
/tmp/sh*

SEE ALSO

cd(1), echo(1), env(1), login(1), newgrp(1), pwd(1), test(1), umask(1), acctcms(1M), acctcom(1M), dup(2), exec(2), fork(2), pipe(2), ulimit(2), umask(2), wait(2), a.out(4), cdf(4), profile(4), environ(5), lang(5), regexp(5), signal(5).

The Bourne Shell, tutorial in *HP-UX Concepts and Tutorials: Shells and Miscellaneous Tools.*

STANDARDS CONFORMANCE

sh: SVID2, XPG2, XPG3

rsh: SVID2

NAME

shar – make a shell archive package

SYNOPSIS

shar [*options*] [*file* | *dir*] ... > *package*

DESCRIPTION

Shar bundles the named files into a single distribution package suitable for mailing or moving. The files can contain any data, including executables. The resulting package, written to standard output, is a shell script file that can be edited.

The package is unpacked by running *sh*(1) with the package name as an argument. For example, the user would type **sh** *package* on the command line to unpack *package*. The files in *package* are written to the path names recorded in the archive.

If a directory is specified and the **–d** option is not given, all files beneath that directory are archived.

If a special file is specified, the appropriate *mknod*(1) commands are emitted to recreate the file.

Shar protects the contained files from mail processing by inserting an **@** (the "at" symbol) at the beginning of each line, if necessary. If the file contains unusual data, the data are transformed into *uuencode* format. The *uuencode* data are also unpacked with **sh**.

The access modes are preserved for both directories and files.

Options

–a Assume that files can be shipped, regardless of their contents; do not protect them specially. *Shar* is conservative, and might decide to *uuencode* a file containing special characters (for example, control-G) that the user knows do not need protection.

–A Suppress warning messages regarding optional access control list entries. *Shar*(1) does not archive optional access control list entries in a file's access control list (see *acl*(5)). Normally, a warning message is printed for each file having optional access control list entries.

–b Archive files under their base names, regardless of the original path names specified. The contents are thus unpacked into the current directory instead of to the originally specified path names. This allows you to archive files from many directories but unpack them into a single directory. It also allows you to unpack **/etc/termcap** into **./termcap** instead of overwriting the original one in **/etc**.

–c Append to the package a simple data-integrity check using *wc*(1) to ensure that the contents were not damaged in transit. This check is performed automatically after unpacking.

–C Insert a line of the form "--- cut here ---" before the archive.

–d If a directory is specified, do not transmit its contents, but rather only create the empty directory.

–D*dir* Cause the archive to contain code that notifies the user if his or her current directory is not the same as *dir*, which must be an absolute path. If the user is not in *dir*, the unpacking can be continued by responding **yes** to the archive's question.

–e Cause the archive to contain code that prevents *shar* from unpacking files that would overwrite existing files.

−f file Read a list of file names from *file* and archive those files as though they were given as arguments.

−m Retain modification and access times on files when they are unpacked.

−o Preserve user and group ownership on files and directories.

−r Cause the archive to contain code that insists that the user unpacking it be **root**. This is useful for processing system archives.

−s Perform error checking using *sum*(1). Both **−c** and **−s** may be specified for better error checking.

−t Write diagnostics and messages directly to your terminal, instead of to the standard error. This is useful when invoking *shar* from programs such as *vi*(1), which normally combine standard error with standard output. Specifying **−t** also invokes the **−v** (verbose) option.

−u Assume that the remote site has *uudecode* for unpacking. If this option is not specified, a version of *uudecode* is sent and compiled if any non-ASCII files are archived.

−v Announce archived file names as they are packed. The **−t** option determines the destination for these announcements.

−Z Compress files using *compress*(1).

Most options are flagged in the header of the resulting package, thereby recording the format of the archive. The name of the archiver, system, and time/date of the archive are also recorded in the header.

EXAMPLES

To archive all files under your home directory, type:

cd; shar -cmos .

or

shar -cmos $HOME

To preserve your **/dev** directory, type:

shar -mor /dev >save_dev_files

To send your newest programs in directory **newstuff** in your home directory to a friend, type:

cd; shar -cmos newstuff | mailx -s 'new source' friend

DIAGNOSTICS

If the **−b** option is specified, *shar* refuses to archive directories.

Non-zero exit status is returned when problems with arguments occur.

WARNINGS

The modification and access time restoration does not take time zones into account.

Files with newlines in their names scramble the table of contents.

Non-ASCII files with white space in their names do not unpack.

DEPENDENCIES

Series 800

The **−a**, **−e**, **−f**, **−A**, and **−Z** options are not currently supported.

AUTHOR

Shar was invented in the public domain. This version of *shar* was developed by HP.

FILES

/dev/tty
/tmp/unpack$$* (for unpacking non-ASCII files)

SEE ALSO

ar(1), compress(1), cpio(1), find(1), tar(1), acl(5).

NAME

shl – shell layer manager

SYNOPSIS

shl

DESCRIPTION

Shl allows a user to interact with more than one shell from a single terminal by using shell layers. A layer is a shell that is bound to a virtual device. The virtual device can be manipulated like an actual terminal by using *stty*(1) and *ioctl*(2). Each layer has its own process group ID. The user controls these layers by using the commands described below.

The current layer is the layer that can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To block the output of a layer when it is not current, the *stty*(1) option **loblk** can be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *Shl* has its own prompt, >>>, to distinguish it from a layer.

Definitions

A *name* is a sequence of characters delimited by a blank, tab, or new-line. Only the first eight characters are significant. When provided as an argument to the **create**, **login**, or **name** commands, *name* cannot be of the form *n* or *(n)*, where *n* is a decimal number.

Commands

The following commands can be issued from the *shl* prompt level. Any unique prefix is accepted.

create[[**-** [*name*] | *name* [*command*]]]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer is created with a name of the form *(n)*, where *n* is the number of the next available slot in an internal table. Future references to this layer can be made with or without the parenthesis. If *name* is followed by a command, that command is executed in the layer instead of a shell. If **-** is the first argument, a "login shell" is created in the layer. The shell prompt variable **PS1** is set to the name of the layer followed by a space.

login[[*name*]]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer is created with a name of the form *(n)*, where *n* is the number of the next available slot in an internal table. Future references to this layer can be made with or without the parenthesis. *Name* is a "login" type of layer in which the user receives a prompt for user name and password.

name [*oldname*] *newname*
: Rename the layer *oldname*, calling it *newname*. If *oldname* is not specified, the current layer name is changed.

! [*command*]
: Invoke a sub-shell and execute *command*. If no *command* is given, a shell is executed according to the **SHELL** environment variable.

block *name* [*name* ...]
: For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty*(1) option **loblk** within the layer.

delete *name* [*name* ...]
: For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(5)).

help (or **?**) Print the syntax of the *shl* commands.

layers [**−l**] [*name* ...]
For each *name*, list the layer name and its process group. The **−l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
Change the status of the layer referred to by *name* to that of current layer. If no argument is given, the last existing current layer is changed.

toggle Change the status of the previous current layer to that of current layer.

unblock *name* [*name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty*(1) option **−loblk** within the layer.

quit Exit *shl*. All layers are sent the SIGHUP signal.

name Change the status of the layer referred to by *name* to that of current layer. Any unique prefix is accepted.

WARNINGS

The behavior of the *shl* commands **block** and **unblock** is not guaranteed when the SHELL environment variable is set to **/bin/csh** (for *csh*(1)) or **/bin/ksh** (for *ksh*(1)), or when the shell saves and restores the tty state (defined in *termio*(7)) before and after each command is invoked interactively from that shell. For both **/bin/csh** and **/bin/ksh**, the **loblk** or **−loblk** options of *stty*(1) can be used from within the layer to block or unblock the output of that layer.

DEPENDENCIES

Series 800

The **login** command is not currently supported.

FILES

$SHELL Variable containing path name of the shell to use (default is **/bin/sh**).

SEE ALSO

sh(1), stty(1), ioctl(2), signal(5).

STANDARDS CONFORMANCE

shl: SVID2, XPG2

NAME

showcdf – show the actual path name matched for a CDF

SYNOPSIS

showcdf [**–cs**] *name* ...

DESCRIPTION

Showcdf accepts from the standard input a list of names and displays on the standard output the actual path name matched for each name. If any *name* is a context-dependent file (CDF), the actual file matched for that name is displayed. If any *name* is not a CDF, the name is simply displayed.

Options

–c Suppress the printing of non-CDF files. Only files whose path contains a CDF will be displayed.

–s Suppress all output. This option is useful when testing for the exit value from *showcdf*.

RETURN VALUE

The exit values are:

0 if at least one CDF is specified.
1 if no CDFs are specified.
2 if an error occurs (for example, a file is inaccessible).

EXAMPLES

The following example shows the output of *showcdf* when applied to the files **/etc/inittab** and **/usr/adm/sulog**, which are CDFs and **/etc/passwd**, which is not a CDF:

```
$ showcdf /etc/inittab /usr/adm/sulog /etc/passwd
/etc/inittab+/donald
/usr/adm+/donald/sulog
/etc/passwd
$ showcdf -c /etc/inittab /usr/adm/sulog /etc/passwd
/etc/inittab+/donald
/usr/adm+/donald/sulog
```

AUTHOR

Showcdf was developed by HP.

SEE ALSO

getcdf(3C), cdf(4).

NAME

size – print section sizes of object files

SYNOPSIS

size [**–d**] [**–o**] [**–x**] [**–V**] *files*

DESCRIPTION

The *size* command produces section size information for each section in the object files. The size of the text, data and bss (uninitialized data) sections are printed along with the total size of the object file. If an archive file is input to the *size* command, the information for all archive members is displayed.

Options

The following options are recognized:

–d Print sizes in decimal. This is the default.

–o Print sizes in octal.

–x Print sizes in hexadecimal.

–V Supply version information on the *size* command.

DEPENDENCIES

Series 300

The **–V** option is not supported.

Series 800

An additional **–v** option causes *size* to print a verbose list of the subspaces in the object files. Each subspace is listed on a separate line with its size, physical address, and virtual address.

SEE ALSO

as(1), cc(1), ld(1). a.out(4), ar(4).

DIAGNOSTICS

size: name: cannot open if *name* cannot be read.

size: name: bad magic if *name* is not an appropriate object file.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

size: SVID2, XPG2

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

RETURN VALUE

Sleep exits with one of the following values:

0 The execution was successfully suspended for *time* seconds, or a SIGALRM signal was received.

>0 If the *time* operand is missing, is not a decimal integer, is negative, or is greater than UINT_MAX, *sleep* returns with exit status 2.

SEE ALSO

alarm(2), sleep(3C).

STANDARDS CONFORMANCE

sleep: SVID2, XPG2, XPG3

NAME

slp − set printing options for a non-serial printer

SYNOPSIS

slp [−**a**] [−**b**] [−**c** *cols*] [−**d**] [−**i** *indent*] [−**k**] [−**l** *lines*] [−**n**] [−**r**] [−**C** *pages*] [−**O** *pages*]

DESCRIPTION

Slp sets printer formatting options such as the number of lines per page, number of characters per line, and indentation. These characteristics are controlled by the printer driver as described in *lp*(7). *Slp* acts on the current standard output.

Options

−**a** Report all option settings.

−**b** Specify a character printer where back spaces pass through the driver unchanged. The absence of this option indicates a line printer. The driver takes the necessary action to accommodate backspace characters.

−**c***cols* Limit the number of columns to be printed to *cols*. Characters beyond the specified limit are truncated.

−**d** Reset options to the defaults for the device. This action is not taken until the next open occurs on the device.

−**i***indent* Indent *indent* columns before printing the first column.

−**k** Select cooked mode. Cooked mode must be used with a cooked device special file which is identified by an lp mnemonic that is not preceded by the character **r**.

−**l***lines* Specify the number of *lines* per page. The last new-line character of each page is changed to a form-feed.

−**n** Set the page size to infinity. Since the last new-line of the page is never encountered, no new-line characters are changed to form-feeds.

−**r** Select a raw mode for graphics dumps. All other options are ignored except −**a**. If −**r** option is not given, −**k** option is assumed.

−**C***pages* Eject zero or more *pages* after the final close of the device.

−**O***pages* Eject zero or more *pages* when the device is opened.

EXAMPLES

In a typical case, the printer is set to 80 columns, no indentation, with no form-feeds between pages:

```
slp -c80 -i0 -n >/dev/lp
```

WARNINGS

Use of the *slp* command in conjunction with the *lp* spooler (see *lp*(1)) might cause undesirable side effects. The spooler model files make assumptions regarding the configuration and can get confused when the default values are altered. Although most options can be altered without difficulty, special problems sometimes result from adjusting the number of lines and the number of columns per page.

DEPENDENCIES

Series 300

The value of *cols* is forced into the range of 1 to 227, the value of *indent* from 0 to 227, and the value of *lines* from 1 to MAXSHORT.

The uppercase-only flag, the no-overprint flag, the raw-mode flag, and no-page-eject-on-open-or-close flag can be selected (enabled) by appropriate use of the minor number in

the *mknod*(1M) command. See the *HP-UX System Administrator Manual* for details.

If the no-page-eject-on-open-or-close flag is enabled by the *mknod*(1M) command, the **−C** and **−O** options are ignored.

AUTHOR

Slp was developed by HP.

SEE ALSO

lp(1), ioctl(2), lp(7).

NAME

soelim – eliminate .so's from nroff input

SYNOPSIS

soelim [*file* ...]

DESCRIPTION

Soelim reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

.so somefile

when they appear at the beginning of input lines. This is useful since programs such as *tbl*(1) do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using ''' instead of '.', i.e.

'so /usr/lib/tmac.s

A sample usage of *soelim* would be

soelim exum?.n | tbl | nroff –ms | col | lp

WARNINGS

The format of the source commands must involve no strangeness – exactly one blank must precede and no blanks follow the file name.

SEE ALSO

more(1), nroff(1), tbl(1).

NAME

softbench – HP SoftBench Software Development Environment

SYNOPSIS

softbench [**-host** *contexthost*] [**-dir** *contextdirectory*] [**-file** *toolsetfile*] [**-nofork**]

REMARKS

The *HP SoftBench* and *HP Encapsulator* products are designed to improve programmer productivity by providing a software development environment that:

- Facilitates rapid, interactive program development and simplifies program maintenance and porting in a distributed computing environment.

- Is easy to learn and use.

- Can be easily customized to leverage a customer's existing environment, processes and tools.

HP SoftBench

HP SoftBench is an integrated set of X11 window-based programming tools and a Tool Integration Platform. Together they provide an integrated software development environment targeted at the program construction, test and maintenance phases of software development.

The HP SoftBench product is composed of:

- A language-sensitive **Program Editor** and **Program Builder** to support rapid, interactive program construction.

- A **Static Analyzer** and **Program Debugger** for understanding the structure and behavior of complex applications in order to support program test, maintenance and porting.

- A **Development Manager** used to manage the versions of files the HP SoftBench tools operate on.

The HP SoftBench programming tools are integrated via services that include:

- A tool communication architecture designed to support a task-oriented environment of cooperative tools.

- Support for a distributed software development environment allowing remote tool execution and remote data access.

- A graphical, *OSF/Motif* style user interface.

- A pervasive, interactive help system.

HP Encapsulator

HP Encapsulator delivers the customizability benefit of HP SoftBench. It allows customers to customize and extend the HP SoftBench environment by:

- Automating custom development processes. HP Encapsulator is used to define actions that will be executed whenever specific events occur in the HP SoftBench environment.

- Adding the HP SoftBench graphical user interface to existing HP-UX utilities, customer tools, and third-party tools.

These extensions are made to the environment without modifying the source code of the tools that are encapsulated.

Product Information

HP SoftBench and HP Encapsulator run on HP 9000 Series 300 and HP 9000 Series 800 computers under HP-UX. Contact your HP Sales Representative for ordering information.

The remainder of this man page describes the HP Softbench Tool Manager and Help Facility, *softbench*(1).

DESCRIPTION

The *softbench*(1) program starts the HP SoftBench Tool Manager, Help Facility, and the HP SoftBench Run-Time Environment. Also, an HP SoftBench **Development Manager** will be started in the context directory (see *softdm (1)*).

The HP SoftBench Tool Manager provides the following capabilities: 1) Tools can be started, stopped, iconified, deiconified, and their contexts can be changed; 2) Sets of tools can be saved away and restored on demand or automatically; 3) The HP SoftBench programming environment can be stopped; and 4) Feedback is provided on the status of each of the tools which are running.

The HP SoftBench Help Facility allows you to access on-line help on applications, specific items on the screen, or by browsing through the list of available topics.

The HP SoftBench Run-Time Environment supports communication between the HP SoftBench tools in a distributed computing environment.

The *softbench*(1) process will fork itself into the background when it is ready. This allows the user to put it in their **$HOME/.x11start** file followed by the explicit start of various tools.

Options

The standard X Toolkit options are all available, as are all the standard HP SoftBench options, described in *softbench*(5). In addition:

–host *contexthost*
: The host filesystem on which the directory lives where HP SoftBench will operate. If this option is not specified, the contexthost is set to the host where the *softbench*(1) process is executing.

–dir *contextdirectory*
: The directory where HP SoftBench will operate. If this option is not specified, the current directory from when HP SoftBench was invoked is used.

–file *toolsetfile*
: If specified, the HP SoftBench Tool Manager will restore the specified toolset file to provide an initial environment.

–nofork
: Normally, the Tool Manager forks itself when it is ready. With this option, the fork will not occur. When invoked through the Execution Manager, this option is required. When HP SoftBench is invoked from a script, the **–nofork** option should not be specified.

USAGE

Tool Manager

The body of the Tool Manager contains a list of all of the running tools. Selecting one or more of the tools allows operations to be specific to the selected tools. Once the user has a desired set of tools for working on a specific project, the toolset can be saved for restoration at a later date.

The tool status information displays the tool class, the tool's status, the host the tool is executing on, and the tool's context (the location of its data).

The Tool Manager will automatically load a saved toolset if the file specified in the resource **saveFile** exists (default is **$HOME/.toolset**), and the resource **autoLoad** is true. This way, a set of tools can automatically be started in the appropriate contexts when the user starts up HP SoftBench. Optionally, the user can specify a saved toolset file on the command line.

HP SoftBench can run two types of programs, *Tools* and *Commands*. Tools are HP SoftBench software tools which are fully integrated with the HP SoftBench Run-Time Environment.

Commands are non-SoftBench programs, such as *vi* or *adb*. When Commands are displayed in the status area, the actions available under the *Tool* menu are not applicable. However, you can still save and restore Commands in toolsets.

Help Facility

The help window contains 5 basic regions, the Pull Down Menu Bar, the Keyword Search Region, the Browser Region, the Browser Manipulation Buttons, and the View Region. The Pull Down Menu Bar is described below. The Keyword Search Region is a place where text may be typed in to be looked up in the documentation. The Browser Region is a scrollable list of items which may be examined at any given time. Clicking on an item causes the detailed description to be displayed in the View Region below. The Browser Manipulation Buttons provide functionality to navigate through the Browser history and to get to a known position.

COMMANDS

Tool Manager Window

The SoftBench Pull Down Menu

Save Toolset...

Prompts for a filename to save the selected toolset to. Initial default is the value of the **saveFile** resource. The complete list of running tools will be saved. The toolset file will be saved relative to the user's home directory unless an absolute path is specified.

Restore Toolset...

Prompts for a toolset file to restore the specified toolset from. Initial default is the value of the **saveFile** resource. The tools will be added to whatever tools may be running. The toolset file will be retrieved relative to the user's home directory unless an absolute path is specified.

About SoftBench...

Displays version information about the HP SoftBench environment.

Quit... The Quit operation refers to the entire HP SoftBench environment. If the **SoftBench** button is selected, then all tools will be stopped, and the run-time environment will be stopped. If the **Tool Manager** button is selected, then just the HP SoftBench Tool Manager will be stopped.

The Tool Pull Down Menu

Minimize

The Minimize operation will cause all of the selected tools to be iconified (minimized).

Open Icon

The Open Icon operation will cause all of the selected tools to be deiconified (opened). As an accelerator, double-clicking on an item will cause it to deiconify.

Stop The Stop operation will cause all of the selected tools to be terminated (they will quit).

Set Context...

Prompts for a new context and instructs the selected tools to change their operating contexts to the new context.

Start... The Start operation provides a dialog where the user can select the tool(s) to be started, the host to execute on, and the location of the data. As an accelerator, double-clicking on a tool is the same as selecting and then pressing the button **Start**. The list of available tools is derived from the Execution Manager initialization file (see section **FILES** below). If the context displayed at the top of the Start dialog box is not the desired context, press the button **New Context** to change the context for

the new tool. The *Execution Host* field is optional, and will default to the host configured in the initialization file. For information on configuring HP SoftBench for remote execution, see the HP SoftBench installation guide.

The Help Pull Down Menu

Item Help
Will provide a question mark cursor for the user to select an item and will then display item-specific help information in the help window described below.

Application Help
Will display general information about the Tool Manager application using the help window described below.

Help Window

The File Pull Down Menu

Quit Will quit the help system. This will not exit *softbench*(1) but merely remove the help window until it is again requested.

The Help Pull Down Menu

Item Help
Will provide a question mark cursor for the user to select an item and will then display item-specific help information in the help window. To select an item, click the left mouse button over the item of interest. If the item is a pull down menu, drag the mouse over the menu and release over the menu button in question.

Application Help
Will display general information about the help window.

Help Accelerator Buttons

Lookup
Will cause the value in the keyword area to be looked up in the help database. The list of tool names and the glossary are searched (case insensitive).

Previous
Will return the display to the previous help query in the history.

Next Will move to the next help query in the query history. This only makes sense after a use of **Previous.**

List Topics
Returns to an overview listing of each tool plus the glossary.

ACCELERATORS

Open Icon
Double-clicking on a tool listed in the running tool list causes that tool to be de-iconified (opened).

Tool Start
Double-clicking on a tool listed in the *Start Tool* dialog causes the specified tool to start.

MESSAGES

The following describes the message interface supported by tool class **TOOLMGR** for use with the HP Encapsulator. See the HP Encapsulator manual for information on using messages. Standard messages understood by HP SoftBench tools are described in *encapsulate*(1).

softbench(1) responds to messages in order to service help requests.

Help Messages
These messages are all requests that tools may make to the help system which will cause the

help window to be displayed with the appropriate text.

Request TOOLMGR HELP-SCREEN-ORIENTED
Notify TOOLMGR HELP-SCREEN-ORIENTED
Failure TOOLMGR HELP-SCREEN-ORIENTED *error-message*

This will cause Item Help mode to be entered (a question mark displayed and the user allowed to select a region of the screen for help).

Request TOOLMGR HELP-TOPIC *topic*
Notify TOOLMGR HELP-TOPIC *topic*
Failure TOOLMGR HELP-TOPIC *topic error-message*

This will request help on the topic specified by *topic.*

Request TOOLMGR HELP-TOPICS
Notify TOOLMGR HELP-TOPICS
Failure TOOLMGR HELP-TOPICS *error-message*

This will request that the help system display itself, but with no specific help topic displayed (the list topics index).

X DEFAULTS

saveFile (class **SaveFile**)
The filename that contains the default restore toolset (Default is **$HOME/.toolset**).

autoSave (class **AutoSave**)
If true, the Tool Manager will automatically save the current toolset when the HP SoftBench environment is stopped via the **SoftBench** Button (Default is **False**).

autoLoad (class **AutoLoad**)
If true, when the Tool Manager is restarted, it will automatically reload the toolset specified in the **saveFile** resource or if specified, the file specified on the command line invocation (Default is **False**).

welcome (class **Welcome**)
If true, the **About SoftBench** message window is displayed when the environment is first started up, which displays version information (Default is **True**).

DEPENDENCIES

HP SoftBench requires the X Version 11 window system.

FILES

$HOME/.softinit
Execution manager initialization table, used upon startup.

/usr/softbench/config/softinit
Default Execution Manager initialization table (if no **$HOME/.softinit** file).

/usr/softbench/help/app-defaults/*
/usr/softbench/help/$LANG/*.cat
/usr/local/softbench/help/app-defaults/*
/usr/local/softbench/help/$LANG/*.cat
/usr/contrib/softbench/help/app-defaults/*
/usr/contrib/softbench/help/$LANG/*.cat
Files containing online help text. These files are only meant to be accessed by the user when adding help information for encapsulated programs. See *encapsulate*(1).

/usr/lib/X11/app-defaults/LibXe,
/usr/lib/X11/app-defaults/Softbench,
/usr/softbench/menus/help.m,
/usr/softbench/menus/softbench.m,

/usr/softbench/schemes/*
/usr/softbench/config/softtypes/$LANG
Internal configuration files for *softbench*(1) application. These files are not meant to be modified by the user.

NOTES

Note that *softbench*(1) is technically an optional facility. It is not required to run any HP SoftBench tool, however the capabilities described above are not available if *softbench*(1) is not used to initiate the HP SoftBench session.

The Minimize, Open Icon, Stop and Set Context operations are not applicable to items of type *Command* (indicated by a status of *Running*).

AUTHOR

softbench was developed by the Hewlett-Packard Company.

SEE ALSO

The following references are contained in other manuals shipped HP SoftBench software, and are not included in the *HP-UX Reference*:

encapsulate(1), *encaprun*(1), *softbench*(5), *softbuild*(1), *softdebug*(1), *softdm*(1), *softedit*(1), *softeditsrv*(1), *softmsg*(1), *softstatic*(1).

INTERNATIONAL SUPPORT

softbench(1) handles both 8-bit and 16-bit languages.

NAME

sort – sort and/or merge files

SYNOPSIS

sort [**−cmu**] [**−o***output*] [**−y***kmem*] [**−z***recsz*] [**−T***dir*] [**−t***x*] [**−bdfilnrM**] [*+pos1* [*−pos2*]] [*file ...*]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if − is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

−c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

−m Merge only, the input files are already sorted.

−u Unique: suppress all but one in each set of lines having equal keys.

−o*output* The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **−o** and *output*.

−y*kmem* The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **−y0** is guaranteed to start with minimum memory. By convention, **−y** (with no argument) starts with maximum memory.

−z*recsz* The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **−c** or **−m** options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

−T*dir* Use *dir* as the directory for temporary sort records rather than the default directory, which is **/usr/tmp**.

The following options override the default ordering rules:

−d Quasi-dictionary order: only letters (isletter()), digits (isdigit()) and blanks (spaces and tabs) are significant in comparisons (see ctype(3c)). The **−d** option is ignored for languages with multi-byte characters; all characters are significant.

−f Fold letters. Prior to being compared, all letters are effectively folded as if by toupper(). The **-f** option is ignored for languages with multi-byte characters; all characters are collated unfolded.

−i In non-numeric comparisons, ignore all characters for which isprint(3c) returns false (see ctype(3c)). For example, character codes 001-037 inclusive and 0177 are ignored in ASCII. The **-i** option is ignored for languages with multi-byte

characters; all characters are significant.

−M Compare as months. The first several non-blank characters of the field are folded to uppercase and compared with the langinfo(3c) items ABMON_1 < ABMON_2 < ... < ABMON_12. An invalid field is considered less than ABMON_1 string. For example, American, month names are compared so that "JAN" < "FEB" < ... < "DEC". An invalid field is considered less than all months. The -M option implies the -b options (see below).

−n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional radix character, is sorted by arithmetic value. The langinfo(3c) item RADIXCHAR is used as the radix character. The **−n** option implies the **−b** option (see below).

−r Reverse the sense of comparisons.

−l This option is ignored. Previously it was used to activate sorting using the collation rules associated with the user's LANG variable (see *environ*(5)). Language sensitive collation is now the standard behavior.

If the language is not specified or is set to the **C** locale, the ordering is lexicographic by bytes in machine-collating sequence. If the user's language includes multi-byte characters, one-byte characters are machine-collated before multi-byte characters.

The notation *+pos1 −pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *−pos2* means the end of the line.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

−t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

−b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **−b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *−pos2* argument (see below). Note that the **−b** option is only effective when restricted sort key specifications are in effect.

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinrM**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect, *n* is counted from the first non-blank in the *m*+1st field; *+m.*0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *−m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect, *n* is counted from the last leading blank in the *m*+1st field; *−m.*1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the default ordering rules applied to the sort.

LC_CTYPE determines the behavior of character classification for the **−d**, **−f**, and **−i** options.

LC_NUMERIC determines the definition of the radix character for the **−n** option.

LC_TIME determines the month names for the **−M** option.

LANG determines the language in which messages are displayed.

If either LC_COLLATE, LC_CTYPE, LC_NUMERIC, or LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used. If any of the internationalization variable contains an invalid setting, *sort* behaves as if all internationalization variables were set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 −2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort −r −o outfile +1.0 −1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2*, using the first non-blank character of the second field as the sort key:

```
sort −r +1.0b −1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort −t: +2n −3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort −um +2 −3 infile
```

DIAGNOSTICS

Sort comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the **−c** option.

When the last line of an input file is missing a new-line character, *sort* appends one, prints a warning message, and continues.

If an error occurs when accessing the tables that contain the collation rules for the specified language, *sort* prints a warning message and defaults to the **C** locale.

If a **−d**, **−f** or **−i** option is specified for a language with multi-byte characters, *sort* prints a warning message and ignores the option.

WARNINGS

A field separator specified by the **−t** option is recognized only if it is a single-byte character.

The ctype(3c) functions isletter(), isdigit(), sspace(), and isprint() are not defined for multi-byte characters. For languages with multi-byte characters, all characters are significant in comparisons.

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1), toupper(3C), collate8(4), environ(5), hpnls(5), langid(5).

STANDARDS CONFORMANCE

sort: SVID2, XPG2, XPG3

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−x**] [**−l**] [**−i**] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

Spell ignores most *troff*, *tbl*(1), and *eqn* constructions.

Options

−v All words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

−b British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*.

−x Every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(1)) follows chains of included files (**.so** and **.nx** *troff* requests), *unless* the names of such included files begin with **/usr/lib**. Under the **−l** option, *spell* will follow the chains of *all* included files. Under the **−i** option, *spell* will ignore all chains of included files.

Under the +*local_file* option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see **FILES**). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy−y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin *n* Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

EXAMPLES

The following example creates the hashed spell list **hlist** and checks the result by comparing the two temporary files; they should be equal.

```
cat goodwds | /usr/lib/spell/hashmake | sort -u >tmp1
cat tmp1 | /usr/lib/spell/spellin ~cat tmp1 | wc -l~ >hlist
cat hlist | /usr/lib/spell/hashcheck >tmp2
diff tmp1 tmp2
```

WARNINGS

The spelling list's coverage is uneven. New installations will probably wish to monitor the output for several months to gather local additions. Typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

The British spelling feature was done by an American.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

VARIABLES

D_SPELL	Your hashed spelling list. (Default as above.)
H_SPELL	Spelling history. (Default as above.)
S_SPELL	Your hashed stop list. (Default as above.)

SEE ALSO

deroff(1), sed(1), sort(1), tbl(1), tee(1).

STANDARDS CONFORMANCE

spell: SVID2, XPG2, XPG3

NAME

split − split a file into pieces

SYNOPSIS

split [−*n*] [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with aa appended, and so on lexicographically, up to zz (only ASCII letters are used, a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if − is given instead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

STANDARDS CONFORMANCE

split: SVID2, XPG2, XPG3

NAME

sqlgen – generate command files to unload, reload an ALLBASE/HP-UX HP SQL relational DataBase Environment

SYNOPSIS

sqlgen

REMARKS

The ALLBASE/HP-UX product must be previously installed on the system for *sqlgen* to function.

DESCRIPTION

Sqlgen invokes the HP SQL utility program for generating command files to unload and reload an HP SQL relational DataBase Environment (DBEnvironment). No options are available with this command. Sqlgen can only be executed by HP SQL users with DBA authority over the DBEnvironment.

Options

–e *editorname* Set the current editor for use with *sqlgen*.

–i Cause *sqlgen* to write its input to the standard output.

AUTHOR

Sqlgen was developed by Hewlett-Packard.

FILES

/usr/bin/hpdbdaemon	Cleanup daemon program file.
/usr/lib/hpsqlproc	HP SQL program file.
/usr/bin/sqlgen	SQLGEN program file.
/usr/lib/hpsqlcat	HP SQL message catalog file.
/usr/lib/nls/$LANG/hpsqlcat	Localized HP SQL message catalog file.

SEE ALSO

ALLBASE/HP-UX HP SQL Database Administration Guide.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

sqlutil – maintain and configure an ALLBASE/HP-UX HP SQL DBEnvironment

SYNOPSIS

sqlutil

REMARKS

The ALLBASE/HP-UX product must be previously installed on the system for *sqlutil* to function.

DESCRIPTION

Sqlutil invokes the HP SQL utility program to maintain and reconfigure an ALLBASE/HP-UX relational DataBase Environment (DBEnvironment). No options are available with this command. *Sqlutil* can be executed on all DataBase Environment Configuration (DBECon) files by all system users; however, only the creator (or a user with designated permission) of the DBEnvironment configuration file can modify it.

AUTHOR

Sqlutil was developed by Hewlett-Packard.

FILES

/usr/bin/hpdbdaemon	Cleanup daemon program file.
/usr/lib/hpsqlproc	HP SQL program file.
/usr/bin/sqlutil	SQLUTIL program file.
/usr/lib/hpsqlcat	HP SQL message catalog file.
/usr/lib/nls/$LANG/hpsqlcat	Localized HP SQL message catalog file.

SEE ALSO

ALLBASE/HP-UX HP SQL Database Administration Guide.
ALLBASE/HP-UX SQL Reference Manual.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

ssp – remove multiple line-feeds from output

SYNOPSIS

ssp

DESCRIPTION

Ssp (single-space) removes redundant blank lines from the standard input and sends the result to the standard output. All blank lines at the beginning of a file are removed, and all multiple blank lines elsewhere in the file (including end-of-file) are reduced to a single blank line.

Ssp is typically used in pipelines like

```
nroff –ms file1 | ssp
```

Ssp is equivalent to the 4.2BSD **cat -s** command.

To remove all blank lines from a file except at beginning of file, use *rmnl*(1). To remove all blank lines from a file including beginning of file, use *rmnl* piped to *ssp*, or *ssp* piped to *rmnl*.

SEE ALSO

cat(1), rmnl(1).

NAME

strings – find the printable strings in an object or other binary file

SYNOPSIS

strings [**–a**] [**–o**] [*–number*] [*file*] ...

DESCRIPTION

Strings looks for ASCII strings in a file. If no *file* is specified, standard input is used. A string is any sequence of 4 or more printing characters ending with a new-line or null character.

Options

–a By default, *strings* looks only in the initialized data space of object files (as recognized by their magic numbers). If this flag is used, the entire file is inspected. This flag is always set if standard input is being read or the file is not recognized as an object file. For backward compatibility, – is understood as a synonym for **–a**.

–o Each string is preceded by its offset in the file (in octal).

–number Specify *number* as the minimum string length, rather than the default 4.

Strings is useful for identifying random object files and many other things.

AUTHOR

Strings was developed by the University of California, Berkeley.

WARNINGS

The algorithm for identifying strings is extremely primitive.

SEE ALSO

od(1).

NAME

strip − strip symbol and line number information from an object file

SYNOPSIS

strip [−**l**] [−**x**] [−**r**] [−**V**] *filename ...*

DESCRIPTION

The *strip* command strips the symbol table and line number information from object files, including archives. Thereafter, no symbolic debugging access is available for that file; thus, this command is normally run only on production modules that have been debugged and tested. The effect is identical to using the −**s** option of *ld*.

Options

The amount of information stripped from the symbol table can be controlled by using any of the following options:

−**l**	Strip line number information only; do not strip any symbol table information.
−**x**	Do not strip static or external symbol information.
−**r**	Reset the relocation indexes into the symbol table.
−**V**	Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* complains and terminates without stripping *filename* unless the −**r** flag is used.

If the *strip* command is executed on an archive file (see *ar*(4)) the archive symbol table is removed. The archive symbol table must be restored by executing the *ar*(1) command with the **s** option before the archive can be link-edited by the *ld*(1) command. *Strip* instructs the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

DIAGNOSTICS

strip: name: cannot open	if *name* cannot be read.
strip: name: bad magic	if *name* is not an appropriate object file.
strip: name: relocation entries present; cannot strip	if *name* contains relocation entries and the −**r** flag is not used, the symbol table information cannot be stripped.

DEPENDENCIES

Series 300

If *name* is a relocatable file, *strip* will remove the local symbols from it. If *name* is an archive file, *strip* removes the local symbols from any *a.out* format files it finds in the archive. Certain libraries, such as those residing in **/lib**, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

The −**l**, −**x**, −**r** and −**V** options are not supported.

Strip removes the debug information from any *a.out* file it finds while stripping an archive file (see *ar*(1)).

Strip does not warn the user that its use on an archive file removes the archive symbol table.

Currently, *strip* does not complain about stripping relocatable files. It strips such files by removing debugging information only.

Series 800

The −**l** and −**x** options are synonymous, since the symbol table contains only static and

external symbols. Either option causes only symbolic debugging information and unloadable data to be stripped. The **-r** option allows strip to be run on relocatable files, in which case the effect is also to strip only symbolic debugging information and unloadable data.

FILES

Series 300

/tmp/s*	temporary files

Series 800

/usr/tmp/strp??????	temporary files

SEE ALSO

ar(1), as(1), cc(1), ld(1), a.out(4), ar(4).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

strip: SVID2, XPG2, XPG3

NAME

stty – set the options for a terminal port

SYNOPSIS

stty [**–a** | **–g** | options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the **–a** option, it reports all of the option settings; with the **–g** option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *termio*(7) for asynchronous lines. Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (**–parenb**) enable (disable) parity generation and detection.

parodd (**–parodd**) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *termio*(7)).

0 hang up phone line immediately.

50 75 110 134.5 150 200 300 600 900 1200 1800 2400 3600 4800 7200 9600 19200 38400 exta extb
Set terminal baud rate to the number given, if possible. (Some speeds are not supported by all hardware interfaces.)

hupcl (**–hupcl**) hang up (do not hang up) modem connection on last close.

hup (**–hup**) same as **hupcl** (**–hupcl**).

cstopb (**–cstopb**) use two (one) stop bits per character.

cread (**–cread**) enable (disable) the receiver.

crts (**–crts**) enable (disable) request-to-send.

clocal (**–clocal**) assume a line without (with) modem control.

loblk (**–loblk**) block (do not block) output from a non-current layer.

Input Modes

ignbrk (**–ignbrk**) ignore (do not ignore) break on input.

ienqak (**–ienqak**) enable (disable) ENQ-ACK handshaking.

brkint (**–brkint**) signal (do not signal) INTR on break.

ignpar (**–ignpar**) ignore (do not ignore) parity errors.

parmrk (**–parmrk**) mark (do not mark) parity errors (see *termio*(7)).

inpck (**–inpck**) enable (disable) input parity checking.

istrip (**–istrip**) strip (do not strip) input characters to seven bits.

inlcr (**–inlcr**) map (do not map) newline (NL) to carriage return (CR) on input.

igncr (**–igncr**) ignore (do not ignore) CR on input.

icrnl (**–icrnl**) map (do not map) CR to NL on input.

iuclc (**–iuclc**) map (do not map) uppercase alphabetics to lowercase on input.

ixon (**–ixon**) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (−ixany) allow any character (only DC1) to restart output.

ixoff (−ixoff) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (−opost) post-process output (do not post-process output; ignore all other output modes).

olcuc (−olcuc) map (do not map) lowercase alphabetics to uppercase on output.

onlcr (−onlcr) map (do not map) NL to CR-NL on output.

ocrnl (−ocrnl) map (do not map) CR to NL on output.

onocr (−onocr) do not (do) output CRs at column zero.

onlret (−onlret) on the terminal NL performs (does not perform) the CR function.

ofill (−ofill) use fill characters (use timing) for delays.

ofdel (−ofdel) fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 select style of delay for carriage returns (see *termio*(7)).

nl0 nl1 select style of delay for line-feeds (see *termio*(7)).

tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *termio*(7).

bs0 bs1 select style of delay for backspaces (see *termio*(7)).

ff0 ff1 select style of delay for form-feeds (see *termio*(7)).

vt0 vt1 select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (−isig) enable (disable) the checking of characters against the special control characters INTR and QUIT.

icanon (−icanon) enable (disable) canonical input (ERASE and KILL processing).

xcase (−xcase) canonical (unprocessed) uppercase and lowercase presentation.

echo (−echo) echo back (do not echo back) every character typed.

echoe (−echoe) echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (−echok) echo (do not echo) NL after KILL character.

lfkc (−lfkc) the same as **echok (−echok)**; obsolete.

echonl (−echonl) echo (do not echo) NL.

noflsh (−noflsh) disable (enable) flush after INTR or QUIT.

tostop (−tostop) enable (disable) generation of SIGTTOU signals when background jobs attempt output.

Control Assignments

control-character c set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **eof**, **eol**, **min**, or **time** (**min** and **time** are used with **−icanon**; see *termio*(7)). For systems that support job control, **susp** and **dsusp** characters may also be set. For systems that support shell layers (see *shl*(1)) **swtch** may also be set. If *c* is preceded by an (escaped from the shell) caret (^), the value used is the corresponding CTRL character (e.g., "^**d**" is a

CTRL-d); "^**?**" is interpreted as DEL and "^**-**" is interpreted as undefined.

line *i* set line discipline to *i* ($0 < i < 127$). (See *termio*(7)).

Combination Modes

evenp or **parity** enable **parenb** and **cs7**.

oddp enable **parenb**, **cs7**, and **parodd**.

-parity, **-evenp**, or **-oddp**
disable **parenb**, and set **cs8**.

raw (**-raw** or **cooked**)
enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).

nl (**-nl**) unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**-lcase**) set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (**-LCASE**) same as **lcase** (**-lcase**).

tabs (**-tabs** or **tab3**) preserve (expand to spaces) tabs when printing.

ek reset ERASE and KILL characters back to normal **#** and

sane resets all modes to some reasonable values.

term set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, **hp**,

EXAMPLES

The command:

stty kill '^X' intr '^C'

sets the delete line character to ^**X** (the control key and the x key) and the interrupt character to ^**C**. This command is usually found in the **.login** or **.profile** files so that ^**X** and ^**C** need not be set by the user at each login session.

DEPENDENCIES

Refer to the DEPENDENCIES section of *termio*(7) for a further description of the capabilities that are not supported.

SEE ALSO

shl(1), tabs(1), ioctl(2), termio(7).

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the valid control characters for printing.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *stty* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

stty: SVID2, XPG2, XPG3

NAME

su – become super-user or another user

SYNOPSIS

su [–] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID, real and effective group ID, and group access list set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or **/bin/sh** if none is specified (see *sh*(1)). To restore normal user ID privileges, type an **EOF** to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell, permitting the super-user to run shell procedures with restricted privileges. When using programs like *sh*(1), an *arg* of the form **–c** *string* executes *string* via the shell and an arg of **–r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a –, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is –, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along unchanged, except that **$PATH**, is unconditionally set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **–sh** or **–su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(1).

The – option always resets **$PATH** to **/bin:/etc:/usr/bin** for the super-user, and **/bin:/usr/bin** for all others. However, the files */etc/profile* and *.profile* are normally executed anyway, thus restoring the intended value of **$PATH**.

All attempts to become another user are logged in /usr/adm/sulog, including failures. Successful attempts are flagged with "+", failures with "-".

EXAMPLES

To become user **bin** while retaining your previously exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

```
su - bin -c "command args"
```

FILES

$HOME/.profile	user's profile

/etc/logingroup	system's default group access list file
/etc/passwd	system's password file
/etc/profile	system's profile
/usr/adm/sulog	log of all attempts

VARIABLES

HOME	the user's home directory
LOGNAME	the user's login name
PATH	the command name search path
PS1	the default prompt
SHELL	the name of the user's shell

SEE ALSO

env(1), login(1), sh(1), initgroups(3C), group(4), passwd(4), profile(4), environ(5).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *su* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Characters in the 7-bit USASCII code sets are supported in login names (see *ascii*(5)).

STANDARDS CONFORMANCE

su: SVID2, XPG2

NAME

sum – print checksum and block count of a file

SYNOPSIS

sum [**–r**] [file]

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. **Stdin** is used if no file names are given. *Sum* is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **–r** causes an alternate algorithm to be used in computing the checksum.

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

SEE ALSO

wc(1).

STANDARDS CONFORMANCE

sum: SVID2, XPG2, XPG3

NAME

tabs – set tabs on a terminal

SYNOPSIS

tabs [tabspec] [**+mn**] [**–T**type]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

If you are using a non-HP terminal, you should keep in mind that behavior will vary for some tab settings.

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **–8**, i.e., HP-UX "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0.

–code Gives the name of one of a set of "canned" tabs. The legal *code*s and their meanings are as follows:

–a 1,10,16,36,72
Assembler, IBM S/370, first format

–a2 1,10,16,40,72
Assembler, IBM S/370, second format

–c 1,8,12,16,20,55
COBOL, normal format

–c2 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t–c2 m6 s66 d:>

–c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **–c2.** This is the recommended format for COBOL. The appropriate format specification is:

<:t–c3 m6 s66 d:>

–f 1,7,11,15,19,23
FORTRAN

–p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

–s 1,10,55
SNOBOL

–u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

–n A repetitive specification requests tabs at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value **–8**: this represents the HP-UX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff*(1) **–h** option for high-speed output. Another special case is the value **–0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the

previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

−−*file* If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **−8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

tabs −− file; pr file

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

−**T***type* *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(5). If no **−T** flag is supplied, *tabs* searches for the **$TERM** value in the *environment* (see *environ*(5)). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+**m***n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

DIAGNOSTICS

illegal tabs	when arbitrary tabs are ordered incorrectly.
illegal increment	when a zero or missing increment is found in an arbitrary specification.
unknown tab code	when a "canned" code cannot be found.
can't open	if −−*file* option used, and file can't be opened.
file indirection	if −−*file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

nroff(1), pr(1), tset(1), environ(5), term(5).

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text within file as single and/or multi-byte characters.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *tabs* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

tabs: SVID2, XPG2, XPG3

NAME

tail − deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +*number* from the beginning, or −*number* from the end of the input (if *number* is null, the value -10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **−f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process.

EXAMPLES

Tail accepts at most two arguments: the first consists of specified options, and the second specifies the file of interest. If the *number* and **f** options are both desired, they must be concatenated to create a single option argument, as shown in this example:

```
tail -3lf john
```

This example prints the last three lines in the file **john** to the standard output, and leaves *tail* in "follow" mode.

If only the **f** option is desired, it must be preceded by a −, as follows:

```
tail −f fred
```

This example prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. Note that this output may build up very quickly for rapidly changing input files, perhaps too fast to read on a CRT.

As another example, the command:

```
tail −15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

The + option starts at the number indicated from the beginning of the file (rather than skipping the number of units indicated and then starting.) For example:

```
tail +1b fred
```

prints the entire contents of the file *fred*.

SEE ALSO

dd(1), head(1).

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Thus, be wary of the results when piping output from other commands into *tail*.

Various kinds of anomalous behavior may happen with character special files.

Tail can pick up a maximum of 4K bytes of data from the specified file.

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE
tail: SVID2, XPG2, XPG3

NAME

tar – tape file archiver

SYNOPSIS

tar *key* [*arg ...*] [*file* | -C *directory*] ...

DESCRIPTION

Tar saves and restores archives of files on a magnetic tape, a flexible disk, or a regular file. Its actions are controlled by the *key* argument. The *key* is a string of characters containing an optional version letter, exactly one function letter, and possibly one or more function modifiers. The *key* string can be preceded by a hyphen (–) (as when specifying options in other HP-UX commands), but it is not necessary.

Next, an additional *arg* argument is required for each of the function modifiers **b** or **f** used (see below). If both **b** and **f** modifiers are specified, the order of the associated *arg* arguments must match the order of the modifiers.

Each *file* argument specifies a file being saved or restored. If *file* is a directory name, it refers to the files and (recursively) subdirectories contained in that directory.

The version portion of the *key* determines in which format *tar* writes the archive. *Tar* can read either format regardless of the version. The version is specified by one of the following letters:

N Write a new (POSIX) format archive. The new format allows file names up to 256 characters in length, and correctly archives and restores the following file types: regular files, character and block special devices, links, symbolic links, directories, and FIFO special files. The new version also stores the user and group name of each file and attempts to use these names to determine the *uid* and *gid* of a file when restoring with the **p** function modifier.

O Write an old (pre-POSIX) format archive. This is the current default.

The function portion of the *key* is specified by one of the following letters:

c Create a new archive; write from the beginning of the archive instead of starting after the last *file*. All previous information in the archive is overwritten.

r Add the named *file* to the end of the archive.

t List the names of all the files on the archive. Adding the **v** function modifier expands this listing to include the file modes and owner numbers. The names of all files are listed each time they occur on the tape.

u Add any named *file* to the archive, if it is not already present or has been modified since last written on that archive.

x Extract the named *file* from the archive and restore it to your system. If a named *file* matches a directory whose contents was written to the archive, this directory is (recursively) extracted. If a named *file* on tape does not exist on the system, the *file* is created as follows:

- The user, group, and other protections are restored from the tape.
- The modification time is restored from the tape unless the **m** function modifier is specified.
- The file owner (*uid*) and group owner (*gid*) are normally that of the restoring process.
- The set-user-ID, set-group-ID, and sticky bits are not set automatically. The **o** and **p** function modifiers control the restoration of protection; see below for more details.

If the files exist, their modes are not changed, but the set-user-ID, set-group-ID and sticky bits are cleared. If no *file* argument is given, the entire content of the archive is extracted. Note that if several files with the same name are on the archive, the last one overwrites all earlier ones.

The following function modifiers can be used in addition to the function letters listed above (note that some modifiers are incompatible with some functions):

A Suppress warning messages that *tar* did not archive a file's access control list. By default, *tar* writes a warning message for each file with optional ACL entries.

b Cause *tar* to use the next *arg* argument as the blocking factor for archive records. The default is 20; the maximum is at least 20. However, if the **f** – modifier is used to specify the standard input, the default blocking factor is 1.

Blocking factor is determined automatically when reading 9-track tapes (key letters **x** and **t**). [On 9-track tapes, the physical tape record length is the same as the block size. The block size is defined as the logical record size times the blocking factor (number of logical records per block).] The blocking factor must be specified when reading flexible disks and cartridge tapes if they were written with a blocking factor other than the default.

If a *tar* file is read using a blocking factor not equal to the blocking used when the file was written, an error may occur at the end of the file but there may or may not be an actual error in the read. To prevent this problem, a blocking factor of **1** can be used, although performance may be reduced somewhat. *Tar* writes logical records of 512 bytes, independent of how logical records may be defined elsewhere by other programs (such as variable-length records (lines) within an ASCII text file).

f Cause *tar* to use the next *arg* argument as the name of the archive instead of **/dev/rmt/0m.** If the name of the file is –, *tar* writes to the standard output or reads from the standard input, whichever is appropriate, and the default blocking factor becomes 1. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:
cd *fromdir*; **tar cf** – **.** | **(cd** *todir*; **tar xf** –**)**

h Force *tar* to follow symbolic links as if they were normal files or directories. Normally, *tar* does not follow symbolic links.

H Cause all entries in hidden directories (context-dependent files) to be written to the archive (see *cdf*(4)). Normally, *tar* only writes the entry in the CDF that matches the context of the *tar* process. See *getcontext*(2). This modifier works only when writing archives. When reading archives, *tar* automatically restores hidden directories if the archive was created with the **H** modifier.

l Tell *tar* to complain if it cannot resolve all of the links to the files being saved. If **l** is not specified, no error messages are printed.

m Tell *tar* not to restore the modification time written on the archive. The modification time of the file will be the time of extraction.

o Suppress writing certain directory information that older versions of *tar* cannot handle on input. *Tar* normally writes information specifying owners and modes of directories in the archive. Earlier versions of *tar*, when encountering this information, give error messages of the form:

<name>/: **cannot create.**

This function modifier suppresses writing that information.

When **o** is used for reading, it causes the extracted *file* to take on the user and group ID (*uid* and *gid*) of the user running the program rather than those of the tape. This is the default for the ordinary user and can be overridden, to the extent that system protections allow, by using the **p** function modifier.

p Cause *file* to be restored to the original modes and ownerships written on the archive, if possible. This is the default for the superuser, and can be overridden by the **o** function modifier. If system protections prevent the ordinary user from executing *chown*(2), the error is ignored, and the ownership is set to that of the restoring process. Set-user-ID, set-group-ID, and sticky bit information are restored as allowed by the protections defined by *chmod*(2), if the *chown* operation above succeeds.

#*d* Specify a particular tape drive and density where **#** is a tape drive number (**0,...,7**), and *d* is the density: (**l** = low (800 bpi), **m** = medium (1600 bpi), or **h** = high (6250 bpi)). This modifier selects the drive on which the nine-track tape is mounted. The default is **0m**.

v Normally, *tar* does its work silently. The **v** (verbose) function modifier causes *tar* to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

V Same as the **v** function modifier except that when using the **t** option, *tar* also prints out a letter indicating the type of the archived file.

w Cause *tar* to print the action being taken, followed by the name of the file, then wait for the user's confirmation. If the user answers **y**, the action is performed. Any other input means "no."

The following option can be included in the file list:

−C*directory* *Tar* performs a *chdir*(2) to *directory*. This allows multiple directories not related by a close or common parent to be archived using short relative path names.

When the end of tape is reached, *tar* prompts the user for a new special file and continues.

If a nine-track tape drive is used as the output device, it must be configured in Berkeley compatibility mode; see *mt*(7).

ERRORS

Tar complains about bad key characters and tape read/write errors.

Tar complains if not enough memory is available to hold the link tables.

EXAMPLES

This example creates a new archive on **/dev/rfd.0** and copies *file1* and *file2* onto it, using the default blocking factor of 20. The *key* is made up of one function letter (**c**) and two function modifiers (**v**, and **f**):

```
tar cvf /dev/rfd.0 file1 file2
```

This example archives files from **/usr/include** and **/etc**:

```
tar cv -C /usr include -C / etc
```

WARNINGS

The default version will probably change from **O** to **N** in a future release.

Due to internal limitations in the header structure, not all file names of fewer than 256 characters fit when using the **N** version key. If a file name does not fit, *tar* prints a message and does not archive the file.

Link names are still limited to 100 characters when using the **N** version key.

There is no way to ask for the *n*-th occurrence of a file.

Tape errors are handled ungracefully.

The **u** function key can be slow.

If the archive is on a flexible disk or cartridge tape, and if the blocking factor specified on output is not the default, the same blocking factor must be specified on input. This is because the blocking factor is not explicitly stored on the archive. Not following this rule and updating the archive can destroy it.

Some previous versions of *tar* have claimed to support selective listing of file names using the **t** function key with a list. This appears to be an error in the documentation because the capability does not appear in the original source code.

There is no way to restore an absolute path name to a relative position.

Tar always pads information written to an archive up to the next multiple of the block size. Therefore, if you are creating a small archive and write out one block of information, *tar* reports that one block was written, but the actual size of the archive might be larger if the **b** function modifier is used.

Note that **tar c0m** is not the same as **tar cm0**.

Do not create archives on block special devices. Attempting to do so can causes excessive wear, leading to premature drive hardware failure.

AUTHOR

Tar was developed by AT&T, the University of California, Berkeley, HP, and POSIX.

FILES

/dev/rmt/*
/dev/rfd.*
/tmp/tar*

SEE ALSO

ar(1), cpio(1), getcontext(2), cdf(4), mt(7), acl(5).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings output when listing the contents of an archive with the **−v option.**

LANG determines the language equivalent of **y** (for yes/no queries).

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *tar* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

tar: SVID2, XPG2, XPG3

NAME

tbl – format tables for nroff

SYNOPSIS

tbl [**−TX**] [*file* ...]

DESCRIPTION

Tbl is a preprocessor that formats tables for *nroff*(1). The input files are copied to the standard output, except for lines between **.TS** and **.TE** command lines, which are assumed to describe tables and are re-formatted by *tbl*. (The **.TS** and **.TE** command lines are not altered by *tbl*).

.TS is followed by global options. The available global options are:

center center the table (default is left-adjust);
expand make the table as wide as the current line length;
box enclose the table in a box;
doublebox enclose the table in a double box;
allbox enclose each item of the table in a box;
tab (*x*) use the character *x* instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi-colon (**;**).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc. The available key-letters are:

c center item within the column;
r right-adjust item within the column;
l left-adjust item within the column;
n numerically adjust item in the column: units positions of numbers are aligned vertically;
s span previous item on the left into this column;
a center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
^ span down previous entry in this column;
_ replace this entry with a horizontal line;
= replace this entry with a double horizontal line.

The characters **B** and **I** stand for the bold and italic fonts, respectively; the character | indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by **.TE**. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ or =, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only _ or =, then that item is replaced by a single or double line.

The **−TX** option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments (or if − is specified as the last argument), *tbl* reads the standard input, so it may be used as a filter. When it is used with *neqn*(1), *tbl* should come first to minimize the volume of data passed through pipes.

EXAMPLES

If we let <tab> represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population
_
Town<tab>Households
<tab>Number<tab>Size
=
Bedminster<tab>789<tab>3.26
Bernards Twp.<tab>3087<tab>3.74
Bernardsville<tab>2018<tab>3.30
Bound Brook<tab>3425<tab>3.04
Bridgewater<tab>7897<tab>3.81
Far Hills<tab>240<tab>3.19
.TE
```

yields:

Household Population		
Town	*Households*	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

The *tbl* command is used most often with *nroff*(1) and *col*(1). A common usage is:

```
tbl filename | nroff | col
```

WARNINGS

See WARNINGS under *nroff*(1).

SEE ALSO

col(1), mm(1), neqn(1), nroff(1), soelim(1), mm(5).

Tbl tutorial, found in *HP-UX Concepts and Tutorials: Text Formatters.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters.

LC_NUMERIC determines the radix character used in numerical data.

LANG determines the language in which messages are displayed.

If LC_CTYPE or LC_NUMERIC is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *tbl* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

tcio – Command Set 80 Cartridge Tape Utility

SYNOPSIS

tcio **–o[dervVZ]** [**–S** *buffersize*] [**–l** *number* [**–n** *limit*]] [**–T** *tty*] *file*
tcio **–i[drvZ]** [**–S** *buffersize*] [**–l** *number* [**–n** *limit*]] [**–T** *tty*] *file*
tcio **–u[rvV]** [**–m** *blocknumber*] [**–l** *number*] *file*

DESCRIPTION

Tcio is designed to optimize the data transfer rate between certain cartridge tape units and the host processor. When used in conjunction with other utilities (such as *cpio*(1)) a significant improvement in throughput can be obtained, in addition to reducing the wear and tear on the tape cartridges and drives. With autochanger mechanisms, *tcio* provides the capability of loading a specified cartridge, or automatically switching to successive cartridges as needed. With the utility operation, *tcio* provides functions that are unique to cartridge tapes.

Tcio **–o** (copy out) reads the standard input and writes the data to the Command Set 80 Cartridge Tape Unit specified by *file*.

Tcio **–i** (copy in) reads the Command Set 80 Cartridge Tape Unit specified by *file* and writes the data to the standard output.

Tcio **–u** (utility) performs utility functions on the cartridge tape, such as unload, mark, and/or verify the cartridge.

In all cases, *file* must refer to a character special file associated with a Command Set 80 cartridge tape unit.

With the output and input operations, *tcio* enables immediate report mode on cartridge tape units that support this mode (see DEPENDENCIES). During writing, this mode enables the drive to complete a write transaction with the host before the data has actually been written to the tape from the drive's buffer. This allows the host to start gathering data for the next write request while the data for the previous request is still in the process of being written. During reading, this mode enables the drive to read ahead after completing a host read request. This allows the drive to gather data for future read requests while the host is still processing data from the previous read request. Under favorable conditions, immediate report mode allows the drive to stream the tape continuously across multiple read/write requests, rather than having to reposition the tape between each read/write request. See ct(7) for further details.

By default, *tcio* puts a tape mark in the first block on each tape to prevent the tape from being image restored over a disk. It also utilizes the last block on each tape to flag whether or not the tape is the last tape in a multi-tape sequence.

The following command options are recognized. One of the options **–o**, **–i**, or **–u** must be specified. Additional options can be specified in any order, but all must precede the *file* name. Options without parameters can be listed individually or grouped together. Options with parameters require the parameter and must be listed individually.

Options

–S *buffersize* Enable specification of buffer size. This option forces the allocation of a block of memory to be used in reading or writing the tape. The size in bytes of the buffer is 1024 times the value specified for *buffersize*. A *buffersize* less than 4 is silently increased to 4; a *buffersize* greater than 64 is silently decreased to 64. If *buffersize* is not specified, *tcio* allocates a 64-kilobyte buffer.

On tape units that support immediate report, a significant performance increase can often be obtained by using a smaller buffer -- 8 kilobytes is the recommended buffer size for these units. On tape units that do not support the immediate report mode, or on tape units that share a controller with a disk (see

DEPENDENCIES) that is simultaneously being accessed, an increase in performance can usually be obtained with a larger buffer. Sixty-four kilobytes, the default, is the recommended buffer size for these units.

−T *tty* Specify *tty* as an alternative to **/dev/tty**. Normally **/dev/tty** is opened by *tcio* when terminal interaction is required. The specified file *tty* is opened instead of **/dev/tty**.

−V This option turns off tape verification. Some cartridge tape units (see DEPENDENCIES) provide hardware for verifying the data output to the tape (called "read-while-write"). For these units software-driven verification is redundant, and this option is suggested.

For drives that do not have the read-while-write hardware, a separate verification operation is suggested. Thus, it is recommended that this option not be used with drives that do not support read-while-write.

−Z This option prevents *tcio* from writing a file mark in the first and last blocks. This option should be used with care, since a tape without a tape mark in block zero can be image-restored to a disk.

−d Print a checksum to standard error. The checksum is a 32-bit unsigned addition of all bytes written to or read from the tape, providing an extra check of data validity (in addition to tape verification). The checksum value is only reported to the user, and is not written on the media; thus, the user must manually record and check it. The checksum is valid *only* if the same number of bytes are read from the tape as were written to it; in other words, the checksum as a data verification test is meaningless unless the **−e** option was used when writing the tape. This option is independent of the verbose modifier.

−e Cause a tape mark to be written on the nearest 1024-byte boundary following the end of the data. When a tape containing an end-of-data tape mark is read back, the read terminates upon encountering the tape mark. Thus, by using this option, the checksums generated by the input and output operations are guaranteed to agree.

−l *number* This option is intended solely for autochanger-type tape units. With the input or output operations (**−i** or **−o**) the autochanger option selects the cartridge from the magazine with which the transfer begins. When used with the utility function (**−u** option), *tcio* loads the specified cartridge into the drive. (Note: the autochanger must be in selective mode for the autochanger options to work properly.)

−m *blocknumber*
This option writes a tape mark on a tape at the specified block. A tape mark in block zero of the tape prevents it from being image-restored to a disk.

−n *limit* This option specifies the maximum number of cartridges to be used in a multi-tape transfer. It applies only to autochanger type units, and must be preceded by the **−l** option. Thus, **−l** starts the transfer by loading cartridge *number* and uses at most *limit* cartridges. If **−l** is specified without **−n**, *tcio* quietly assumes the remaining cartridges (in ascending order) from the magazine.

−r Unload the tape from the drive. On autochanger units, the tape is returned to the magazine.

−v Verbose mode; prints information and error messages to standard error.

EXAMPLES

The first example below copies the contents of a directory into an archive; the second restores

it:

ls | cpio −o | tcio −o /dev/rct/c0d1

tcio −i /dev/rct/c0d1 | cpio −i

To unload the cartridge from the drive (without verifying the tape) execute:

tcio -urV /dev/rct/c0d1

The next example copies all files in the current directory to the tape specified by the device file **/dev/rct/c1d0s2**. Because the device has a read-while-write head, verify is turned off; a buffer size (option **-S)** of 8 blocks (that is, 8 kilobytes) is specified:

ls | cpio -o | tcio -oV -S 8 /dev/rct/c1d0s2

The next example assumes that the cartridge tape unit is an autochanger, on controller 2, with 8 tapes in the magazine. The *tcio* operation starts writing with cartridge 3, and uses at most 4 cartridges before prompting the user for additional media:

find usr -cpio | tcio -oV -S 8 -l 3 -n 4 /dev/rct/c2

DEPENDENCIES

HP 7941CT, HP 9144A, and HP 35401

These cartridge tape devices support the immediate report mode and provide the read-while-write hardware.

HP 7942, HP 7946

These cartridge tape devices support the immediate report mode and provide the read-while-write hardware. The use of a small buffer size is not recommended with these shared controller devices when there is simultaneous access to the disk, because the disk accesses prevents proper tape streaming.

HP 7908, HP 7911, HP 7912, and HP 7914

These cartridge tape devices do not support the immediate report mode and provide the read-while-write hardware.

AUTHOR

Tcio was developed by HP.

SEE ALSO

ct(7).

HP-UX System Administrator Manual.

NAME

tee – pipe fitting

SYNOPSIS

tee [**–i**] [**–a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The **–i** option ignores interrupts; the **–a** option causes the output to be appended to the *files* rather than overwriting them.

RETURN VALUE

Exit values are:

0	Successful completion.
>0	Error condition occured.

EXAMPLES

The command:

who | tee -a hunt

writes a list of users to the screen and also appends the list to the file **hunt**.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG.

If any internationalization variable contains an invalid setting, *tee* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

tee: SVID2, XPG2, XPG3

NAME

test – condition evaluation command

SYNOPSIS

test *expr*
[*expr*]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned. *Test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

−r *file*	true if *file* exists and is readable.
−w *file*	true if *file* exists and is writable.
−x *file*	true if *file* exists and is executable.
−f *file*	true if *file* exists and is a regular file.
−d *file*	true if *file* exists and is a directory.
−c *file*	true if *file* exists and is a character special file.
−b *file*	true if *file* exists and is a block special file.
−p *file*	true if *file* exists and is a named pipe (fifo).
−u *file*	true if *file* exists and its set-user-ID bit is set.
−g *file*	true if *file* exists and its set-group-ID bit is set.
−k *file*	true if *file* exists and its sticky bit is set.
−s *file*	true if *file* exists and has a size greater than zero.
−H *file*	true if *file* exists and is a hidden directory (see *cdf*(4)).
−h *file*	true if *file* exists and is a symbolic link.
−t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
−z *s1*	true if the length of string *s1* is zero.
−n *s1*	true if the length of the string *s1* is non-zero.
s1 = *s2*	true if strings *s1* and *s2* are identical.
s1 != *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 **−eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** can be used in place of **−eq**.

These primaries can be combined with the following operators:

!	unary negation operator.
−a	binary *and* operator.
−o	binary *or* operator (**−a** has higher precedence than **−o**).
(expr **)**	parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and therefore must be escaped.

Test is interpreted directly by the shell.

WARNINGS

In the second form of the command (that is, when using **[]**, instead of the word *test*), the square brackets must be delimited by blanks.

The expression **-x** *filename* always tests true for super-user, even if *filename* is not executable.

AUTHOR

Test was developed by the University of California, Berkeley and HP.

SEE ALSO

find(1), sh(1), cdf(4).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

test: SVID2, XPG2, XPG3

NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The execution time can depend on the performance of the memory in which the program is running.

The times are printed on standard error.

SEE ALSO

times command in sh(1), timex(1), times(2).

STANDARDS CONFORMANCE

time: SVID2, XPG2, XPG3

NAME

timex – time a command; report process data and system activity

SYNOPSIS

timex [**−o**] [**−p[fhkmrt]**] [**−s**] *command*

DESCRIPTION

The elapsed time, user time and system time spent in execution of the given *command* are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on the standard error.

Options

−o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

−p[fhkmrt] List process accounting records for *command* and all its children. The suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported. They behave as defined in *acctcom*(1M). The number of blocks read or written and the number of characters transferred are always reported.

−s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

EXAMPLES

A simple example:

```
timex −ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex −opskmt sh
        session commands
EOT
```

WARNINGS

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

SEE ALSO

sar(1), acctcom(1M).

STANDARDS CONFORMANCE

timex: SVID2

NAME

touch – update access, modification, and/or change times of file

SYNOPSIS

touch [–**amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch causes the access, modification, and last change times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used. The –**a** and –**m** options cause touch to update only the access or modification times respectively (default is –**am**). The –**c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1), utime(2).

EXTERNAL INFLUENCES

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

touch: SVID2, XPG2, XPG3

NAME

tput – query terminfo database

SYNOPSIS

tput [**–T** type] capname

DESCRIPTION

Tput uses the *terminfo* (4) database to make terminal-dependent capabilities and information available to the shell. *Tput* outputs a string if the attribute (**cap**ability **name**) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, tput simply sets the exit code (0 for TRUE, 1 for FALSE), and does no output.

-T*type* indicates the type of terminal. Normally this flag is unnecessary, as the default is taken from the environment variable **$TERM.**

capname indicates the attribute from the *terminfo* database. See *terminfo* (4).

EXAMPLES

tput clear	Echo clear-screen sequence for the current terminal.
tput cols	Print the number of columns for the current terminal.
tput -Thp2623 cols	Print the number of columns for the hp2623 terminal.
bold='tput smso'	Set shell variable "bold" to stand-out mode sequence for current terminal. This might be followed by a prompt: **echo "${bold}Please type in your name: \c"**
tput hc	Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/usr/lib/terminfo/?/* Terminfo data base

/usr/include/curses.h Definition files
/usr/include/term.h

DIAGNOSTICS

Tput prints error messages and returns the following error codes on error:

–1 Usage error.
–2 Bad terminal type.
–3 Bad capname.

In addition, if a capname is requested for a terminal that has no value for that capname (e.g., **tput -Thp2623 vt**), **-1** is printed.

SEE ALSO

stty(1), terminfo(4).

STANDARDS CONFORMANCE

tput: SVID2

NAME

tr – translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **−cds** may be used:

−c Complements the set of characters in *string1* with respect to the all the characters contained in the current character set.

−d Deletes all input characters in *string1*.

−s Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[*c1*−*c2***]** Stands for the ordered string of collating elements *c1* to *c2*, inclusive.

[[:*class***:]]** Stands for the string of characters in type *class*. *Class* must be one of **alpha**, **upper**, **lower**, **digit**, **xdigit**, **alnum**, **space**, **print**, **punct**, **graph** or **cntrl**. Character classes are expanded in collation order.

[[=*c***=]]** Stands for an equivalence class, i.e., all characters defined as having the same primary collation sequence number as *c*.

[[.*cc***.]]** Stands for the collating element *cc*. Multi-character collating elements must be represented in this manner to distinguish them from a list of single-character collating elements.

[a***n*****]** Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order of collating elements, the members of equivalence classes, the order in which character classes are expanded, and the identification of multi-character collating elements.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the characters matched by character classes, and the current universe of characters when using the **-c** option.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *tr* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

EXAMPLES

1. For the ASCII character set and default collation sequence, the following creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

   tr −cs "[A−Z][a−z]" "[\012*]" <file1 >file2

2. The following example does the same as the one above but will work for all character sets and collation sequences.

   tr −cs "[[:alpha:]]" "[\012*]" <file1 >file2

3. The next example may translate all lower-case characters in *file1* to upper-case and write the results to standard output.

   tr "[[:lower:]]" "[[:upper:]]" <file1

   Note that character classes specified in either *string1* or *string2* are expanded in collation order. As a consequence, this example will not produce the desired effect in locales in which there is not a one to one mapping between lower- and upper-case letters, or in which lower- and upper-case letters collate in a different relative order.

4. This example uses an equivalence class to identify accented variants of the base character **e** in *file1*, which are stripped of diacritical marks and written to *file2*.

   tr "[[=e=]]" "[e*]" <file1 >file2

5. This example translates instances of the multi-character collating element **ch** to upper-case. Note that the single characters **c** and **h** will not be affected by this operation unless they are part of a **ch** character sequence.

   tr "[[.ch.]]" "[[.CH.]]" <file1 >file2

SEE ALSO

ed(1), sh(1), ascii(5), environ(5), lang(5), regexp(5).

BUGS

Will not handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

STANDARDS CONFORMANCE

tr: SVID2, XPG2, XPG3

NAME

true, false – return zero or one exit status respectively

SYNOPSIS

true

false

DESCRIPTION

The command *true* does nothing, and returns exit code zero. The command *false* does nothing, and returns exit code one. They are typically used to construct command procedures.

RETURN VALUE

Exit values are:

0 always from *true*.
1 always from *false*.

EXAMPLES

This command loop will be executed forever:

```
while true
    do
        command
    done
```

SEE ALSO

sh(1).

STANDARDS CONFORMANCE

true: SVID2, XPG2, XPG3

false: SVID2, XPG2, XPG3

NAME

tset – terminal-dependent initialization

SYNOPSIS

tset [options] [**–m** [ident] [test baudrate] :type] ... [type]

reset ...

DESCRIPTION

Tset sets up your terminal when you first log in to an HP-UX system. It does terminal-dependent processing, such as setting erase and kill characters, setting or resetting delays, and sending any sequences needed to properly initialize the terminal. It first determines the *type* of terminal involved, and then does the necessary initializations and mode settings. The type of terminal attached to each HP-UX port is specified in the */etc/ttytype* data base. Type names for terminals may be found in the files under the */usr/lib/terminfo* directory (see *terminfo*(4)). If a port is not wired permanently to a specific terminal (not hardwired), it will be given an appropriate generic identifier, such as *dialup*.

In the case where no arguments are specified, *tset* simply reads the terminal type out of the environment variable TERM and re-initializes the terminal. The rest of this manual entry concerns itself with mode and environment initialization, typically done once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script (*.profile* for *sh*(1) users, or *.login* for *csh*(1) users) it is desirable to give information about the type of terminal you will usually use on ports which are not hardwired. These ports are identified in */etc/ttytype* as *dialup* or *plugboard*, etc. To specify what terminal type you usually use on these ports, the **–m** (map) option flag is followed by the appropriate port type identifier, an optional baud rate specification, and the terminal type. (The effect is to "map" from some conditions to a terminal type, that is, to tell *tset*, "If I'm on this kind of port, then I'll probably be on this kind of terminal".) If more than one mapping is specified, the first applicable mapping prevails. A missing port type identifier matches all identifiers. A *baudrate* is specified as with *stty*(1), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate *test* may be any combination of **>**, **=**, **<**, **@ @**, and **!**; **is a synonym for** = and **!** inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **–m** within single quotes; users of *csh*(1) must also put a "\" before any "!" used.

Thus,

tset –m 'dialup>300:2622' –m 'dialup:2624' –m 'plugboard:?2623'

causes the terminal type to be set to an HP 2622 if the port in use is a dialup at a speed greater than 300 baud, or to an HP 2624 if the port is otherwise a dialup (i.e. at 300 baud or less). If the *type* finally determined by *tset* begins with a question mark, the user is asked if he or she really wants that type. A null response means to use that type; otherwise, another type can be entered. Thus, in the above case, if the user is on a plugboard port, he or she will be asked whether or not he or she is actually using an HP 2623.

If no mapping applies and a final *type* option, not preceded by a **–m**, is given on the command line, then that type is used. Otherwise, the identifier found in the */etc/ttytype* data base will be taken to be the terminal type. The latter should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by *tset*, and information about the terminal's capabilities to a shell's environment. This can be done using the **-s** option. Using the Bourne shell (*sh*(1)), the command

eval `tset –s *options...*` or using the C shell, *csh*(1):

set noglob; eval `tset –s *options...*`

These commands cause *tset* to generate as output a sequence of shell commands which place the variable TERM in the environment; see *environ*(5).

Once the terminal type is known, *tset* engages in terminal mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the full line erase or line-kill) characters, and setting special character delays. Tab and new-line expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ("#" on standard systems), the erase character is changed to BACKSPACE (^H).

The options are:

−e*c* sets the erase character to be the named character *c*; *c* defaults to ^H (BACKSPACE). The character *c* can either be typed directly, or entered using the "hat" notation used here (e.g. the "hat" notation for control-H is ^H; in *sh*(1), the ^ character should be escaped (\^)).

−k*c* sets the kill character to *c*. The default *c* is ^X. If *c* is not specified, the kill character will remain unchanged unless the original value of the kill character is null. In this case, the kill character is set to an "at" sign (@).

− report terminal type. Whatever type is decided on is reported. If no other flags are given, the only effect is to write the terminal type on the standard output.

−s generates appropriate commands (depending on your SHELL environment variable) to set TERM.

−I suppresses transmitting terminal initialization strings.

−Q suppresses printing the "Erase set to" and "Kill set to" messages.

−A asks the user for the TERM type.

−S Outputs the strings that would be assigned to TERM in the environment rather than generating commands for a shell. In *sh*(1), the following is an alternate way of setting TERM.

```
set -- `tset −S ...`
TERM=$1
```

−h forces a read of */etc/ttytype*. When **−h** is not specified, the terminal type is determined by reading the environment, unless some mapping is specified.

For compatibility with earlier versions of *tset*, the following flags are accepted, but their use is discouraged:

−r report to the user in addition to other flags.

−E*c* sets the erase character to *c* only if the terminal can backspace. *C* defaults to ^H.

EXAMPLES

These examples all assume the Bourne shell (*sh*(1)). Note that a typical use of *tset* in a *.profile* will also use the **−e** and **−k** options, and often the **−n** or **−Q** options as well. These options have not been included here to keep the examples small.

Assume, for the moment, that you are on an HP 2622. This is suitable for typing by hand but not for a *.profile*, unless you are *always* on a 2622.

```
export TERM; TERM=`tset − 2622`
```

Now, you have an HP 2623 at home which you dial up on, but your office terminal is hardwired and known in */etc/ttytype*.

```
export TERM; TERM=`tset - -m dialup:2623`
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use an HP 2622 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on an HP 2623. Sometimes you use someone else's terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2623. Note the placement of the question mark, and the quotes to protect the > and ? from interpretation by the shell.

```
export TERM; TERM=`tset - -m 'switch>1200:?2622' -m 'switch<=1200:2623'`
```

All of the above entries will fall back on the terminal type specified in */etc/ttytype* if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an HP 2622. It always asks you what kind of terminal you are on, defaulting to 2622.

```
export TERM; TERM=`tset - ?2622`
```

If the file */etc/ttytype* is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
export TERM; TERM=`tset - -m '>1200:2624' 2622`
```

FILES

/etc/ttytype	port name to terminal type mapping data base;
/usr/lib/terminfo/?/*	terminal information data base.

VARIABLES

SHELL	if "csh" then generate *csh*(1) commands, otherwise generate *sh*(1) commands.
TERM	the (canonical) terminal name.

AUTHOR

Tset was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

csh(1), sh(1), stty(1), ttytype(4), environ(5).

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

STANDARDS CONFORMANCE

tsort: SVID2, XPG2, XPG3

NAME

tty, pty – get the name of the terminal

SYNOPSIS

tty [**–s**]

pty [**–s**]

DESCRIPTION

Tty and *pty* print the path name of the user's terminal. The **–s** option inhibits printing of the terminal path name and any diagnostics, allowing one to test only the exit code.

RETURN VALUE

Exit status codes for *tty* are:

2	if invalid options were specified,
1	if the standard input is not a terminal or pseudo-terminal,
0	if the standard input is a terminal or pseudo-terminal.

Exit status codes for *pty* are:

2	if invalid options were specified,
1	if the standard input is not a pseudo-terminal,
0	if the standard input is a pseudo-terminal.

DIAGNOSTICS

not a tty	standard input is not a terminal or pseudo-terminal for *tty*.
not a pty	standard input is not a pseudo-terminal for *pty*.

AUTHOR

Tty was developed by AT&T. *Pty* was developed by HP.

STANDARDS CONFORMANCE

tty: SVID2, XPG2, XPG3

NAME

ul – do underlining

SYNOPSIS

ul [**–t** terminal] [**–i**] [name ...]

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable **TERM**. The **–t** option overrides the terminal kind specified in the environment. The *terminfo*(4) file corresponding to **TERM** is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

The **–i** option causes *ul* to indicate underlining onto by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

FILES

/usr/lib/terminfo/?/* terminal capability files

AUTHOR

Ul was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

man(1), nroff(1).

BUGS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed with four octal digits. The first digit, a zero, specifies that the output is expressed in octal.

Umask is recognized and executed by the shell.

Note that the file creation mask does not affect the set-user-ID, set-group-ID, or "sticky" bits.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2).

STANDARDS CONFORMANCE

umask: SVID2, XPG2, XPG3

NAME

umodem – XMODEM-protocol file transfer program

SYNOPSIS

umodem [–options] files
umodem –c

DESCRIPTION

Umodem is a file transfer program that incorporates the well known XMODEM protocol used on CP/M systems and on the HP110 portable computer.

Options

–1	Employ TERM II FTP 1.
–3	Enable TERM FTP 3 (CP/M UG).
–7	Enable 7-bit transfer mask.
–a	Turn on ARPA Net flag.
–c	Enter command mode.
–d	Do not delete *umodem.log* before starting.
–l	Turn on entry logging.
–m	Allow overwriting of files.
–p	Print all messages.
–r[t b]	Receive file. Specify **t** for text, or **b** for binary.
–s[t b]	Send file. Specify **t** for text, or **b** for binary.
–y	Display file status only.

EXAMPLES

To receive a text file:

umodem -rt7*file*

To receive a binary file:

umodem -rb*file*

To send a text file:

umodem -st7*file*

To send a binary file:

umodem -sb*file*

AUTHOR

Umodem is in the public domain.

SEE ALSO

kermit(1), cu(1), uucp(1).

NAME

uname – print name of current HP-UX version

SYNOPSIS

uname [**–snrvmia**]

uname [**–S** *nodename*]

DESCRIPTION

Uname prints the current system name of the HP-UX system on the standard output file. It is used to determine which system one is using. This system name is also used by *uucp*(1), *mail*(1), and related utilities.

Options

–s	Print the system name (default).
–n	Print the nodename, which can be a name by which the system is known on a communications network such as *uucp*.
–r	Print the operating system release.
–v	Print the operating system version.
–m	Print the machine hardware name.
–i	Print the nodename if the machine identification number cannot be ascertained.
–a	Print all the above information.

The nodename can be changed by specifying *nodename* as an argument to the **–S** option. The *nodename* argument is restricted to UTSLEN-1 characters; UTSLEN is defined in <**sys/utsname.h**>. Only the superuser can use this capability.

The system might be known by other names if networking products are supported. See the node manager's documentation supplied with your system for details.

SEE ALSO

hostname(1), gethostname(2), sethostname(2), uname(2).

STANDARDS CONFORMANCE

uname: SVID2, XPG2, XPG3

NAME

unget − undo a previous get of an SCCS file

SYNOPSIS

unget [−r*SID*] [−**s**] [−**n**] *file* ...

DESCRIPTION

Unget undoes the effect of a **get** −**e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. Refer to *sact*(1), which describes how to determine what deltas are currently binding for an s-file.

Keyletter arguments apply independently to each named file.

−r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line (see *sact*(1)).

−**s** Suppresses the printout, on the standard output, of the intended delta's *SID*.

−**n** Causes the retention of the gotten file which would normally be removed from the current directory.

Note: *unget* can only be executed by the user who did the corresponding **get** −**e**. If a system administrator needs to *unget* a **get** −**e** done by another user, he must either use *su*(1) to change into that user, or edit the p-file directly (which can be done either by the s-file owner or the super-user).

DIAGNOSTICS

Use *help*(1) for explanations.

FILES

p-file See *delta*(1).

g-file See *delta*(1).

SEE ALSO

delta(1), get(1), help(1), sact(1).

SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

STANDARDS CONFORMANCE

unget: SVID2, XPG2, XPG3

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**−udc** [+n] [−n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **−u** flag is used, just the lines that are not repeated in the original file are output. The **−d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **−u** and **−d** mode outputs.

The **−c** option supersedes **−u** and **−d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

RETURN VALUE

Exit values are:

0 Successful completion.

>0 Error condition occured.

SEE ALSO

comm(1), sort(1).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE must be equal to the value it had when the input files were sorted.

LC_CTYPE determines the space character when the **−n** or **+n** option is enabled.

LANG determines the language in which messages are displayed.

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *uniq* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

uniq: SVID2, XPG2, XPG3

NAME

units – conversion program

SYNOPSIS

units [- file]

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, and division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

Units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

An alternate unit database file can be specified for use with the '- file' option. Units will look in this file rather than the default /usr/lib/unittab for the table of conversions. This must be in the same format as /usr/lib/unittab. This is useful in defining your own units and conversions.

WARNINGS

The monetary exchange rates are out of date.

FILES

/usr/lib/unittab

NAME

uptime – show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time, the length of time the system has been up, the number of users logged on to the system, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes.

EXAMPLES

The command:

uptime

might yield the following:

2:30pm up 14days, 2:39, 33 users, load average: 1.71, 1.88, 1.80

depending upon the current status of the system.

AUTHOR

Uptime was developed by the University of California, Berkeley.

NAME

users – compact list of users who are on the system

SYNOPSIS

users [*–c*]

DESCRIPTION

Users lists the login names of the users currently on the system in a compact, one-line format. In the HP Clustered environment, the **–c** option can be used to list the login names of all users in the cluster (see *Glossary*(9)).

The login names are sorted in ascending collation order (see Environment Variables below).

AUTHOR

Users was developed by the University of California, Berkeley and HP.

FILES

/etc/utmp

SEE ALSO

who(1).

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the order in which the output is sorted.

If LC_COLLATE is not specified in the environment or is set to the empty string, the value of LANG is used as a default. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *users* behaves as if all internationalization variables are set to "C" (see *environ*(5)).

NAME

uucp, uulog, uuname – UNIX system to UNIX system copy

SYNOPSIS

uucp [*options*] *source-files destination-file*

uulog **–f** *system* [**–x**] [*–number*]

uulog [**–s** *system*] ... [**–x**] [*–number*]

uuname [**–l**]

DESCRIPTION

Uucp

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file, via the specified route, to the destination. Care should be taken to insure that intermediate nodes in the route are willing to forward information (see WARNINGS below for restrictions).

The shell metacharacters **?**, * and **[...]** appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user's login directory; Note that if an invalid login is specified, the default will be to the public directory (**/usr/spool/uucppublic**);

(3) a path name preceded by ~*/destination*, where *destination* is appended to **/usr/spool/uucppublic**.

NOTE: This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example, ~**/dan/** as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory.

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2) and Access Control Lists below).

Options

The following options are interpreted by *uucp*:

–c Do not copy local file to the spool directory for transfer to the remote machine (default).

–C Force the copy of local files to the spool directory for transfer.

–d Make all necessary directories for the file copy (default).

−f Do not make intermediate directories for the file copy.

−g*grade* *Grade* is a single letter or number. Lower ASCII sequence characters cause the job to be transmitted earlier during a particular conversation.

−j Output the job identification ASCII string on standard output. This job identification can be used by *uustat*(1) to obtain the status or terminate a job.

−m*file* Send mail to the requester when the copy is completed.

−n*user* Notify *user* on the remote system that a file was sent.

−r Do not start the file transfer; just queue the job.

−s*file* Report status of the transfer to *file*. Note that the *file* must be a full path name.

−x*debug_level* Produce debugging on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more information.

Uulog

Uulog queries a log file of *uucp* transactions in a file **/usr/spool/uucp/.Log/uucico/** *system*.

The options cause *uulog* to print logging information:

−s *system* Print information about work involving system.

−f *system* Do a **tail -f** (see *tail*(1)) of the file transfer log for system.

Other options used in conjunction with the above:

−x Search in the **/usr/spool/uucp/.Log/uuxqt/***system* file for the given system, instead of the *uucico* log file.

−number Do a *tail*(1) command of *number* lines.

Uuname

Uuname lists the *uucp* names of known systems. The **−l** option returns the local system name.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

Access Control Lists (ACLs)

A file's optional ACL entries are not preserved across *uucp* transmission. Instead, new files have a summary of the access modes (as returned in *st_mode* by *stat*(2)).

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons, you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~/).

All files received by *uucp* will be owned by *uucp*.

The **−m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **? ∗ [...]** will not activate the **−m** option.

Protected files and files that are in protected directories that are owned by the requester can be sent by *uucp*. However, if the requester is root and the directory is not searchable by "other" or the file is not readable by "other," the request will fail.

DEPENDENCIES

Series 300

The following options to *uucp* are not currently supported: **−s**, **−x**.

The following options to *uulog* are not currently supported: **−f** *system*, **−x**, *−number*.

FILES

/usr/lib/uucp/*	other data and program files
/usr/spool/uucp	spool directories
/usr/spool/uucppublic	public directory for receiving and sending

SEE ALSO

mail(1), uux(1), chmod(2), stat(2), acl(5).

UUCP tutorial in *HP-UX Concepts and Tutorials.*

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings displayed by the *uucp* and *uulog* commands.

LANG determines the language in which messages are displayed by the *uucp* and *uuname* commands.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting,

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported. *uucp*, *uulog*, and *uuname* behave as if all internationalization variables are set to "C".
See *environ*(5).

STANDARDS CONFORMANCE

uucp: SVID2, XPG2, XPG3

uulog: SVID2, XPG2, XPG3

uuname: SVID2, XPG2, XPG3

NAME

uupath, mkuupath − access and manage the pathalias database

SYNOPSIS

uupath [**−f** *pathsfile*] *mailaddress*

mkuupath [**−v**] *pathsfile*

DESCRIPTION

Uupath provides electronic message routing by expanding a simple *uucp*(1) address into a full *uucp*(1) path. For example, *host*!*user* could be expanded into *hostA*!*hostB*!*host*!*user*.

Uupath expands an address by parsing *mailaddress* for the dominant host (see below) and looking up the host in the appropriate *pathalias* database (see *pathalias*(1)). If the host is found in the database, the expanded address is written to the standard out. If the host is not found, *uupath* writes the original address to the standard out and returns an exit status of 1. *Uupath* expects *mailaddress* to be in uucp format (*host*!...!*hostZ*!*user*) or ARPANET format (*user*@*host*).

The **-f** option opens the *pathalias*(1) database based on *pathsfile* rather than the default database based on **/usr/lib/mail/paths**. This database must be a database created by *mkuupath*, consisting of the two files *pathsfile*.**dir** and *pathsfile*.**pag**.

The dominant host is the leftmost uucp host in *mailaddress*. If no uucp host is found (no **!** is in the address), *uupath* assumes that the address is in the simple ARPANET format *user*@*host*. If the address does not match either format, *uupath* writes the original address to the standard out and returns an exit status of 1.

Mkuupath constructs a mail routing database by using the *pathsfile* data file obtained from *pathalias*(1) as input. The recommended *pathsfile* location is **/usr/lib/mail/paths**, because this is the default database used by *uupath*. The database files *pathsfile*.**dir** and *pathsfile*.**pag** are created by *mkuupath*. If these files already exist, they must be removed prior to running *mkuupath*.

The **−v** option specifies verbose mode, which writes a line to the standard out for each entry written to the database.

DIAGNOSTICS

Uupath returns an exit status of 1 and writes the original *mailaddress* to the standard out if the address is not found or is incorrectly formatted. *Uupath* returns an exit status of 2 and prints a diagnostic message if the database files are not accessible or if improper parameters are given. Otherwise, *uupath* returns an exit status of 0.

If the database files *pathsfile*.**dir** and *pathsfile*.**pag** already exist prior to running *mkuupath*, the message "**mkuupath: *pathsfile*.dir: File exists**" is displayed. These files must be removed before running *mkuupath*.

AUTHOR

Uupath was developed by University of California, Berkeley.

FILES

/usr/lib/mail/paths
/usr/lib/mail/paths.dir
/usr/lib/mail/paths.pag

SEE ALSO

pathalias(1), uucp(1).

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [-a]
uustat [-m]
uustat [-p]
uustat [-q]
uustat [-k *jobid***]**
uustat [-r *jobid***]**
uustat [-s *sys***] [-u** *user***]**

DESCRIPTION

Uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

–a Output all jobs in queue.

–m Report the status of accessibility of all machines.

–p Execute a **ps -flp** for all the process-IDs that are in the lock files.

–q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in **()** next to the number of **C** or **X** files it is the age in days of the oldest **C.** or **X.** file for that system. The Retry field is the number of hours until the next possible call. The Count field is the number of failure attempts. Note that for systems with a moderate number of outstanding jobs, this could take 30 seconds or more or realtime to execute. As an example of the output produced by **uustat –q**:

```
eagle  3C 04/07-11:07 NO DEVICES AVAILABLE
mh3bs3 2C 07/07-10:42 SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (a command file with no files to be sent causes the *uucp* system to call the remote system and see if work is waiting). The date and time refer to the previous interaction with the system followed by the status of interaction.

–k *jobid* Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

–r *jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the cleanup daemon.

The options below may not be used with the ones listed above; however, these options may be used singly or together:

–s *sys* Report the status of all *uucp* requests for remote system *sys*.

–u*user* Report the status of all *uucp* requests issued by *user*.

Output for both the **-s** and **-u** options has the following format:

```
eaglen0000   4/07-11:01:03   (POLL)
eagleN1bd7   4/07-11:07      S       eagle dan 522 /usr/dan/A
eagleC1bd8   4/07-11:07      S       eagle dan 59 D.3b2a12cd4924
```

4/07-11:07 S eagle dan rmail mike

With the **−s** and **−u** options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an **S** or **R**, depending on whether the job is to send or request a file. The next field is the destination system name. This is followed by the user-ID of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution the name of the command (**rmail**, which is the command used for remote mail). When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., **D.3b2alce4924**) that is created for data files associated with remote execution (**rmail** in this example).

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user. The format used is the same as with the **-s** or **-u** options.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

DEPENDENCIES

Series 300

The following options are not currently supported: **−a** and **−p**.

The following options are supported. Only one of these can be specified, and only if none of **−k**, **−r** or **−q** is used:

−j *jobid* Report the status of the *uucp* request *jobid*. If **all** is used for *jobid*, the status of all *uucp* requests is reported. An argument must be supplied otherwise the usage message will be printed and the request will fail.

−c*hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.

−m*mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.

−M*mch* This is the same as the *−m* option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.

The following additional options may not be used with ones listed above; however, these options may be used singly or together with each other or with **−s** or **−u**:

−o*hour* Report the status of all *uucp* requests which are older than *hour* hours.

−y*hour* Report the status of all *uucp* requests which are younger than *hour* hours.

−O Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed with each *uucp* request.

The **−q** option cannot be used in combination with the options **−o**, **−y**, **−s**, **−u** or **−O**.

FILES

/usr/spool/uucp/* spool directories

SEE ALSO

uucp(1).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

LANG determines the language in which messages are displayed.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *uustat* behaves as if all internationalization variables are set to "C". See *environ*(5).

NAME

uuto, uupick – public UNIX system to UNIX system file copy

SYNOPSIS

uuto [*options*] *source-files destination*
uupick [**–s** *system*]

DESCRIPTION

Uuto sends *source-files* to *destination*. *Uuto* uses the *uucp*(1) facility to send files while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname* on *uucp*(1)). *User* is the login name of someone on the specified system.

Two *options* are available:

–p Copy the source file into the spool directory immediately, and send the copy.
–m Send mail to the requester when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is the *uucp* public directory (/usr/spool/uucppublic). Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The recipient is notified by *mail*(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the recipient. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] **?**

Uupick then reads a line from the standard input to determine the disposition of the file:

<new-line>	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default). Note that, if the current working directory is desired for *dir*, you should **not** specify any parameter with **m**. A construction like **m.** is erroneous, and results in loss of data.
a [*dir*]	Same as **m** except move all the files sent from *system*.
p	Print the contents of the file.
q	Stop.
EOT (control-d)	Same as **q**.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the **–s***system* option will only search the PUBDIR for files sent from *system*.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

WARNINGS

To send files that begin with a dot (e.g., **.profile**) the files must by qualified with a dot. For example: **.profile**, **.prof***, and **.profil?** are correct, whereas ***prof*** and **?profile** are incorrect.

FILES

PUBDIR /usr/spool/uucppublic public directory

SEE ALSO

mail(1), uuclean(1M), uucp(1), uustat(1), uux(1).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

STANDARDS CONFORMANCE

uuto: SVID2, XPG2, XPG3

uupick: SVID2, XPG2, XPG3

NAME

uux – UNIX system to UNIX system command execution

SYNOPSIS

uux [*options*] *command-string*

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail*(1)) via *uux*.

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of the following:

- A full path name;
- A path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory. Note that if an invalid login is specified, the default will be to the public directory (**/usr/spool/uucppublic**);
- A path name preceded by ~*/destination*, where *destination* is appended to **/usr/spool/uucppublic**.
- A simple file name (which is prefixed by the current directory). See *uucp*(1) for details.

For example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

gets files **file1** and **file2** from the "usg" and "pwba" machines, executes a *diff*(1) command and puts the results in **file.diff** in the local **/usr/spool/uucppublic** directory.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

Uux attempts to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!cut -f1 b!/usr/file \(c!/usr/file\)
```

gets **/usr/file** from system "b" and sends it to system "a", performs a *cut* command on the file and sends the result of the cut command to system "c".

Uux will notify you if the requested command on the remote system was disallowed. The list of commands allowed is specified in the **Permissions** file in **/usr/lib/uucp**. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

–	The standard input to *uux* is made the standard input to the *command-string*.
–a*name*	Use *name* as the user identification replacing the initiator user-id. (Notification is returned to the user.)
–b	Return whatever standard input was provided to the *uux* command if the exit status is non-zero.
–c	Do not copy the local file to the spool directory for transfer to the remote machine (default).

−C	Force the copy of local files to the spool directory for transfer.
−g*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.
−j	Output the *jobid*, which is the job identification ASCII string on the standard output. This job identification can be used by *uustat*(1) to obtain the status or terminate a job.
−n	Do not notify the user if the command fails.
−r	Do not start the file transfer, just queue the job.
−s*file*	Report status of the transfer in *file*.
−x*debug_level*	Produce debugging output on standard output. The *debug_level* is a number between 0 and 9. The higher the numbers, the more detailed information returned.
−z	Send success notification to user.

In the HP Clustered environment, all UUCP activity is handled through device files residing on the cluster server as if the cluster were a single system.

WARNINGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will NOT work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command:

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work (if *diff*(1) is a permitted command).

Protected file and files that are in protected directories that are owned by the requester can be sent in commands using *uux*. However, if the requester is root, and the directory is not searchable by "other", the request will fail.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory
/usr/lib/uucp/*	other data and programs

SEE ALSO

mail(1), uuclean(1M), uucp(1).

UUCP, a tutorial in *HP-UX Concepts and Tutorials.*

STANDARDS CONFORMANCE

uux: SVID2, XPG2, XPG3

NAME

val − validate SCCS file

SYNOPSIS

val −
val [−**s**] [−**r**SID] [−**m**name] [−**y**type] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a −, and named files.

Val has a special argument, −, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

−s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

−rSID The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e.g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

−m*name* The argument value *name* is compared with the SCCS %M% keyword in *file*.

−y*type* The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **−y** mismatch;
bit 7 = %M%, **−m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned − a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a fatal error.

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

STANDARDS CONFORMANCE

val: SVID2, XPG2, XPG3

NAME

vc – version control

SYNOPSIS

vc [**–a**] [**–t**] [**–c**char] [**–s**] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the **–t** keyletter (see below). The default control character is colon (**:**), except as modified by the **–c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The **–a** keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

–a Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.

–t All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.

–c*char* Specifies a control character to be used in place of **:**.

–s Silences warning messages (not errors) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]
Used to declare keywords. All keywords must be declared.

:asg keyword=value
Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition
:
:

:end

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.
The syntax of a condition is:

```
<cond>      ::= [ "not" ] <or>
<or>        ::= <and> | <and> "|" <or>
<and>       ::= <exp> | <exp> "&" <and>
<exp>       ::= "(" <or> ")" | <value> <op> <value>
<op>        ::= "=" | "!=" | "<" | ">"
<value>     ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```
= != > <      all of equal precedence
&
|
```

Parentheses may be used to alter the order of precedence.
Values must be separated from operators or parentheses by at least one blank or tab.

::text

Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **−a** keyletter.

:on

:off

Turn on or off keyword replacement on all lines.

:ctl char

Change the control character to char.

:msg message

Prints the given message on the diagnostic output.

:err message

Prints the given message followed by:

ERROR: err statement on line ... (915)

on the diagnostic output. *Vc* halts execution, and returns an exit code of 1.

SEE ALSO

ed(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 – normal
1 – any error

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of keywords, values, the control character assigned through *ctl* and within text as single- and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *vc* behaves as if all internationalization variables are set to "C". See *environ*(5).

NAME

vi – screen-oriented (visual) display editor

SYNOPSIS

vi [–] [**–v**] [**–r**] [**–R**] [*+command*] [**–l**] [**–t** *tag*] [**–V**] [**–w***size*] [**–x**] [*file ...*]
view [–] [**–v**] [**–r**] [**–R**] [*+command*] [**–l**] [**–t** *tag*] [**–V**] [**–w***size*] [**–x**] [*file ...*]
vedit [–] [**–v**] [**–r**] [**–R**] [*+command*] [**–l**] [**–t** *tag*] [**–V**] [**–w***size*] [**–x**] [*file ...*]

REMARKS

The decryption facilities provided by this software are under the control of the United States Government and cannot be exported without special licenses. These capabilities are only available by special arrangement through HP.

DESCRIPTION

The command *vi* (visual) is a display-oriented text editor. It is based on the underlying line editor *ex*(1). It is possible to switch back and forth between the two, and to execute *ex* commands from within *vi*.

When using *vi*, the terminal screen acts as a window into the file being edited. Changes made to the file are reflected in the screen display; the position of the cursor on the screen indicates the position within the file.

The environment variable TERM must give the terminal type and the terminal must be defined in the *terminfo*(4) database. As with *ex*(1), editor initialization scripts can be placed in the environment variable EXINIT, or in the file **.exrc** in the current or home directory.

The *view* invocation is the same as *vi* except that the **readonly** editor option is set.

The *vedit* invocation is intended for beginners. The **report** editor option is set to **1**, and the **showmode** and **novice** editor options are set. These settings make it easier to get started learning the editor.

Options

The following options are available from the command line:

–	Suppress all interactive user feedback. This is useful in processing editor scripts.
–v	Invoke *vi* (this option is intended for use with *ex*(1) and has no effect on *vi*).
–r	Recover *file*(s) after an editor or system crash. If no *file* is specified, a list of all saved files is printed. You must be owner of the saved file in order to recover it (super-user cannot recover files owned by other users).
–R	Set "read-only" mode to prevent overwriting the file inadvertently.
+command	Begin editing by executing the specified **ex** search or positioning *command*.
–l	*Lisp* mode; indents appropriately for *lisp* code; the **()** {} **[[** and **]]** commands in *vi* are modified to have meaning for *lisp*.
–t *tag*	Edit the file containing the *tag* and position the editor at its definition (see the **tag** command in *ex*(1)).
–V	Verbose mode; editor commands are displayed as they are executed when input from a **.exrc** file or a source file (see the **source** command in *ex*(1)).
–w*size*	Set the value of the **window** editor option to *size*.

-x Encryption mode; the user is prompted for a key to allow for the creation or editing of an encrypted file (see the **crypt** command in *ex*(1)).

See *ex*(1) for the complete description of *ex*. Only the visual mode of the editor is described here.

When invoked, *vi* is in the command mode; the input mode is entered by several commands used to insert or change text. In input mode, ESC (escape) is used to leave input mode; however, two consecutive ESC characters are required to leave input mode if the **doubleescape** editor option is set (see *ex*(1)). In command mode, ESC is used to cancel a partial command; the terminal bell sounds if the editor is not in input mode and there is no partially entered command.

The last (bottom) line of the screen is used to echo the input for search commands (**/** and **?**), *ex*(1) commands (**:**), and system commands (**!**). It is also used to report errors or print other messages.

Receipt of SIGINT during text input or during the input of a command on the bottom line terminates the input (or cancels the command) and returns the editor to command mode. During command mode, SIGINT causes the bell to be sounded; in general the bell indicates an error (such as an unrecognized key).

Lines displayed on the screen containing only a ~ indicate that the last line above them is the last line of the file (the ~ lines are past the end of the file). Terminals with limited local intelligence might display lines on the screen marked with an these indicate space on the screen not corresponding to lines in the file. (These lines may be removed by entering a **^R**, forcing the editor to retype the screen without these holes.)

Command Summary

Most commands accept a preceding number as an argument, either to give a size or position (for display or movement commands), or as a repeat count (for commands that change text). For simplicity, this optional argument is referred to as *count* when its effect is described.

The following operators can be followed by a movement command to specify an extent of text to be affected: **c**, **d**, **y**, **<**, **>**, **!**, and **=**. The region specified begins at the current cursor position and ends just prior to the cursor position indicated by the move. If the command operates on lines only, all the lines that fall partly or wholly within this region are affected. Otherwise the exact marked region is affected.

In the following listing, control characters are indicated in the form "**^X**", which stands for "control-X". Whitespace is defined to be the characters space, tab and alternative space. Alternative space is the first character of the *langinfo*(3C) ALT_PUNCT item for the language specified by the LANG environment variable (see *environ*(5)).

Unless otherwise specified, the commands are interpreted in command mode and have no special effect in input mode.

^B Scroll backward to display the previous window of text. A *count* specifies the number of windows to go back. Two lines of overlap are kept if possible.

^D Scroll forward a half-window of text. A *count* gives the number of (logical) lines to scroll, and is remembered for future **^D** and **^U** commands.

In input mode, **^D** backs **shiftwidth** spaces over the indentation provided by *autoindent* or **^T**.

^E Scroll forward one line, leaving the cursor where it is if possible.

^F Scroll forward to display the window of text following the current one. A *count* specifies the number of windows to advance. Two lines of overlap are kept if possible.

^G	Print the current file name and other information, including the number of lines and the current position. (Equivalent to the *ex* command **f**.)
^H	Move one space to the left (stops at the left margin). A *count* specifies the number of spaces to back up. (Same as **h**.)
	In input mode, **^H** returns the cursor to the last input character without erasing it.
^J	Move the cursor down one line in the same column. A *count* specifies the number of lines to move down. (Same as **^N** and **j**.)
^L	Clear and redraw the screen. (Use when the screen is scrambled for any reason.)
^M	Move to the first non-whitespace character in the next line. A *count* specifies the number of lines to advance.
^N	Same as **^J** and **j**.
^P	Move the cursor up one line in the same column. A *count* specifies the number of lines to move up. (Same as **k**.)
^R	Redraw the current screen, eliminating the false lines marked with **(which do not correspond to actual lines in the file).**
^T	In input mode, if the cursor is at the beginning of the line or is preceded only by white space, **^T** inserts **shiftwidth** white space. This inserted space can only be backed over using **^D**.
^U	Scroll up a half-window of text. A *count* gives the number of (logical) lines to scroll, and is remembered for future **^D** and **^U** commands.
^V	In input mode, **^V** quotes the next character to permit the insertion of special characters (including ESC) into the file.
^W	In input mode, **^W** backs up one word; the deleted characters remain on the display.
^Y	Scroll backward one line, leaving the cursor where it is, if possible.
^[	Cancel a partially formed command; **^[** sounds the bell if there is no partially formed command.
	In input mode, **^[** terminates input mode. However, two consecutive ESC characters are required to terminate input mode if the **doubleescape** editor option is set (see *ex*(1)).
	When entering a command on the bottom line of the screen (*ex* command line or search pattern with \ or **?**), terminate input and execute command.
	On many terminals, **^[** can be entered by pressing the ESC or ESCAPE key.
^	Exit *vi* and enter *ex* command mode. If in input mode, terminate the input first.
^]	Take the word after the cursor as a tag and execute the **tag** editor command (see *ex*(1)).
^^	Return to the previous file (equivalent to **:ex #**).
<space>	Move one space to the right (stops at the end of the line). A *count* specifies the number of spaces to go forward. (Same as **l**.)
erase	Erase, where *erase* is the user-designated erase character (see *stty*(1)). Same as ^H.

kill Kill, where *kill* is the user-designated kill character (see *stty*(1)). In input mode, *kill* backs up to the beginning of the current input line without erasing the line from the screen display.

susp Suspend the editor session and return to the calling shell, where *susp* is the user-designated process-control suspend character (see *stty*(1)). See *ex*(1) for more information on the **suspend** editor command.

! An operator that passes specified lines from the buffer as standard input to the specified system command, and replaces those lines with the standard output from the command. The **!** is followed by a movement command specifying the lines to be passed (lines from the current position to the end of the movement) and then the command (terminated as usual by a return). A *count* preceding the **!** is passed on to the movement command after **!**.

Doubling **!** and preceding it by *count* causes that many lines, starting with the current line, to be passed.

" Use to precede a named buffer specification. There are named buffers **1–9** in which the editor places deleted text. The named buffers **a–z** are available to the user for saving deleted or yanked text; see also **y**, below.

$ Move to the end of the current line. A *count* specifies the number of lines to advance (for example, **2$** causes the cursor to advance to the end of the next line).

% Move to the parenthesis or brace that matches the parenthesis or brace at the current cursor position.

& Same as the *ex* command **&** (that is, **&** repeats the previous **substitute** command).

' When followed by a **'**, *vi* returns to the previous context, placing the cursor at the beginning of the line. (The previous context is set whenever a non-relative move is made.) When followed by a letter **a–z**, returns to the line marked with that letter (see the **m** command), at the first non-whitespace character in the line.

When used with an operator such as **d** to specify an extent of text, the operation takes place over complete lines. (See also **`**.)

`** When followed by a **`, *vi* returns to the previous context, placing the cursor at the character position marked. (The previous context is set whenever a non-relative move is made.) When followed by a letter **a–z**, returns to the line marked with that letter (see the **m** command), at the character position marked.

When used with an operator such as **d** to specify an extent of text, the operation takes place from the exact marked place to the current position within the line. (See also **'**.)

[[Back up to the previous section boundary. A section is defined by the value of the **sections** option. Lines that start with a formfeed (**^L** character) or **{** also stop **[[**.

If the option **lisp** is set, the cursor stops at each **(** at the beginning of a line.

]] Move forward to a section boundary (see **[[**).

^ Move to the first non-whitespace position on the current line.

(Move backward to the beginning of a sentence. A sentence ends at a **.**, **!** or **?** followed by either the end of a line or by two spaces. Any number of closing

),], " and ' characters can appear between the ., ! or ? and the spaces or end of line. If a *count* is specified, the cursor moves back the specified number of sentences.

If the **lisp** option is set, the cursor moves to the beginning of a **lisp** *s*-expression. Sentences also begin at paragraph and section boundaries (see { and **[[**).

) Move forward to the beginning of a sentence. If a *count* is specified, the cursor advances the specified number of sentences. (See "**(**").

{ Move back to the beginning of the preceding paragraph. A paragraph is defined by the value of the **paragraphs** option. A completely empty line and a section boundary (see **[[** above) are also interpreted as the beginning of a paragraph. If a *count* is specified, the cursor moves backward the specified number of paragraphs.

} Move forward to the beginning of the next paragraph. If a *count* is specified, the cursor advances the specified number of paragraphs. (See "**{**".)

| Requires a preceding *count*; the cursor moves to the specified column of the current line (if possible).

+ Move to the first non-whitespace character in the next line. If a *count* is specified, the cursor advances the specified number of lines. (Same as ^**M**.)

, The comma (,) performs the reverse action of the last **f F t** or **T** command issued, by searching in the opposite direction on the current line. If a *count* is specified, the cursor repeats the search the specified number of times.

- The hyphen character (-) moves the cursor to the first non-whitespace character in the previous line. If a *count* is specified, the cursor moves back the specified number of times.

_ The underscore character (_) moves the cursor to the first non-whitespace character in the current line. If a *count* is specified, the cursor advances the specified number of lines, with the current line being counted as the first line; no *count* or a *count* of 1 specifies the current line.

. Repeat the last command that changed the buffer. If a *count* is specified, the command is repeated the specified number of times.

/ Read a string from the last line on the screen, interpret it as a regular expression, and scan forward for the next occurrence of a matching string. The search begins when the user types a carriage return to terminate the pattern; the search can be terminated by sending SIGINT (or the user-designated interrupt character).

When used with an operator to specify an extent of text, the defined region begins with the current cursor position and ends at the beginning of the matched string. Entire lines can be specified by giving an offset from the matched line (by using a closing / followed by a +*n* or -*n*).

0 Move to the first character on the current line. (The **0** is not interpreted as a command when preceded by a non-zero digit.)

: The colon character (:) begins an *ex* command. The : and the entered command are echoed on the bottom line; the *ex* command is executed when the user types a carriage return.

; Repeat the last single character find using **f F t** or **T**. If a *count* is specified, the search is repeated the specified number of times.

< An operator that shifts lines to the left by one **shiftwidth**. The < can be followed by a move to specify lines. A *count* is passed through to the move command.

When repeated (<<), shifts the current line (or *count* lines starting at the current one).

> An operator that shifts lines right one **shiftwidth**. (See <.)

= If the **lisp** option is set, the = reindents the specified lines, as though they were typed in with **lisp** and **autoindent** set. Can be preceded by a *count* to indicate how many lines to process, or followed by a move command for the same purpose.

? Scan backwards, the reverse of /. (See /.)

Execute the commands stored in the named *buffer*. Be careful not to include a <return> character at the end of the buffer contents unless the <return> is part of the command stream. Commands to be executed in *ex* mode should be preceded by a colon (:).

~ The tilde (~) switches the case of the character under the cursor (if it is a letter) and then moves one character to the right, stopping at the end of the line). A *count* specifies how many characters from the current line are switched.

A Append at the end of line. (Same as **$a**.)

B Back up a word, where a word is any non-blank sequence, placing the cursor at the beginning of the word. If a *count* is specified, the cursor moves back the specified number of words.

C Change the rest of the text on the current line. (Same as **c$**.)

D Delete the rest of the text on the current line. (Same as **d$**.)

E Move forward to the end of a word, where a word is any non-blank sequence. If a *count* is specified, the cursor advances the specified number of words.

F Must be followed by a single character; scans backwards in the current line searching for that character and moving the cursor to it, if found. If a *count* is specified, the search is repeated the specified number of times.

G Go to the line number given as preceding argument, or the end of the file if no preceding *count* is given.

H Move the cursor to the top line on the screen. If a *count* is given, the cursor moves to *count* number of lines from the top of the screen. The cursor is placed on the first non-whitespace character on the line. If used as the target of an operator, entire lines are affected.

I Insert at the beginning of a line. (Same as ^ followed by **i**.)

J Join the current line with the next one, supplying appropriate white space: one space between words, two spaces after a period, and no spaces at all if the first character of the next line is a closing parenthesis (")"). A *count* causes the specified number of lines to be joined, instead of two.

L Move the cursor to the first non-whitespace character of the last line on the screen. If a *count* is given, the cursor moves to *count* number of lines from the bottom of the screen. When used with an operator, entire lines are affected.

M Move the cursor to the middle line on the screen, at the first non-whitespace position on the line.

N Scan for the next match of the last pattern given to **/** or **?**, but in the opposite direction; this is the reverse of **n**.

O Open a new line above the current line and enter input mode.

P Put back (replace) the last deleted or yanked text before/above the cursor. Entire lines of text are returned above the cursor if entire lines were deleted or yanked. Otherwise, the text is inserted just before the cursor.

The **P** can be preceded by a named buffer specification (**"***x*), to retrieve the contents of the buffer.

Q Exit *vi* and enter *ex* command mode.

R Replace characters on the screen with characters entered, until the input is terminated with ESC.

S Change entire lines (same as **cc**). A *count* changes the specified number of lines.

T Must be followed by a single character; scan backwards in the current line for that character, and if found, place the cursor just after that character. A *count* is equivalent to repeating the search the specified number of times.

U Restore the current line to its state before the cursor was last moved to it.

W Move forward to the beginning of a word in the current line, where a word is a sequence of non-blank characters. A *count* specifies the number of words to advance.

X Delete the character before the cursor. A *count* repeats the effect, but only characters on the current line are deleted.

Y Place (yank) a copy of the current line into the unnamed buffer (same as **yy**). If a *count* is specified, *count* lines are copied to the buffer. If the **Y** is preceded by a buffer name, the lines are copied to the named buffer.

ZZ Exit the editor, writing out the buffer if it was changed since the last write. (Same as the *ex* command **x**.)

a Enter input mode, appending the entered text after the current cursor position. A *count* causes the inserted text to be replicated the specified number of times, but only if the inserted text is all on one line.

b Back up to the previous beginning of a word in the current line. A word is a sequence of alphanumerics or a sequence of special characters. A *count* repeats the effect.

c Must be followed by a movement command. Delete the specified region of text, and enter input mode to replace deleted text with new text. If more than part of a single line is affected, the deleted text is saved in the numeric buffers. If only part of the current line is affected, the last character deleted is marked with a **$**. A *count* is passed through to the move command. If the command is **cc**, the entire current line is changed.

d Must be followed by a movement command. Delete the specified region of text. If more than part of a line is affected, the text is saved in the numeric buffers. A *count* is passed through to the move command. If the command is **dd**, the entire current line is deleted.

e Move forward to the end of the next word, defined as for **b**. A *count* repeats the effect.

f Must be followed by a single character; scan the rest of the current line for that character, and moves the cursor to it if found. A *count* repeats the action that many times.

h Move the cursor one character to the left. (Same as ^**H**.) A *count* repeats the effect.

i Enter input mode, inserting the entered text before the cursor. (See **a**.)

j Move the cursor one line down in the same column. (Same as ^**J** and ^**N**.)

k Move the cursor one line up. (Same as ^**P**.)

l Move the cursor one character to the right. (Same as <**space**>.)

m Must be followed by a single lowercase ASCII letter **x**; mark the current position of the cursor with that letter. The exact position on the marked line is referred to by `*x*; the marked line is referred to by ´*x*.

n Repeat the last **/** or **?** scanning commands.

o Open a line below the current line and enter input mode; otherwise like **O**.

p Put text after/below the cursor; otherwise like **P**.

r Must be followed by a single character; the character under the cursor is replaced by the specified one. (The new character can be a new-line.) If **r** is preceded by a *count*, *count* characters are replaced by the specified character.

s Delete the single character under the cursor and enter input mode; the entered text replaces the deleted character. A preceding *count* specifies how many characters on the current line are changed. The last character being changed is marked with a **$**, as for **c**.

t Must be followed by a single character; scan the remainder of the line for that character. The cursor moves to the column prior to the character, if the character is found. A *count* is equivalent to repeating the search *count* times.

u Reverse the last change made to the current buffer. If repeated, **u** alternates between these two states; thus is its own inverse. When used after an insertion of text on more than one line, the lines are saved in the numerically named buffers.

w Move forward to the beginning of the next word (where word is defined as in **b**). A *count* specifies how many words the cursor advances.

x Delete the single character under the cursor. When **x** is preceded by a *count*, **x** deletes the specified number of characters forward from the cursor position, but only on the current line.

y Must be followed by a movement command; the specified text is copied (yanked) into the unnamed temporary buffer. If preceded by a named buffer specification, "*x*, the text is placed in that buffer also. If the command is **yy**, the entire current line is yanked.

z Redraw the screen with the current line placed as specified by the following options: **z**<return> specifies the top of the screen, **z.** the center of the screen, and **z–** the bottom of the screen. The commands **z^** and **z+** are similar to ^**B** and ^**F**, respectively; however, **z^** and **z+** do not attempt to maintain two lines of overlap. A *count* can be given after the **z** and before the following character to specify the number of lines displayed in the redrawn screen. A *count* before the **z** gives the number of the line to use as the reference line instead of the default current line.

Keyboard Editing Keys

At initialization, the editor automatically maps some terminal keyboard editing keys to equivalent visual mode commands. These mappings are only established for keys that are listed in the following table and are defined in the *terminfo*(4) database as valid for the current terminal (as specified by the TERM environment variable).

Both command and input mode mappings are created (see the **map** command in *ex*(1)). With the exception of the **insert char** keys, which simply toggle input mode on and off, the input mode mappings exit input mode, perform the same action as the command mode mapping, and then reenter input mode.

On certain terminals, the character sequence sent by a keyboard editing key, which is then mapped to a visual mode command, can be the same character sequence a user might enter to perform another command or set of commands. This is most likely to happen with the input mode mappings; therefore, on these terminals the input mode mappings are disabled by default. Users can override the disabling and enabling of both the command and input mode keyboard editing key mappings by setting the **keyboardedit** and **keyboardedit!** editor options as appropriate (see *ex*(1)). The **timeout**, **timeoutlen** and **doubleescape** editor options are alternative methods of addressing this problem.

terminfo entry	command mode map	input mode map	map name	description
key_ic	i	^[	inschar	insert char
key_eic	i	^[	inschar	end insert char
key_up	k	^[ka	up	arrow up
key_down	j	^[ja	down	arrow down
key_left	h	^[ha	left	arrow left
key_right	l	^[la	right	arrow right
key_home	H	^[Ha	home	arrow home
key_il	o^[	^[o^[a	insline	insert line
key_dl	dd	^[dda	delline	delete line
key_clear	^L	^[^La	clear	clear screen
key_eol	d$	^[d$a	clreol	clear line
key_sf	^E	^[^Ea	scrollf	scroll down
key_dc	x	^[xa	delchar	delete char
key_npage	^F	^[^Fa	npage	next page
key_ppage	^B	^[^Ba	ppage	previous page
key_sr	^Y	^[^Ya	sr	scroll up
key_eos	dG	^[dGa	clreos	clear to end of screen

EXTERNAL INFLUENCES

Environment Variables

LC_COLLATE determines the collating sequence used in evaluating regular expressions and in processing the **tags** file.

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, the classification of characters as upper- or lower-case letters, the shifting of the case of letters, and the characters matched by character class expressions in regular expressions.

LANG determines the language in which messages are displayed.

LANGOPTS specifies options determining how text for right-to-left languages is stored in input and output files. See *environ*(5).

If LC_COLLATE or LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not

specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, the editor behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

WARNINGS

See warnings in ex(1).

AUTHOR

Vi was developed by the University of California, Berkeley. The 16-bit extensions to *vi* are based in part on software of the Toshiba Corporation.

SEE ALSO

ex(1), edit(1), terminfo(4), environ(5), lang(5), regexp(5).

The *vi/ex Editor* tutorial in the *Text Editors and Processors* volume of *HP-UX Concepts and Tutorials*.

STANDARDS CONFORMANCE

vi: SVID2, XPG2, XPG3

NAME

vis, inv – make unprintable characters in a file visible or invisible

SYNOPSIS

vis [**–n**] [**–s**] [**–t**] [**–u**] [**–x**] file ...
inv [**–n**] [**–s**] [**–t**] [**–u**] [**–x**] file ...

DESCRIPTION

Vis reads characters from each *file* in sequence and writes them to the standard output, converting those which are not printable into a visible form. *Inv* performs the inverse function, reading printable characters from each *file* and writing them, returned if appropriate to non-printable form, to standard out.

Non-printable characters are represented using C-like escape conventions:

\\	backslash
\b	backspace
\e	escape
\f	form-feed
\n	new-line
\r	carriage return
\s	space
\t	horizontal tab
\v	vertical tab
n	the 8-bit character whose ASCII code is the 3-digit octal number *n*.
\x*n*	the 8-bit character whose ASCII code is the 2-digit hexadecimal number *n*.

Space, horizontal tab, and new line may be treated as printable (and therefore passed unscathed to the output) or non-printable dependent on the options selected. Backslash, although printable, is expanded by *vis*, to a pair of backslashes so that when passed back through *inv*, it can be mapped back to a single backslash.

If no input file is given, or if the argument – is encountered, *vis* and *inv* read from the standard input file.

The options are:

–n causes new-line, space, and horizontal tab to be treated as non-printable characters. Thus *vis* expands them visibly as **\n**, **\s**, and **\t**, rather than passing them directly to the output. *Inv* discards these characters, expecting only the printable expansions. New-line characters are inserted by *vis* every 16 characters so that the output will be in form acceptable for most editors.

–s makes *vis* and *inv* silent about non-existent files, identical input and output, and write errors. Normally, no input file may be the same as the output file unless it is a special file.

–t treats horizontal tab and space as non-printable characters, in the same manner in which **–n** options treats them.

–u causes output to be unbuffered (character-by-character); normally, output is buffered.

–x causes *vis* output to be in hexadecimal form rather than the default octal form. Either form is accepted to *inv* as input.

AUTHOR

Vis was developed by the Hewlett-Packard Company.

SEE ALSO

cat(1), echo(1), od(1).

WARNING

Command formats such as

```
vis file1 file2 >file1
```

will cause the original data in file1 to be lost.

EXTERNAL INFLUENCES

Environment Variables

LANG determines the language in which messages are displayed.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

vmstat – report virtual memory statistics

SYNOPSIS

vmstat [**–dfsSz**] [interval [count]]

DESCRIPTION

Vmstat normally reports certain statistics kept about process, virtual memory, trap and cpu activity. If given a **–d** argument, it also reports disk transfer information in the form of transfers per second. If given a **–f** argument, it instead reports on the number of *forks* and *vforks* since system startup and the number of pages of virtual memory involved in each kind of fork. If given a **–s** argument, it instead prints the contents of the *sum* structure, giving the total number of several kinds of paging related events that have occurred since boot. Giving a **–S** argument causes vmstat to execute normally with the exception that the number of processes swapped in and out are displayed instead of page reclaims and address translation faults. The **–z** option requires super-user capabilities. It clears all accumulators in the sum structure.

If none of these options are given, *vmstat* will report in the first line a summary of the virtual memory activity since the system has been booted. If *interval* is specified, then successive lines are summaries over the last *interval* seconds. The command **vmstat 5** will print what the system is doing every five seconds. This is a good choice of printing interval since this is how often some of the statistics are sampled in the system; others vary every second. If a *count* is given, the statistics are repeated *count* times. The format fields are:

Procs: information about numbers of processes in various states.

- r — in run queue
- b — blocked for resources (i/o, paging, etc.)
- w — runnable or short sleeper (< 20 secs) but swapped

Memory: information about the usage of virtual and real memory. Virtual pages are considered active if they belong to processes that are running or have run in the last 20 seconds.

- avm — active virtual pages
- free — size of the free list

Page: information about page faults and paging activity. These are averaged each five seconds, and given in units per second.

- re — page reclaims
- at — address translation faults
- pi — pages paged in
- po — pages paged out
- fr — pages freed per second
- de — anticipated short term memory shortfall
- sr — pages scanned by clock algorithm, per-second

Faults: trap/interrupt rate averages per second over last 5 seconds.

- in — (non clock) device interrupts per second
- sy — system calls per second
- cs — cpu context switch rate (switches/sec)

Cpu:	breakdown of percentage usage of CPU time	
	us	user time for normal and low priority processes
	sy	system time
	id	cpu idle

AUTHOR

Vmstat was developed by the University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/dev/kmem
/hp-ux

NAME

vt – log in on another system over lan

SYNOPSIS

vt *nodename* [*lan device*]
vt **–p** [*lan device*]

DESCRIPTION

vt enables a user to log in on another HP 9000 system (*nodename*) over an HP local area network. The **–p** option will cause *vt* to send a poll request over the local area network to find out what systems currently have *vtdaemon(1M)* running. An asterisk (*) following a nodename in the response indicates that the system is a vt gateway. Plus signs (+) following the nodename indicate how many vt gateways have to be traversed to reach that system.

The optional argument *lan device* specifies a character special device name to use instead of the default device name to send/receive data to/from the local area network. The major number for this device must correspond to a CIO IEEE802.3 local area network device.

Once a connection has been established, *vt* enters input mode. In this mode, text typed is sent to the remote host. To issue *vt* commands when in input mode, precede them with the *vt* escape character. When in command mode, the normal terminal editing conventions are available.

The connection should automatically be terminated upon logging off of the remote machine. If the connection is not terminated then this indicates that the *ptydaemon* on the remote system has either been terminated or restarted. In this case the user should enter command mode and use the **quit** command to terminate the connection.

Commands

The following commands are available. Only enough of each command to uniquely identify it need be typed.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory.* This command is applicable for file transfer only.

escape [*escape-char*]
Set the *vt* escape character. If a character is not specified *vt* will prompt for one. If current *vt* escape character will be printed if the user just hits return in response to this prompt.

help

? Print a *vt* command summary.

lcd [*directory*]
Change the local working directory. If no *directory* is specified, use the user's home directory. This command is applicable for file transfer and shell escapes only.

get *remote-file local-file*

receive *remote-file local-file*
Retrieve the *remote-file* and store it on the local machine as *local-file. vt* will prompt for the file names if they are not specified. The file transfer can be aborted by typing the interrupt character or hitting the break key.

put *local-file remote-file*

send *local-file remote-file*
Retrieve the *local-file* and store it on the remote machine as *remote-file. vt* will prompt for the file names if they are not specified. The file transfer can be aborted by typing the

interrupt character or hitting the break key.

quit Terminate the connection and exit *vt*.

user *user-name*[:[*password*]]
Identify yourself to the remote *vt* server. *vt* will prompt for a password (after disabling local echo) if a colon (:) is appended to *user-name*. It is necessary to do this before any file transfer command can be used.

! [*shellcommand*]
Shell escape. The given command is given to a sub-shell to execute. If no command is given, then a shell is started and connected to the user's terminal.

Access Control Lists (ACLs)
When sending or receiving files using *vt*, optional ACL entries are removed. New files have a summary of the access modes (as returned in *st_mode* by *stat*(2)) of the file being transferred.

DIAGNOSTICS
The diagnostics produced by *vt* are intended to be self-explanatory.

WARNINGS
vt uses the Hewlett-Packard **LLA** (Link Level Access) direct interface to the HP network drivers. *vt* uses the multicast address **0x01AABBCCBBAA.** It should not be used or deleted by other applications accessing the network. *vt* uses the following **IEEE 802.3** *sap* (service access point) values: **0x90, 0x94, 0x98, 0x9C, 0xA0, 0xA4, 0xA8, 0xAC, 0xB0, 0xB4, 0xB8, 0xBC, 0xC0, 0xC4, 0xC8, 0xCC, 0xD0 and 0xD4**. They should not be used by other applications accessing the network.

FILES
/dev/ieee Default lan device name.

SEE ALSO
vtdaemon(1M), stat(2), lan(4), acl(5).

The *vt* reference in *HP-UX Concepts and Tutorials: Shells and Miscellaneous Tools.*

NAME

wait – await completion of process

SYNOPSIS

wait [*pid*]

DESCRIPTION

With no argument, *wait* waits until all processes (started with **&**) of the current shell have completed and reports on abnormal terminations. If a numeric argument *pid* is given and is the process ID of a background process, then *wait* waits until that process has completed. Otherwise, if *pid* is not a background process, *wait* exits without waiting for any processes to complete.

Because the *wait*(2) system call must be executed in the parent process, the shell itself executes *wait* without creating a new process.

SEE ALSO

sh(1), wait(2)

BUGS

Not all the processes of a 2-or-more-stage pipeline are children of the shell, and thus cannot be waited for.

STANDARDS CONFORMANCE

wait: SVID2, XPG2, XPG3

NAME

wc – word, line, and character count

SYNOPSIS

wc [**–lwc**] [names]

DESCRIPTION

Wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **–lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

EXAMPLES

The command:

wc -w *file1*

prints the number of words in *file1*.

The following is printed when the above command is executed:

n file1

where *n* is the number of words in *file1*.

BUGS

Wc counts the number of new-lines to determine the line count. If an ASCII text file has a final line that is not terminated with a new-line character, the count will be off by one.

If there are very many characters, words, and/or lines in an input file, the output may be hard to read. This is because *wc* reserves a fixed column width for each count.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the range of graphics and space characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *wc* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

wc: SVID2, XPG2, XPG3

NAME

wdedit – edit Native Language I/O word dictionary

SYNOPSIS

wdedit *file*

DESCRIPTION

Wdedit is an editor for the word dictionary, which is used for phrase or word conversion by Native Language I/O. *Wdedit* modifies the word dictionary to establish or change the correspondence between designations and words. A designation may be connected to several words. By attaching the word dictionary to Native Language I/O using *nlioenv*(1), designations can be translated into one of their corresponding words via host input conversion.

Wdedit must be invoked on terminals with Native Language I/O enabled. If *file* does not exist, it is created and initialized with the default capacity defined in *wdutil*(1).

To create a new designation or modify the words that are attached to an existing one, use the **/** command, then attach or modify corresponding words by using the edit commands below.

Command Summary

Cursor motion commands:

CR,^**M**	next line, first column
j,^**J**,^**N**	next line, same column
k,^**P**	previous line, same column
h,^**H**	backward a character
l,**space**	forward a character

Screen manipulation commands:

^**F**	forward screen
^**B**	backward screen
^**L**	clear and redraw the screen

Editing commands:

a	append after cursor
i	insert before cursor
o	open line below
O	open line above
R	replace characters
x	delete a character on the cursor
dd	delete the current line
J	join the next line into the current line

Text input is terminated by pressing the ESC key.

The dictionary file is modified after each command.

Exit command:

ZZ	save changes and exit the editor

Other commands:

/desig	enter the designation to be added or modified
!*cmd*	run the shell command *cmd*
:p [*desig*]	print all designations, or a particular designation
:sd [[+|-]*val*]	change the capacity of the data block to *val* as in *wdutil*(1)
:sk [[+|-]*val*]	change the capacity of the key block to *val* as in *wdutil*(1)

The designation, *desig*, can consist of up to eight 8-bit characters, and the word can consist of up to ten 16-bit characters.

WARNINGS

Since the dictionary file is modified as each command is executed, it is impossible to quit *wdedit* without specified changes affecting the dictionary file.

AUTHOR

Wdedit was developed by HP.

SEE ALSO

nlsgenv(1), wdutil(1)

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

wdutil – manipulate Native Language I/O word dictionary

SYNOPSIS

wdutil [**−c** | **−i** [*kcap*][,*dcap*] | **−j** *jfile*] *file*
wdutil [**−pd** [*desig*] | **−pk** [*desig*]] *file*
wdutil [**−sd** [[+ | -]*val*] | **−sk** [[+ | -]*val*]] *file*
wdutil [**−ud** | **−uk** | **−ut**] *file*

DESCRIPTION

Wdutil is used to manipulate the word dictionary used for phrase or word conversion of Native Language I/O. The word dictionary consists of a key block and a data block, which hold the designations and words corresponding to them, respectively.

Wdutil recognizes one of the options below. If no option is specified, the capacity of the key and data blocks in *file* are displayed (if *file* is a valid word dictionary). Otherwise, an error message is printed.

The capacity of key block represents the maximum number of designations, and the capacity of data block represents the maximum number of words.

Options

−c Condense the data block in *file* to obtain a larger contiguous free area.

−i[*kcap*][,*dcap*] Initialize *file* as the word dictionary with the capacities specified by *kcap* and *dcap*. If *file* does not exist, it is created. If *kcap* or ,*dcap* is omitted, the default values are **499** for *kcap* and **650** for *dcap*.

−j*jfile* Join the dictionary *jfile* into *file*. The capacity of the resulting *file* is the sum of the capacities of the original *file* and *jfile*.

−pk[*desig*] Display the designations in the order of their code value. If *desig* ends with *, designations starting with *desig* are printed. If *desig* is omitted or is *, all designations in *file* are printed.

−pd[*desig*] Display the designations and corresponding words. The string *desig* has the same connotations as in **−pk**.

−sd[[+|-]*val*] Change the capacity of the data block in *file*. If + or - precedes *val*, the current value is incremented or decremented by *val*. Otherwise, the capacity is changed to *val*. If *val* is omitted, the default value is **650**.

−sk[[+|-]*val*] Change the capacity of the key block in *file*. The number *val* is evaluated the same as with the **−sd** option, except that the default value is **499**.

−ud Display the capacity and usage of the data block in the number of designations.

−uk Display the capacity and usage of the key block in the number of words.

−ut Display the capacity and usage of both the key and data blocks.

WARNINGS

The smallest prime number not smaller than the given value is used as the capacity of a key block. However, if the given value is smaller than **5**, **5** is used.

AUTHOR

Wdutil was developed by HP.

SEE ALSO

wdedit(1).

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

what − get SCCS identification information

SYNOPSIS

what [−**s**] *file* ...

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is **@(#)** at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:        identification information
f.o:        identification information
a.out:      identification information
```

What is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

−s Quit after finding the first occurrence of pattern in each file.

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use *help*(1) for explanations.

WARNINGS

It is possible that an unintended occurrence of the pattern **@ @(#)** could be found just by chance, but this causes no harm in nearly all cases.

SEE ALSO

get(1), help(1).

SCCS User's Guide, in *HP-UX Concepts and Tutorials: Programming Environment.*

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the interpretation of the pattern substituted for **%Z%** as single and/or multi-byte characters.

LANG determines the language in which messages are displayed.

If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *what* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported with the exception that multi-byte-character file names are not supported.

STANDARDS CONFORMANCE

what: SVID2, XPG2, XPG3

NAME

whereis − locate source, binary, and/or manual for program

SYNOPSIS

whereis [**−bsm**] [**−u**] [**−BMS** *dir* ... **−f**] *name* ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading path name components and any (single) trailing extension of the form ".ext", such as ".c". Prefixes of "s." resulting from use of SCCS are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places. If any of the **−b, −s** or **−m** flags are given, *whereis* searches only for binaries, sources or manual sections, respectively (or any two thereof). The **−u** flag can be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u *" asks for those files in the current directory that have no documentation.

Finally, the **−B**, **−M** and **−S** flags can be used to change or otherwise limit the places where *whereis* searches. The **−f** file flag is used to terminate the last such directory list and signal the start of file names.

EXAMPLES

The following finds all the files in /usr/bin that are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/bin
whereis −u −M /usr/man/man1 −S /usr/src/cmd −f *
```

WARNINGS

Since the program uses *chdir*(2) to run faster, path names given with the **−B**, **−M** and **−S** flags must be absolute path names.

AUTHOR

Whereis was developed by the University of California, Berkeley.

FILES

/usr/src/*
/bin, /etc, /lib, /usr/{bin, games, lib}
/usr/man/*
/usr/local/{man/*, bin, games, include, lib}
/usr/contrib/{man/*, bin, games, include, lib}
/usr/man/$LANG/*
/usr/local/man/$LANG/*
/usr/contrib/man/$LANG/*

EXTERNAL INFLUENCES

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

which – locate a program file including aliases and paths

SYNOPSIS

which [*name* ...]

DESCRIPTION

For each *name* given, *which* searches for the file that would be executed if *name* were given as a command, and displays the absolute path of that file. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are determined by sourcing (executing) the user's **.cshrc** file.

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

EXAMPLES

The command:

which sh

specifies where the executable program of the *sh*(1) command is found. For example, the response might be:

/bin/sh

if the *sh*(1) being used is located in **/bin**.

WARNINGS

Which reports **.cshrc** aliases even when not invoked from *csh*.

Which will not find *csh* built-in commands (e.g. jobs).

Which's information may be incorrect since it is unaware of any path or alias changes that have occurred in the current shell session.

AUTHOR

Which was developed by the University of California, Berkeley.

FILES

~/.cshrc source of aliases and path values

NAME

who − who is on the system

SYNOPSIS

who [**−uTlHqpdbrtasAc**] [*file*]

who am i

who am I

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current system user. It examines the **/etc/utmp** file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

Who with the **am i** or **am I** option identifies the invoking user.

Except for the default **−s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process.

Options

−u This option lists only those users who are currently logged in. *name* is the user's login name. *line* is the name of the line as found in directory **/dev**. The *time* field indicates when the user logged in.

activity is the number of hours and minutes since activity last occurred on that particular line. A dot (**.**) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's login process. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

−T This option is the same as the **−u** option, except that the *state* of the terminal line is printed. *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a − appears if it is not. **Root** can write to all lines having a + or a − in the *state* field. If a bad line is encountered, a **?** is printed.

−l This option lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

−H This option prints column headings above the regular output.

−q This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

−p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab**

that spawned this process. See *inittab*(4).

−d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values of the dead process (as returned by *wait*(2)). This can be useful in determining why a process terminated.

−b This option indicates the time and date of the last reboot.

−r This option indicates the current *run-level* of the *init* process. The last three fields contain the current state of *init*, the number of times that state has been previously entered, and the previous state. These fields are updated each time *init* changes to a different run state.

−t This option indicates the last change to the system clock (via the *date*(1) command) by **root**. See *su*(1).

−a This option processes **/etc/utmp** or the named *file* with all options turned on.

−s This option is the default and lists only the *name*, *line*, and *time* fields.

−A When the **/etc/wtmp** file is specified, this option indicates when the accounting system was turned on or off using the *startup* or *shutacct* commands on *acctsh*(1M). The *name* field is ".". The *line* field is "acctg on", "acctg off", or a reason that was given as an option to the *shutacct* command. The *time* is the time that the on/off activity occurred.

−c This option displays information about an entire HP Cluster. If *file* is given and is context dependent (see *cdf*(4)), data from all elements of the CDF are displayed. If *file* is given and is not a CDF, the **−c** option has no effect.

EXAMPLES

To check who is logged onto the system, type:

who

To check whether or not you can write to the terminal that another user is using, type:

who -T

and look for a plus (+) after the user ID.

AUTHOR

Who was developed by AT&T and HP.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp

SEE ALSO

date(1), login(1), init(1), mesg(1), su(1), wait(2), cdf(4), inittab(4), utmp(4).

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *who* behaves as if all internationalization variables are set to "C". See *environ*(5).

Environment Variables

LC_TIME determines the format and contents of date and time strings.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *who* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single-byte character code sets are supported.

STANDARDS CONFORMANCE

who: SVID2, XPG2, XPG3

NAME

whoami – print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES

/etc/passwd Name data base

AUTHOR

Whoami was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

who (1).

NAME

write – interactively write (talk) to another user

SYNOPSIS

write *user* [*line*]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* **(tty??)** [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should *write* back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg*(1) command. Writing to others is normally allowed by default. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has superuser permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

EXAMPLES

By issuing the command:

write matthew

linda sends a message to **matthew**'s screen. If **matthew** types **write linda**, two-way communication between **matthew** and **linda** is established.

FILES

/etc/utmp	to find user
/bin/sh	to execute **!**

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), who(1).

DIAGNOSTICS

user is not logged on	The person you are trying to *write* to is not logged on.
Permission denied	The person you are trying to *write* to denies that permission (with *mesg*).

Warning: cannot respond, set mesg -y
Your terminal is set to *mesg n* and the recipient cannot respond to you.

Can no longer write to user The recipient has denied permission (*mesg n*) after you had started writing.

EXTERNAL INFLUENCES

Environment Variables

LC_TIME determines the format and contents of date and time strings.

If LC_TIME is not specified in the environment or is set to the empty string, the value of LANG is used as a default for each unspecified or empty variable. If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used instead of LANG. If any internationalization variable contains an invalid setting, *write* behaves as if all internationalization variables are set to "C". See *environ*(5).

International Code Set Support

Single- and multi-byte character code sets are supported.

STANDARDS CONFORMANCE

write: SVID2, XPG2, XPG3

NAME

xargs − construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **−i** flag). Flags **−i**, **−l**, and **−n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **−l** vs. **−n**), the last flag has precedence. *Flag* values are:

−l*number* — *Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **−x** is forced.

−i*replstr* — Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **−x** is also forced. {} is assumed for *replstr* if not specified.

−n*number* — Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **−x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

−t — Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

−p — Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**−t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

−x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.

−s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **−s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

−e*eofstr* *Eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **−e** is not coded. The value **−e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

Xargs will terminate if either it receives a return code of **−1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **−1**.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs −i −t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1. ls | xargs −p −l ar r arch
2. ls | xargs −p −l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs −n2 diff
```

SEE ALSO

sh(1).

DIAGNOSTICS

Self-explanatory.

STANDARDS CONFORMANCE

xargs: SVID2, XPG2, XPG3

NAME

xdb – C, FORTRAN, and Pascal Symbolic Debugger

SYNOPSIS

xdb [**–d** *dir*] [**–r** *file*] [**–p** *file*] [**–P** *process ID*] [**–L**] [**–i** *file*] [**–o** *file*] [**–e** *file*] [**–S** *num*] [*objectfile* [*corefile*]]

TABLE OF CONTENTS

DESCRIPTION

Xdb is a source level debugger for C, HP FORTRAN, and HP Pascal programs. It provides a controlled environment for their execution.

Objectfile is an executable program file with zero or more of its component modules compiled with debug options turned on (that is, enabled by the **–g** flag of *cc*(1), *fc*(1), or *pc*(1)). The support module (**/usr/lib/end.o** for the series 300, **/usr/lib/xdbend.o** for the series 800) must be included as the last object file linked, except for libraries included with the **–l** option to *ld*(1). The default for *objectfile* is **a.out**.

Corefile is a core image from a failed execution of *objectfile*. The default for *corefile* is **core**.

Options

−d *dir* Specify an alternate directory for source files. Alternate directories are searched in the order given. If a source file is not found in any alternate directory, the current directory is searched last. When searching for the source file <*file*> in an alternate directory <*altdir*>, *xdb* first attempts to open <*altdir*>/<*dirname*>/<*basename*>. If this fails, *xdb* attempts to open <*altdir*>/<*basename*> (see *basename*(1)).

−r *file* Specify a record *file*, which is invoked immediately for overwrite, but not for append (see Record and Playback Commands below).

−p *file* Specify a playback *file*, which is invoked immediately (see Record and Playback Commands below).

−P *process ID* Specify the process ID of an existing process that the user wishes to debug (see Adopting an Existing Process below).

−L Force the line-oriented interface, even if *xdb* can support the window-oriented interface on the terminal type specified by environment variable TERM.

−i *file* Redirect standard input to the child process from the designated file or character device.

−o *file* Redirect standard output from the child process to the designated file or character device.

−e *file* Redirect standard error from the child process to the designated file or character device.

−S *num* Set the size of the string cache to *num* bytes (default is 1024, which is also the minimum). The string cache holds data read from the *objectfile*.

There can only be one *objectfile* and one *corefile* per debugging session (activation of the debugger). The program (*objectfile*) is not invoked as a child process until you give an appropriate command (see the *Job Control Commands* section below). The same program may be restarted, as different child processes, many times during one debugging session.

At startup, *xdb* executes commands from the file **.xdbrc**, if it exists in the user's home directory as specified by the environment variable HOME.

This debugger is a complex, interactive tool whose many capabilities are often limited only by your imagination. However, the debugger is also only a "window" to the world of the program being debugged and the system on which it runs. If something puzzling happens, consult a manual that describes the program or the system to better understand the behavior.

USER INTERFACE

Environment

Xdb reads the environment variable TERM to determine the applicable terminal type and user interface. The locale category LC_CTYPE, which defaults to the 'C' locale, is used to determine the character-set and semantics to be used when reading or writing character and string data.

Window-Oriented Interface

This user interface is only supported on HP terminals with memory-lock. The top of the screen is a "window" into the current source file, and the bottom of the screen is for *xdb* and user program input and output. Separating the two areas is a line (in inverse video) indicating the current file, procedure, and line number. Within the source file window, a ">" points to the current location (which might not be the location at which the user program is currently stopped).

To determine the number of lines and columns, *xdb* looks first for the environment variables LINES and COLUMNS. If these environment variables are not found, *xdb* uses the line and column information from the **/usr/lib/terminfo** file for the terminal type found in the

environment variable TERM.

Note that a TERM setting of 'hp' implies the terminal supports memory-lock, which is not true of some HP terminals (such as the HP2621). For this and other cases, the debugger attempts to provide a window-oriented interface even if the terminal itself does not provide the necessary capabilities. For such situations, the debugger can be forced into the line-oriented interface mode by invoking it with the **−L** command line option.

Line-Oriented Interface

This is the user interface currently supported on HP terminals without memory-lock and on all non-HP terminals. This interface does not use windows. The source file is displayed one line at a time, but some commands compensate by supplying more information than with the "window" interface.

CONVENTIONS

The debugger remembers the current file, procedure, line, and data location that you have been viewing (not necessarily executing) most recently. Many commands use these current locations as defaults, and many commands set them as a result. Keep this in mind when deciding what a command does in any particular situation.

For example, if you stop in procedure "abc", then view procedure "def", then ask for the value of local variable "xyz", the debugger assumes that the variable belongs to procedure "def".

Notational Conventions

Most commands are of the form "*command* [*location*] [*command-arguments*] [*command-list*]". Numeric modifiers after commands can be any numeric expression. They need not be just simple numbers. A blank is required before any numeric *option*. Multiple commands on one line must be separated by "**;**".

These are common modifiers and other special notations:

(A | B | C) Any one of A or B or C is required.

[A | B | C] Any one of A or B or C is optional.

command-list A series of debugger commands, separated by "**;**", entered on the command line or saved with a breakpoint or assertion. Semicolons are ignored (as commands) so they can be freely used as command separators. Commands may be grouped with "{}" for the "**a**", "**b**", "**if**", "**i**" (the abbreviated "if" command), and "**!**" commands. In all other cases commands inside "{}" are ignored.

count The number of repetitions specified for a command.

depth A stack depth as printed by the "**t**" command. The top procedure is at a *depth* of zero. A negative *depth* acts like a *depth* of zero. Stack depth usually means "exactly at the specified depth", not "the first instance at or above the specified depth."

expr Any expression, but with limitations stated below.

file A file name.

format A style for printing data (see Data Viewing Commands below).

line A *number* that refers to a particular line in a file.

location A particular *line* in a file (and its corresponding address in the user's program if there exists executable code for that line). *location* has the following general forms:

line

file [: *line*]
proc [: *proc* [. . .]] [: (*line* | *#label*)]

number A specific, constant number (e.g. "9", not "4+5"). Floating point (real) numbers may be used any place a constant is allowed.

proc A procedure (or function, or subroutine) name.

var A variable name.

Variable Name Conventions

Variables are referenced exactly as they are named in your source file(s). Case sensitivity is controlled by the "**tc**" command.

If you are interested in the value of some variable *var*, there are a number of ways of getting it, depending on where and what it is:

var Search the stack for the most recent instance of the current procedure. If found, see if *var* is a parameter or local variable of that procedure. If not, search for a global variable named *var*.

proc:*var* Search the stack for the most recent instance of *proc*. If found, see if it has a parameter or local variable named *var*, as before.

proc:*depth*:*var* Use the instance of *proc* that is at depth *depth* (exactly), instead of the most recent instance. This is very useful for debugging recursive procedures where there are multiple instances on the stack.

:*var* Search for a global (not local) variable named *var*.

Dot is shorthand for the last thing you viewed (see the *Data Viewing Commands* section below). It has the same size it did when you last viewed it. For example, if you look at a **long** as a **char**, then "." is considered to be one byte long. This is useful for treating things in unconventional ways, like changing the second highest byte of a **long** without changing the rest of the **long**. *Dot* may be treated like any other variable.

NOTE: "." is the *name* of this magic location. If you use it, it is dereferenced like any other name. If you want the *address* of something that is, say, 30 bytes farther on in memory, do not say ".+30". That would take the contents of *dot* and add 30 to it. Instead, say "&.+30", which adds 30 to the *address* of *dot*.

Special variables are names for things that are not normally directly accessible. Special variables include:

$*var* The debugger has room in its own address space for a number of user-created special variables. They are all of type **long**, and do not take on the type of any expression assigned to them. Names are defined when they are first seen. For example, saying "p $xyz = 3*4" creates special symbol "$xyz", and assigns to it the value 12. Special variables may be used just like any other variables. Names are limited to 100 characters.

$pc, **$sp**, **$r7**, etc.
These are the names of the program counter, the stack pointer, the CPU general registers, etc. To find out which names are available on your system, use the "**lr**" (list registers) command. All registers act as type **integer**.

$result This is used to reference the return value from the last command line procedure call. Where possible, it takes on the type of the procedure. **$short** and **$long** are available as alternate ways of looking at **$result**.

$signal This lets you see and modify the current child process signal number.

$lang This lets you see and modify the current language. The current language determines the operators that can be used in expressions, and the format in which variables are displayed. Values that can be assigned to $lang are "C", "FORTRAN", "Pascal" and "default" ("default" means use whatever language the current procedure is written in).

$print Alters the behavior of the "print" command when printing character data. Values that can be assigned are "ascii", "native", and "raw". Default is "ascii". "Ascii" causes all non-ASCII characters to be displayed as octal-escapes. "Native" causes unprintable characters, as determined by the locale category (environment variable) LC_CTYPE, to be displayed as octal-escapes. "Raw" causes all bytes to be output unaltered. This also affects the default display format for character types (see Display Formats).

$line This lets you see and modify the current source line number, which is also settable with a number of different commands.

$malloc This lets you see the current amount of memory (bytes) allocated at run-time for use by the debugger itself.

$step This lets you see and modify the number of machine instructions the debugger will step while in a non-debuggable procedure before setting an up-level breakpoint and free-running to it. Setting it to a small value can improve debugger performance at the risk of taking off free-running after missing the up-level break for some reason.

To see all the special variables, including the predefined ones, use the "**ls**" (list specials) command.

You can also look up code addresses with

proc:*line*

which searches for the given procedure name and line number (which must be an executable line within *proc*) and uses the code address of that line. Just referring to a procedure *proc* by name uses the code address of the first executable statement in that procedure.

Expression Conventions

Every expression has a value, even simple assignment statements, as in C.

Integer constants may begin with "**0**" for octal or "**0x**" or "**0X**" for hexadecimal. If followed immediately by "**l**" or "**L**", they are forced to be of type **long**. Likewise, "**u**" and "**U**" force the type to **unsigned**. "**ul**" or "**UL**" corresponds to **unsigned long**. If no suffix is used, the smallest type in which the value will fit is used.

Floating point constants must be of the form *digits.digits*[**e**|**E**|**d**|**D**|**L**|**l**[+|-]*digits*][**f**|**F**|**l**|**L**], for example, "1.0", "3.14e8f", or "26.62D-31". One or more leading digits is required to avoid confusion with "." (*dot*). A decimal point and one or more following digits is required to avoid confusion for some command formats. If the exponent does not exactly fit the pattern shown, it is not taken as part of the number, but as separate token(s). The "**d**" and "**D**" exponent forms are allowed for compatibility with FORTRAN. The "**l**" and "**L**" exponent forms are allowed for compatibility with Pascal.

In the absence of a suffix character, the constant is assumed to be of type "**double**" (8-byte IEEE real). The suffixes "**f**" and "**F**" cause the value to be evaluated as type "**float**" (4-byte IEEE real). The suffixes "**l**" and "**L**" cause the value to be evaluated as type "**long double**" (16-byte IEEE real). Unless a direct assignment is made, float and long double types are converted to type double before the expression is evaluated.

Character constants must be entered in ' and are treated as **integer**s. C string constants must be entered in "" and are treated like "**char ***" (e.g. pointer to **char**). FORTRAN and Pascal strings may be enclosed in either ' or "". Character and string constants may contain the standard backslashed escapes understood by the C compiler and the *echo*(1) command, including "\a", "\b", "\f", "\n", "\r", "\t", "\?", "\\", "\'", "*nnn*", and "\x*nnn*...". In the case of hex-escapes, the longest possible value is evaluated and then truncated to size of the destination type (either 1 or 4 bytes). "\<newline>" is not supported, neither in quotes nor at the end of a command line.

The prefix character "**L**" can be used to denote wide character or string constants (wchar_t). Use of the prefix will cause the value to be mapped to its wide-character equivalent before being stored (see *multibyte*(3C)). If an unmappable value is encountered, it is stored unconverted.

Expressions are composed of any combination of variables, constants, and operators. The debugger recognizes the language of the object file (that is, **a.out**), and switches to its language-specific operators. The user can also create a composite program, built of two or more supported languages, and debug without regard to which parts are comprised of which language. The global variable **$lang** is set as necessary by the debugger, based on the symbol table data supplied with the object file. The **$lang** variable can be set to "C", "FORTRAN", "Pascal" or "default".

If no active child process and no *corefile* exist, you can only evaluate expressions containing constants.

Expressions approximately follow the C rules of promotion, e.g. **char**, **short**, and **int** become **long**, and **float** becomes **double**. If either operand is a **double**, floating math is used. If either operand is **unsigned**, unsigned math is used. Otherwise, normal (integer) math is used. Results are then cast to proper destination types for assignments.

If a floating point number is used with an operator that does not normally permit it, the number is cast to **long** and used that way. For example, the C binary operator "~" (bit invert) applied to the constant "3.14159" is the same as "~3".

Note that "=" means "assign" except for Pascal; to test for equality, use "==" for C, or "**.EQ.**" for FORTRAN. In Pascal, "=" is a comparison operator; use "**:=**" for assignments. For example, if you invoke the debugger, then set "p $lang = Pascal", you must say "p $lang := C" to return to C.

The special unary operator "**$in**" (not to be confused with debugger local variables) evaluates to 1 (true) if the operand is an address inside a debuggable procedure and **$pc** (the current child process program location) is also in that procedure, else it is 0 (false). For example, "**$in** main" is true if the child process is stopped in main().

You can attempt to dereference any constant, variable, or expression result using the C "*" operator. If the address is invalid, an error is given.

Type casting is allowed. For simple types, the syntax is identical to C. For example:

```
(short) size
(double *) mass_ptr
```

These casts are limited to **char**, **short**, **long**, **int**, **unsigned**, **float**, **double**, appropriate combinations of these keywords, and single level pointer types. Also supported are structure and union pointer type dereferences. For example:

```
bar_ptr = &bar
(struct foo) &bar
(struct foo) bar_ptr
```

Both of these casts treat "bar" as a struct of type "foo" during printing. Structure and union casts may only include the keyword "struct" or "union" and an appropriate tag. No pointers

("*") are allowed. The argument of the cast is simply treated as an address.

Whenever an array variable is referenced without giving all its subscripts, the result is the address of the lowest element referenced. For example, consider an array declared as "x[5][6][7]" in C, "x(5,6,7)" in FORTRAN, or "x[1..5,2..6,3..7]" in Pascal. Referencing it simply as "x" is the same as just "x" in C, the address of "x(1,1,1)" in FORTRAN, or the address of "x[1,2,3]" in Pascal. Referencing it as "x[4]" is the same as "& (x[4][0][0])" in C, the address of "x(1,1,4)" in FORTRAN, or the address of "x[4,2,3]" in Pascal.

If a not-fully-qualified array reference appears on the left side of an assignment, the value of the right-hand expression is stored into the element at the address specified.

String constants are stored in a buffer in the file **/usr/lib/xdbend.o** on the series 800 or **/usr/lib/end.o** on the series 300, which you link with your program. The debugger starts storing strings at the beginning of this buffer, and The capacity of this buffer is 512 bytes. moves along as more assignments are made. If the debugger reaches the end of the buffer, it goes back and reuses it from the beginning. In general this does not cause any problems. However, if you use very long strings, or if you assign a string constant to a global pointer, problems could arise.

Procedure Call Conventions

Procedures may be invoked from the command line, even within expressions. For example:

```
p xyz = $abc * (3 + def (ghi - 1, jkl, "Hi Mom"))
```

calls procedure "def" when its value is needed in the expression.

Any breakpoints encountered during command line procedure invocation are handled as usual. However, the debugger has only one active command line at a time. If it stops in a called procedure for any reason, the remainder (if any) of the old command line is discarded, with notice given.

If you attempt to call a procedure when there is no active child process, one is started for you as if you gave a single-step command first. Unfortunately, this means that the data in *corefile* (if any) may disappear or be reinitialized.

If you send signal SIGINT (e.g., hit the BREAK key) while in a called procedure, the debugger aborts the procedure call and returns to the previous stopping point (the start of the main program for a new process).

You can call any procedure that is in your *objectfile*, even if it is not debuggable (was not compiled with debug on). For example, assume that you reference "printf()" in your program, so the code for it is in your *objectfile*. Then you can enter on the command line:

```
p printf ("This works! %d %c\n", 9, '?');
```

To determine what procedures are available, do a list labels command ("**ll**").

COMMANDS

The debugger has a large number of commands for viewing and manipulating the program being debugged. They are explained below, grouped by functional similarity.

The command line editing and history features from *ksh*(1) are available during command input (see *ksh*(1)). The environment variables XDBEDIT, EDITOR, or VISUAL are checked (in that order) to determine which of the three available editing modes (vi, emacs, or gmacs) is used. The command history file is specified by the environment variable XDBHIST, and its size is derived from HISTFILE. If any of these environment variables is not set, the default is the same as with ksh(1) except that XDBHIST defaults to "$HOME/.xdbhist".

Window Mode Commands

These commands control what is displayed in the source window, in addition the "**td**" command determines whether single stepping is at the source statement level or the assembly

instruction level. The source window has three different modes. In source mode, the window is filled with source lines from the user program. In disassembly mode, the top five lines of the source window displays one of several sets of registers (see the **"gr"**, **"fr"** and **"sr"** commands), and the remainder of the window displays assembly language instructions. In split screen mode, the top half of the window displays source code, while the bottom half displays the corresponding assembly instructions.

td Toggle disassembly mode. When in disassembly mode, the source window displays one of several sets of registers (see **"gr"**, **"fr"** and **"sr"**) and the code in assembly language. In addition, the single step command steps one assembly instruction at a time rather than a source statement at a time. The assembly language display consists of the source line number, the address in hex, the address in the form of nearest label plus offset, and the assembly instruction. If the debugger is already in split screen mode, then the **"td"** command changes the level of single stepping, but has no change on the display other than the mode displayed in the line separating the source statements from the assembly instructions.

ts Toggle split-screen mode. When in split-screen mode, the source window is half source code and half assembly instructions. In split screen mode, the **"td"** command still toggles the debugger between symbolic (or source) mode and assembly mode, as indicated by the line separating the source from the assembly. The only difference is whether single stepping is at the source statement or assembly instruction level.

gr Display the general registers when the debugger is in assembly (non-split-screen) mode. When the value of a register changes, that register is highlighted until after the next command. General registers may be modified by using the debugger special variables. When displaying the general registers or the floating point registers, the line dividing the registers from the assembly code displays certain special processor registers. Some registers are displayed as a string of letters, each letter representing a bit in the register. A lowercase letter indicates that the bit is off, uppercase means on.

fr Display the floating point registers when the debugger is in assembly (non-split-screen) mode. On a series 300 having multiple floating point processors, you will be asked which set of registers you want to display. When the value of a register changes, that register is highlighted until after the next command. On the series 800, these registers may be modified by using the debugger special variables $f0 through $f31 (Note that the register display indicates sixteen 64-bit registers, while there are 32 32-bit floating point register special variables), not all floating point registers are writeable.

w [*size*] Set the size of the source viewing window. It is normally set to 15 lines for a 24 line terminal. If the line oriented interface is being used, this command prints *size* lines centered around the current location.

u Update the screen to reflect the current location. This command is best used as part of an assertion (see Assertion Control Commands).

U Redraw the screen. This is used when the screen is corrupted.

File Viewing Commands

These commands may change the current viewing position, but they do not affect the next statement to be executed in the child process, if any.

v View the source one window forward from the *current* source window. One or two lines from the previous window are preserved for context. If the line

oriented interface is in use only the next source line is displayed.

v [*location*] View the source at the specified *location*, placing it in the center of the window. If the line oriented interface is in use only the source line *location* is displayed.

V [*depth*] View the current procedure at depth *depth* on the stack in the source window. If not specified the *depth* defaults to zero, which is where the program is currently stopped.

va [*aaaress*] View the assembly code at *aaaress* in the source window. *Aaaress* is an expression that may include constants and code labels (ex. _start + 0x20) This command is useful only when in disassembly mode.

L Display the file name, procedure name, line number, and the current source statement corresponding to the object code being executed or examined. This allows you to determine where you are in the program, and is most useful in assertion and breakpoint command lists.

+[*lines*] Move to *lines* (default one) lines after the current line.

-[*lines*] Move to *lines* (default one) lines before the current line.

/[*string*] Search forward through the current file, from the line after the current line, for *string*.

?[*string*] Search backward for *string*, from the line before the current line.

Searches wrap around the end or beginning of the file, respectively. If *string* is not specified, the previous one is used. Wild cards and regular expressions are not supported; *string* must be literal.

n Repeat the previous "**/**" or "**?**" command using the same *string* as previously.

N The same as "**n**", but the search goes in the opposite direction as specified by the previous "**/**" or "**?**" command.

Display Formats

A *format* is of the form "[*][*count*]*formchar*[*size*]".

"*" means "use alternate address map" (only supported on the series 300).

Count is the number of times to apply the format style *formchar*. It must be a *number*.

Size is the number of bytes to be formatted for each *count*, and overrides the default *size* for the format style. It must be a positive decimal *number* (except short hand notations, see below). *Size* is disallowed with those *formchar*s where it makes no sense.

For example, "p abc\4x2" prints, starting at the location of "abc", four two-byte numbers in hexadecimal.

The formats which print numbers allow an uppercase character to be used instead, for the same results as appending "**l**" (see below). For example, "**O**" prints in long octal. The following formats are available:

n Print in the "normal" format, based on the type. Arrays of **char** and pointers to **char** are interpreted as strings, and structures are fully dumped.

(**d** | **D**) Print in decimal (as **integer** or **long**).

(**u** | **U**) Print in unsigned decimal (as **integer** or **long**).

(**o** | **O**) Print in octal (as **integer** or **long**).

(**x** | **X**) Print in hexadecimal (as **integer** or **long**).

(b | B) Print a byte in decimal (either way).

c Print a character.

C Print a wide-character. Attempts conversion to the external character-set (as determined by the locale category LC_CTYPE) before printing; see *multibyte(3C)*.

(e | E) Print in "e" floating point notation (as **float**, **double**, or **long double**) (see *printf*(3S)). Remember that floating point expressions are always doubles! Some floating point values represent non-numeric values. Refer to ecvt(3c).

(f | F) Print in "f" floating point notation (as **float**, **double**, or **long double**).

(g | G) Print in "g" floating point notation (as **float**, **double**, or **long double**).

a Print a string using *expr* as the address of the first byte.

w Print a wide-character string using *expr* as the address of the first element. Attempts conversion to the external character set before printing.

W Print a wide-character string using *expr* as the address of a pointer to the first element. Attempts conversion to the external character set before printing. This is the same as saying "*expr***w**", except for arrays.

s Print a string using *expr* as the address of a pointer to the first byte. This is the same as saying "*expr***a**", except for arrays.

t Show the type of *expr* (usually a variable or procedure name). For true procedure types you must actually call the procedure, e.g. "def (2)\t".

p Print the name of the procedure containing address *expr*.

S Do a formatted dump of a structure. Note that *expr* must be the address of a structure, not the address of a pointer to a structure.

There are some short hand notations for *size*:

b 1 byte (**char**).

s 2 bytes (**short**).

l 4 bytes (**long**).

D 8 bytes (double). Can only be used with floating-point formats.

L 16 bytes (long double). Can only be used with floating-point formats.

These can be appended to *formchar* instead of a numeric *size*. For example, "abc\xb" prints one byte in hexadecimal.

If you view an object with a *size* (explicitly or implicitly) less than or equal to the size of a **long**, the debugger changes the basetype to something appropriate for that *size*. This is so "**.**" (*dot*) works correctly for assignments. For example, "abc\c2" sets the type of "**.**" to **short**. One side effect is that if you look at a **double** using a **float** format, *dot* loses accuracy or has the wrong value.

The value of the **$print** special variable affects the default format for character types as follows: for "ascii" mode the default format is "**x**" for unsigned char and wchar_t types. In "native" and "raw" modes, the defaults for unsigned char and wchar_t are "**c**" and "**C**" respectively. Likewise, "**s**" and "**W**" are used for pointers to unsigned char and wchar_t.

Data Viewing and Modification Commands

p *expr* If *expr* does not look like anything else (such as a command), it is handled as if you had typed "**p** *expr***n**" (print expression in normal format). Note that

modification of variables is done by using the assignment operator in the expression (ex. "p foo = 7" in C or FORTRAN, or "p foo := 7" in Pascal).

p *expr**format* Print the contents (value) of *expr* using *format*. For example, "abc\x" prints the contents of "abc" as an **integer**, in hexadecimal.

p *expr*?*format* Print the address of *expr* using *format*. For example, "abc?o" prints the address of "abc" in octal.

p -[[\] *format*] Back up to the preceding memory location (based on the size of the last thing displayed). Use *format* if supplied, or the previous *format* if not. Note that no "\" is needed after the "-".

p +[[\] *format*] Go forward to the following memory location (based on the size of the last thing displayed). Use *format* if supplied, or the previous *format* if not. Note that no "\" is needed after the "+".

l [*proc*[:*depth*]] List all parameters and local variables of the current procedure (or of *proc*, if given, at the specified *depth*, if any). Data is displayed using "**\n**" format, except that all arrays and pointers are shown simply as addresses, and only the first word of any structure is shown.

l (**a** | **b** | **d** | **z**) List all **a**ssertions, **b**reakpoints, **d**irectories (where to search for files), or **z**ignals (signal actions).

l (**c** | **f** | **g** | **l** | **m** | **p** | **r** | **s**) [*string*]
List all **c**ommon blocks in the current procedure, **f**iles (source files which built *objectfile*), **g**lobal variables, **l**abels (program entry points known to the linker), **m**acros, **p**rocedure names, **r**egisters, or **s**pecial variables (except registers). If *string* is present, only those things with the same initial characters are listed.

Stack Viewing Commands

t [*depth*] [*format*]
Trace the stack for the first *depth* (default 20) levels. Use the *format* if specified. For procedures that are not compiled with symbolic debug, *xdb* displays the name of the procedure, and in parentheses, a best guess at the procedure parameters. For the series 300, it is five integers. For the series 800, it is the first four words of the parameter spill area. Note that the procedure might not have spilled the four argument registers (the values might still be in the argument registers, or might have been moved to some other registers), and therefore the values printed by *xdb* are not guaranteed to be the correct values of the first four words of the procedure's argument list.

T [*depth*] [*format*]
The same as "**t**", but local variables are also displayed, using "**\n**" format (except that all arrays and pointers are shown simply as addresses, and structures as first words only). Use the *format* if specified.

Job Control Commands

The parent (debugger) and child (*objectfile*) processes take turns running. The debugger is only active while the child process is stopped due to a signal, including hitting a breakpoint, or terminated for whatever reason.

r [*arguments*] Run a new child process with the given argument list (if any). The existing child process, if any, is terminated first. If no *arguments* are given, the ones used with the last "**r**" command are used again (none if "**R**" was used last).

Arguments can contain "<" and ">" for redirecting standard input and standard output. ("<" does an *open*(2) of file descriptor 0 for read-only; ">" does

a *creat*(2) of file descriptor 1 with mode 0666). Redirection can also be done with ">>" and ">&". *Arguments* can contain shell variables and metacharacters, quote marks, or other special syntax. The remainder of the input-line following the "**r**" command is used as the argument-list, so it cannot be enclosed in a command list ("{}"). Thus, "**r**" cannot be used within a breakpoint, assertion, or "**if**" command.

R Run a new child process with no argument list.

k Terminate (kill) the current child process if it exists.

c [*location*] Continue from a breakpoint ignoring the signal. Set a temporary breakpoint at the specified *location.*

C [*location*] Continue just like "**c**", but allow the signal (if any) to be received. This is fatal to the child process if it does not catch or ignore the signal! Set a temporary breakpoint at the specified *location.*

s [*count*] Single step 1 (or *count*) statements. Successive carriage-returns repeat with a *count* of 1. If *count* is less than one, the child process is not stepped. Note that the child process continues with the current signal, if any! (You can set "$signal = 0" to prevent this.)

If you accidentally step down into a procedure you do not care about, use the "**bu**" command to set a temporary up-level breakpoint, and then continue using "**c**".

S [*count*] Single step like "**s**", but treat procedure calls as single statements (do not follow them down). If a breakpoint is hit in such a procedure, or in one that it calls, its *commands* are executed. This is usually all right, but beware if there is a "**c**" command in that breakpoint's command list!

The debugger has no knowledge about or control over child processes forked in turn by the process being debugged. Also, it gets very confused (leading to "Bad access" messages) if the process being debugged executes a different program via *exec*(2).

Child process output may be (and usually is) buffered. Hence it may not appear immediately after you step through an output statement such as *printf*(3S). It may not appear at all if you kill the process.

Breakpoint Commands

The debugger provides a number of commands for setting and deleting breakpoints. Associated with a breakpoint are three attributes:

address All the commands which set a breakpoint are simply alternate ways to specify the breakpoint address. The breakpoint is then encountered whenever *address* is about to be executed, regardless of the path taken to get there. Only one breakpoint at a time of a particular type may be set at a given *address.* Setting a new breakpoint at *address* replaces the old one of the same type, if any.

count The number of times the breakpoint is encountered prior to recognition. A count is of the form \<**expr**>, \<**expr**> **p** (p for permanent, the default), or \<**expr**> **t** (t for temporary). *count* decrements with each encounter. Each time *count* goes to zero, the breakpoint is recognized. If the breakpoint is permanent, *count* is reset to the original count. If the breakpoint is temporary, once *count* goes to zero, the breakpoint is recognized, then deleted.

A *count* of zero is used internally by the debugger and means that the breakpoint is deleted when the child process next stops for any reason, whether it hit that breakpoint or not. Commands saved with such breakpoints are

ignored. Normally you never see these sorts of breakpoints.

Note that once a breakpoint exists, the count can then be modified only by the "**bc**" command.

commands Actions to be taken upon recognition of a breakpoint before waiting for command input. These are separated by "**;**" and may be enclosed in "{}" to delimit the list saved with the breakpoint from other commands on the same line. If the first character is anything other than "{", or if the matching "}" is missing, the rest of the line is saved with the breakpoint.

Saved commands are not parsed until the breakpoint is recognized. If *commands* are nil then, after recognition of the breakpoint, the debugger just waits for command input.

The debugger has only one active command line at a time. When it begins to execute breakpoint commands, the remainder (if any) of the old command line is discarded, with notice given.

Each breakpoint is individually active or suspended, and there is an overall breakpoint mode. If any breakpoint is added or activated, or if all breakpoints become suspended, the global mode follows suit.

Here are the breakpoint commands:

lb List all breakpoints in the format "*num*: count: *nnn proc*: *ln*: *contents*", followed by "{*commands*}", for example:

```
1:  count: 1t  sortall: 12: abc += 1;
      {t;i\D}
2:  count: 5  fixit: 29: def = abc >> 4;
      {Q;if *argv == -1 {"Oops"}{c}}
```

The leftmost number is an index number for use with the "**db**" (delete) command.

b [*location*] [*count*] [*commands*]
Set a permanent breakpoint at the current location (or at *location*). Set the *count* number of times through breakpoint. When the breakpoint is hit, *commands* are executed. If there are none, the debugger pauses for command input. If immediate continuation is desired, finish the command list with "**c**" (see breakpoint 2 in the example above).

db [*number*] Delete breakpoint number *number*. If *number* is absent, delete the breakpoint at the current line, if any. If there is none, the debugger executes a "**lb**" command instead.

db * Delete all breakpoints (including "procedure" breakpoints).

bp [*commands*] Set permanent breakpoints at the beginning (first executable line) of every debuggable procedure. When any procedure breakpoint is hit, *commands* are executed.

It is permissible to set other permanent or temporary breakpoints at the same locations as these "procedure entry" breakpoints. If a procedure and non-procedure breakpoint are both hit at the same location, the non-procedure breakpoint has priority. It is not possible to alter the "count" of a procedure entry breakpoint. Procedure entry breakpoints must be activated and deleted as a group; it is not possible to set or delete individual ones.

Procedure entry breakpoints are useful for procedure stepping and tracing. For example, the command

```
bp Q;t 1;c
```

sets up procedure tracing by printing the current procedure at each breakpoint.

bpx [*commands*] Set permanent breakpoints at the exit (final executable statement) of every debuggable procedure. When any procedure exit breakpoint is hit, *commands* are executed.

bpt [*commands*] Set permanent breakpoints at the entry and exit (first and final executable statements) of every debuggable procedure. The commands if any, are associated with the entry breakpoint. Commands for the exit breakpoint are "**Q;L;c**" for the series 300, and "**Q;p $ret0;c**" for the series 800.

dp Delete all "procedure entry" breakpoints. All breakpoints set by commands other than "**bp**" will remain set.

Dpx Delete all "procedure exit" breakpoints. All breakpoints set by commands other than "**bpx**" will remain set.

Dpt Delete all "procedure trace" breakpoints. All breakpoints set by commands other than "**bpt**" will remain set.

abc *commands* Define a global breakpoint command that will be executed whenever any breakpoint is hit (normal, procedure, procedure exit, or procedure trace).

dbc Delete the global breakpoint command.

bb [*depth*] [*count*] [*commands*]
Set a breakpoint at the beginning (first executable line) of the procedure at the given stack *depth*. If *depth* is not specified, it uses the current procedure, which might not be the same as the one at *depth* zero.

bx [*depth*] [*count*] [*commands*]
Set a breakpoint at the exit (last executable line) of the procedure at the given stack *depth*. If *depth* is not specified, it uses the current procedure, which might not be the same as the one at *depth* zero. The breakpoint is set at a point such that all returns of any kind go through it.

bu [*depth*] [*count*] [*commands*]
Set an up-level breakpoint. The breakpoint is set immediately after the return to the procedure at the specified stack *depth* (default one, not zero). A *depth* of zero means "current location", e.g. "**bu 0**" is a way to set a temporary breakpoint at the current value of **$pc**.

bt [(*depth* | *proc*)] [*count*] [*commands*]
Trace the current procedure (or procedure at *depth*, or *proc*). This command sets breakpoints at both the entrance and exit of a procedure. By default, the entry breakpoint *commands* are "**Q;2t;c**", which shows the top two procedures on the stack and continues. The exit breakpoint is always set to execute "**Q;L;c**" on the series 300, and "**Q;p $ret0;c**" on the series 800.

If *depth* is given, *proc* must be absent or it is taken as part of *commands*. If *depth* is missing but *proc* is specified, the named procedure is traced. If both *depth* and *proc* are omitted, the current procedure is traced, which might not be the same as the one at *depth* zero.

If *commands* are present, they are used for the entrance breakpoint instead of the default shown above.

ba *address* [*count*] [*commands*]
Set a breakpoint at the given code address. Note that *address* can be the name of a procedure or an expression containing such a name. Of course, if the child process is stopped in a non-debuggable procedure, or in prologue code (before the first executable line of a procedure), things may seem a little strange.

bc *number count*
Set the count of breakpoint *number* to *count*.

sb [*num*] Suspend breakpoint number *num*. If no breakpoint number is given, suspend the breakpoint at the current line.

sb * Suspend all breakpoints.

ab [*num*] Activate breakpoint number *num*. If no breakpoint number is given, activate the breakpoint at the current line.

ab * Activate all breakpoints.

tb Toggle the overall breakpoints mode between *active* and *suspended*.

The next three commands are not strictly part of the breakpoint group, but are used almost exclusively as arguments to breakpoints or assertions.

if [*expr*] {*commands*}[{*commands*}]
If *expr* evaluates to a non-zero value, the first group of commands (the first "{}" block) is executed, else it (and the following "{", if any) is skipped. In general, all other "{}" blocks are always ignored (skipped), except when given as an argument to an "**a**", "**b**", or "**!**" command. The "**if**" command is nestable, and may be abbreviated to "**i**".

Q If the "**Q**uiet" command appears as the first command in a breakpoint command list, the usual announcement of "*proc: line: text*" is not made. This allows quiet checks of variables, etc. to be made without cluttering the screen with unwanted output. The "**Q**" command is ignored if it appears anywhere else.

"*any string you like*"
Print the given string, which may have the standard backslashed character escapes in it, including "**\n**" for newline. This command is useful for labeling output from breakpoint commands.

Assertion Control Commands

Assertions are lists of commands that are executed *before every statement*. This means that, if there is even one active assertion, the program is single stepped at the machine instruction level. In other words, it runs very slowly. The primary use for assertions is tracking down nasty bugs, such as when someone corrupts a global variable. Some examples follow the command descriptions.

Each assertion is individually active or suspended, and there is an overall assertions mode. If any assertion is added or activated, or if all assertions become suspended, the global mode follows suit.

a *commands* Create a new assertion with the given command list, which is not parsed until it is executed. As with breakpoints, the command list may be enclosed in "{}" to delimit it from other commands on the same line. Use the "**la**" command to list all current assertions and the overall mode.

aa *number* Activate assertion *number*.

aa * Activate all assertions.

da *number* Delete assertion *number*.

da * Delete all assertions.

sa *number* Suspend assertion *number*.

sa * Suspend all assertions.

ta Toggle the overall assertions mode between *active* and *suspended*.

x [*mode*] Force an exit from assertions mode. If *mode* is absent, or if it evaluates to zero, exit immediately. Otherwise, finish executing the current assertion first. If any assertion executes an "**x**" command, the child process stops and the assertion doing the "**x**" is identified.

The debugger has only one active command line at a time. When it begins to execute assertion commands, the remainder (if any) of the old command line is discarded, with notice given.

Certain commands ("**r**", "**R**", "**c**", "**C**", "**s**", "**S**", and "**k**") are not allowed while assertions are running. They must appear after the "**x**", if at all.

A useful assertion might be:

```
a u
```

This "walks" the program (traces execution) until "something" happens (e.g., you hit the BREAK key).

Another example:

```
a L; if(xyz > (def - 9) * 10) {ta; x 1; c} {p abc -= 10}
```

This assertion prints the line just executed, then checks the condition. If it is false, "abc" is decremented by 10. If it is true, assertions are suspended, assertion mode is exited, and the program continues at normal speed. Without the number after the "**x**" command, the "**c**" command is not executed.

Another example:

```
a  if  ( abc != $abc)  {p  $abc = abc;p   abc\d;  if (abc > 9)  {x} }
```

This command sets up an assertion to report the changing value of some global variable ("abc"), and stop if it ever exceeds some value. It uses a debugger local variable ("$abc") to keep track of the old value of "abc".

Signal Control Commands

The debugger catches all signals bound for the child process before the child process sees them. (This is a function of the *ptrace*(2) mechanism.) For many signals, this is a reasonable thing to do. Most processes are not set up to handle segmentation errors, etc. However, some processes do quite a bit with signals and the constant need to continue from a signal catch can be tedious.

z [*signal*] **[i][r][s][Q]**

Modifies the "zignal" (signal) handling table. *Signal* is a valid signal number (the default is the current signal). The options (which must be all one word) toggle the state of the appropriate flag: **i**gnore, **r**eport, or **s**top. If "**Q**" is present, the new state of the signal is not printed.

Use the "**lz**" command to list the current handling of all signals. Note that just "**z** *signal*" with no options tells you the state of the selected signal.

For example, assuming a start up state of (do not ignore, do not report, do not stop), the command "**z 14 sr**" sets the alarm clock signal to **stop** (but still do not **ignore**) and **report** that it occurred. Doing "**z 14 sr**" again toggles the flags back to the original state.

When the child process stops or terminates on a signal it is always reported, except for the breakpoint signal when the breakpoint commands start with "**Q**".

When the debugger ignores a signal, the "**C**" command then does not know about it, and the signal will not be passed to the child process. The signal is never ignored when the child process terminates, only when it stops.

Record and Playback Commands

The debugger supports a record and playback feature to recreate program states and record all debugger output. It is particularly useful for bugs requiring long setups. Note: The file name can't be "**t**", "**f**", or "**c**", or begin with a "**@**".

The following commands are available:

>*file*	Set or change recordfile to *file* and turn recording on. This rewrites *file* from the start. Only commands are recorded to this file.
>>*file*	This is the same, but appends to *file* instead of overwriting.
>**@***file* >>**@***file*	Set or change record-all file to *file*, for overwriting or appending. The record-all file may be opened or closed independently of (in parallel with) the recordfile. All debugger standard output is copied to the record-all file, including prompts, commands entered, and command output. However, child process output is not captured.
>(**t** | **f** | **c**)	Turn recording on ("**t**") or off ("**f**"), or close the recording file ("**c**"). When recording is resumed, it appends after commands recorded earlier. In this context, ">>" is the same as ">".
>**@**(**t** | **f** | **c**)	Turn record-all on, off, or close the record-all file. In this context, ">>**@**" is the same as ">**@**".
tr [@]	Toggle recording [record-all]; if ON turn it OFF, if OFF turn it ON.
>	Tell the current recording status. ">>" does the same thing.
>**@**	Tell the current record-all status. ">>**@**" does the same thing.
<*file*	Start playback from *file*.
<<*file*	Start playback from *file* using the single-step feature of playback. Each command line from the playback file is presented before it is executed. A simple menu lets you execute ("<cr>") or skip ("**S**") the line, execute more than one line ("<num>"), continue ("**C**") or quit ("**Q**") single stepping, or ask for help ("**?**").

Only command lines read from the keyboard or a playback file are recorded in the recordfile. For example, if recording is turned on in an assertion, it does not "take effect" until assertion execution stops.

Command lines beginning with ">", "<", or "**!**" are not copied to the current recordfile (but they are copied to the record-all file). You can override this by beginning such lines with blanks.

NOTE: The debugger can of course be invoked with standard input, standard output, and/or standard error redirected, independent of record and playback. If the debugger encounters an end of file while standard input is redirected from anything other than a terminal, it prints a message to standard output and exits, returning zero.

Macro Definition Commands

def *name* [*replacement-text*]
Define *name* as a macro whose value is *replacement-text*. *Name* can be any string of letters or digits. *Replacement-text* can be any string of letters, digits, blanks, tabs, or other printing characters, but it cannot be interrupted by a new line.

undef *name* Remove the macro definition from *name* so that *name* no longer exists as a replacement string macro. As a special case "*" can be entered for *number* to undefine all macros.

tm Toggle the state of the macro substitution mechanism between active and suspended. When macro substitution is suspended, the currently defined macros continue to exist, but they are not replaced in the command-line by their definitions. Additional macros may still be defined while macro substitution is suspended.

Miscellaneous Commands

sm Suspend the "more" (pagination) facility of the debugger output. This is most useful when breakpoints or assertions are printing a great deal of information to the screen, and you do not want the debugger to keep waiting for you to hit the space bar.

am Activate the "more" facility to paginate the debugger output.

<carriage-return>
~ Repeat the last command, if possible, with an appropriate increment, if any. Repeatable commands are those which print a line, print a window of lines, print a data value, single step, and single step over procedures. Note that <carriage-return> is saved in a *record* file as a "~" command.

! [*command-line*]
Invoke a shell program. If *command-line* is present, it is executed via *system*(3S). Otherwise, the environment variable SHELL gives the name of the shell program to invoke with a **−i** option, also using *system*(3S). If SHELL is not found, the debugger executes **/bin/sh −i**. In any case, the debugger then waits for the shell or *command-line* to complete.

As with breakpoints, *command-line* may be enclosed in "{}" to delimit it from other (debugger) commands on the same line. For example,

```
b 14 {!{date};c}; t; la
```

sets a breakpoint at line 14 that calls *date*(1), then continues; then (after setting the breakpoint), the debugger does a stack trace, then lists assertions.

[*text*] Flag this *text* as a comment to be echoed to the command window. The **#** must appear as the first non-blank character on the line and the remainder of the line is treated as a comment. It is also written to the currently open record file.

D *dirs* Adds *dirs* to the list of additional directory search paths for source files. This command is equivalent to the command-line option -d.

f ["*printf-style-format*"]
Set the address printing format using *printf*(3S) format specifications (**not** debugger format styles). Only the first 19 characters are used. If there is no argument, the format is set to a system-dependent default. All addresses are assumed to be of type **long**, so you should handle all four bytes to get something meaningful.

g *line* | *#label* Go to an address in the procedure on the stack at *depth* zero (not necessarily the same as the current procedure). This changes the program counter so *line* or the line *#label* appears on is the next line to be executed.

h
help Print the debugger help file (command summary) using *more*(1).

I Print information (inquire) about the state of the debugger.

M Print the current text (*objectfile*) and core (*corefile*) address maps.

M (t | c) [*expr*; [*expr*;...]]
Set the text (*objectfile*) or core (*corefile*) address map. The first zero to six map values are set to the *expr*s given (refer to adb(1) for details on address maps).

q Quit the debugger. To be sure you do not lose a valuable environment, this command requests confirmation.

tc Toggle case sensitivity in searches. This affects everything: file names, procedure names, variables, and string searches!

ADOPTING AN EXISTING PROCESS

Xdb is capable of adopting and debugging a free-running process. This is accomplished by invoking *xdb* with the **−P** *process ID* option.

To adopt a process, the effective user IDs of the debugger and the process to be adopted must match, or the effective user ID of the debugger must be root. When a process is adopted, it halts, and *xdb* displays where the program is halted, at which point the program can be debugged. If the user quits the debugger without killing the process, *xdb* removes all breakpoints from the process, and allows it to continue running. If a program is designed to be adopted by *xdb* when in a certain state (such as an error condition), it is important that the program infinite loop, rather than calling the system routine *sleep0*. A sleeping program cannot be adopted correctly by *xdb*.

SYMBOL TABLE DEPENDENCIES

When you try to display a variable which is a FORTRAN format label, a Pascal file-of-text, or a Pascal set, with no display format or with normal format ("**\n**"), the value is shown as "{format-label}", "{file-of-text}", or "{set}", respectively. You can use other formats, such as "**\x**", to display the contents of such variables.

Procedures in FORTRAN and Pascal may have alias names in addition to normal names. Aliases are shown by the "**lp**" (list procedures) command. They can be used in place of the normal name, as desired.

The procedure name "_MAIN_" is used as the alias name for the main program (main procedure) in all supported languages. Do not use it for any debuggable procedures.

FORTRAN ENTRY points are flagged "ENTRY" by the "**lp**" command.

When a compiler does not know array dimensions, such as for some C and FORTRAN array parameters, it uses 0:MAXINT or 1:MAXINT, as appropriate. The "**\t**" format shows such cases with "**[]**" (no bounds specified), and subscripts from 0 (or 1) to MAXINT are allowed in expressions.

Even though the symbol table supports C structure, union, and enumeration tags, C typedefs, and Pascal types, the debugger does not know how to search for them, even for the "**\t**" format. They are "invisible".

Some variables are indirect, so a child process must exist in order for the debugger to know their addresses. When there is no child process, the address of any such variable is shown as 0xfffffffe.

Symbol names in the Value Table are never preceded by underscores, so the debugger never bothers to search for names of that form. The only time a prefixed underscore is expected is when searching the Linker Symbol Table for names of non-debuggable procedures.

DIAGNOSTICS

Most errors cause a reasonably accurate message to be given. Normal debugger exits return zero and error exits return one. All debugger output goes to standard output except error messages given just before non-zero exits, which go to standard error.

Debugger errors are preceded by "panic: ", while user errors are not. If any error occurs during initialization, the debugger then prints "cannot continue" and quits. If any error happens after initialization, the debugger attempts to reset itself to an idle state, waiting for command input. If any error occurs while executing a procedure call from the command line, the context is reset to that of the normal program.

Child process (program) errors result in signals which are communicated to the debugger via the *ptrace*(2) mechanism. If a program error occurs while executing a procedure call from the command line, it is handled like any other error (i.e. you can investigate the called procedure). To recover from this, or to abort a procedure call from the command line, type DEL, BREAK, ^C, or whatever your interrupt character is.

WARNINGS

Xdb does not terminate on an interrupt (SIGINT), it jumps to its main loop, and awaits another command. However, this does not imply that sending xdb an interrupt is harmless. It can result in internal tables being left in an inconsistent state that could produce incorrect behavior.

Code that is not debuggable or does not have a corresponding source file is dealt with in a half-hearted manner. The debugger shows "unknown" for unknown file and procedure names, cannot show code locations or interpret parameter lists, etc. However, the linker symbol table provides procedure names for most procedures, even if not debuggable.

On some systems, if the debugger is run on a shared *objectfile* you cannot set breakpoints. (This may only apply if someone else is also executing the program.) This may be indicated by the error "Bad access" when you attempt to start a child process. If another person starts running *objectfile* while you are debugging, they and you may have some interesting interactions.

If the *address* given to a "**ba**" command is not a code address in the child process, strange results or errors may ensue.

If you set the address printing format to something *printf*(3S) does not like, you may get an error (usually memory fault) each time you try to print an address, until you fix the format with another "**f**" command.

Do not use the "**z**" command to manipulate the SIGTRAP signal. If you change its state you had better know what you are doing or be a very good sport!

If you single step or run with assertions through a call to *longjmp* (on *setjmp*(3C)), the child process will probably take off free-running as the debugger sets but never hits an up-level breakpoint.

Do not modify any file while the debugger has it open. If you do, the debugger gets confused and may display garbage.

Although the debugger tries to do things reasonably, it is possible to confuse the recording mechanism. Be careful about trying to play back from a file currently open for recording, or vice versa; strange things can happen.

Some compilers only issue source line symbols at the end of each logical statement or physical line, *whichever is greater*. This means that, if you are in the habit of saying "a = 0; b = 1;" on one line, there is no way to put a breakpoint after the assignment to "a" but before the

assignment to "b".

Some statements do not emit code where you would expect it. For example, assume:

```
 99:    for (i = 0; i < 9; i++) {
100:            xyz (i);
101:    }
```

A breakpoint placed on line 99 will be hit only once in some cases. The code for incrementing is placed at line 101. Each compiler is a little different; you must get used to what your particular compiler does. A good way of finding out is to use single stepping to see in what order the source lines are executed.

The output of some program generators, such as *yacc*(1), have compiler line number directives in them that can confuse the debugger. It expects source line entries in the symbol table to appear in sorted order. Removal of line directives fixes the problem, but makes it more difficult to find error locations in the original source file. The following script, run after *yacc*(1) and before *cc*(1), comments out line number changes in C programs:

```
sed "/# *line/s/^.*$/\/*&*\//" y.tab.c >temp.c
```

In general, line number directives (or compiler options) are only safe so long as they never set the number backwards.

The C operators "++", "--", and "**?:**" are not available. The debugger always understands all the other C operators, except "**sizeof**", if the default language is FORTRAN or Pascal. Users should use $sizeof which works in any language.

For FORTRAN, only the additional operators "**.NE.**", "**.EQ.**", "**.LT.**", "**.LE.**", "**.GT.**", "**.GE.**" "**.OR.**" "**.NOT.**" "**.AND.**" "**.EQV.**" and "**.NEQV.**" are supported.

For Pascal, only the operators ":=", "<>", "^", "^." (as in "x^.y"), "**and**", "**or**", "**not**", "**div**", "**mod**", "**addr**", and "**sizeof**" are added.

There is no support for FORTRAN **complex** variables, except as a series of two separate **float**s or **double**s.

The C operators "**&&**" and "**||**" are not short circuit evaluated as in the compiler. All parts of expressions involving them are evaluated, with any side-effects, even if it's not necessary.

The debugger does not understand C pointer arithmetic. "*(a+n)" is not the same as "a[n]" unless "a" has an element size of 1.

Xdb does not support identically-named procedures (legal in Pascal if the procedures are in different scopes). Xdb will always use the first procedure with the given name.

There is no support for Pascal packed arrays where the element size is not a whole number of bytes. Any reference into such an array may produce garbage or a bad access.

Pascal WITH statements are not understood. To access any variable you must specify the complete "path" to it.

The debugger supports call-by-reference only for known parameters of known (debuggable) procedures. If the object to pass lives in the child process, you can fake such a call by passing "& object", i.e. the address of the object.

Only the first number of a complex pair is passed as a parameter. Functions which return complex numbers are are not called correctly; insufficient stack space is allocated for the return area, which can lead to overwriting the parameter values.

Assignments into objects greater than four bytes in size from debugger special variables, result in errors or invalid results.

Case-insensitive searches are done in a crude way which equates some non-letters with other non-letters. For example, "[" and "{" are equal, as are "@" and "`".

Command lines longer than 1024 bytes are broken into pieces of that size. This may be relevant if you run the debugger with playback or with input redirected from a file.

DEPENDENCIES

Series 300

There is no support for C local variables declared in nested blocks, nor for any local overriding a parameter with the same name. When looking up a local by name, parameters come first, then locals in the order of the closing "}"'s of the block in which they are declared. When listing all locals, they are shown in the same order. When there is a name overlap, the address or data shown is that of the first variable with that name. There is no support for Pascal intermediate variables. To reference a variable local to an enclosing procedure, you must specify the procedure name and stack depth in the usual way (proc:depth:var).

There is no support for the C types **wchar_t** and **long double**.

Two special variables are available only on Series 300:

$fpa If this is set to a non-zero value, any sequence of machine instructions that constitute a single floating-point accelerator instruction is treated as a single instruction for machine-level single-stepping and display.

$fpa_reg If **$fpa** is set to a non-zero value, **$fpa_reg** indicates the address register used in floating point accelerator instruction sequences. A 0 corresponds to register a0, 1 to a1, etc. The default value is 2.

The following window-mode commands are available only on Series 300:

+r Scroll the floating-point register display forward four lines.

-r Scroll the floating-point register display back four lines.

Series 800

All programs are shared executables. This implies three limitations. You cannot set breakpoints or single step a program if another process is running it; you will get a "Bad access to child process". If you are debugging a program, and another process starts to run the same program, either through your process executing a fork(), or another process, such as a shell, executing an exec(), this second process can hit one of your breakpoints and generate a SIGTRAP. You cannot single step through a call to fork().

sr Display the special registers (space and control registers) when the debugger is in assembly (non-split-screen) mode. When the value of a register changes, it is highlighted until after the next command. The control registers cannot be modified.

To adopt a process running the program **a.out**, the file **a.out** must be preprocessed before it is run by issuing the command *pxdb* **a.out**. If the program is run first, *pxdb* does not get write permission for the **a.out** file, and the program will have to be aborted.

AUTHOR

Xdb was developed by HP and Third Eye Software.

FILES

a.out	Default *objectfile* to debug.
core	Default *corefile* to debug.
/usr/lib/xdb.help	Text file listed by the "**help**" command.

/usr/lib/end.o	Object file to link with all debuggable programs. (Series 300 only)
/usr/lib/xdbend.o	Object file to link with all debuggable programs. (Series 800 only)
/usr/lib/nls/$LANG/xdb.cat	The xdb message catalog.
$HOME/.xdbrc	The xdb startup file.

SEE ALSO

cc(1), echo(1), ld(1), more(1), creat(2), exec(2), fork(2), open(2), ecvt(3C), printf(3S), system(3S), a.out(4), LANG(5).

On some systems any of the following may exist: adb(1), fc(1), pc(1), sdb(1), ptrace(2), core(4), symtab(4), user(4), multibyte(3C).

EXTERNAL INFLUENCES

Environment Variables

LANG determines the local language equivalent of **y** for (yes/no) queries. LANG will also determine the language in which messages are displayed.

If LANG is not specified or is set to the empty string, a default of "C" (see *lang*(5)) is used. If any internationalization variable contains an invalid setting, behaves as if all internationalization variables are set to "C". See *environ*(5).

LC_CTYPE determines the interpretation of text as single and/or multi-byte characters, and their printability, when reading or writing character and string data. If LC_CTYPE is not specified in the environment or is set to the empty string, the value of LANG is used as the default.

International Code Set Support

Single- and multi-byte character code sets are supported.

NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**–c**] [–] [*file*]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, which are most useful if they are also read-only.

The command:

xstr **–c** *name*

extracts the strings from the C source in *name*, replacing string references by expressions of the form **(&xstr[***number***])** for some *number*. An appropriate declaration of *xstr* is placed at the beginning of the file. The resulting C text is placed in the file **x.c**, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled, a file **xs.c** declaring the common *xstr* space, can be created by the command:

xstr

This **xs.c** file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared), saving space and swap overhead.

Xstr can also be used on a single file. A command:

xstr *name*

creates files **x.c** and **xs.c** as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code that contains strings, which are not, in fact, needed. *Xstr* reads from its standard input when the argument '–' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc –E name.c | xstr –c –
cc –c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make*(1) can avoid remaking **xs.o** unless truly necessary.

AUTHOR

Xstr was developed by the University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

strings	Data base of strings
x.c	Massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	Temp file when 'xstr name' does not touch *strings*

WARNINGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr*, both strings are placed in the data base, when just placing the longer one there would be sufficient.

SEE ALSO

mkstr(1).

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [**–vdlt**] [**–N**<*secondary*><*n*> ...] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **–v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **–d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **–l** flag is given, the code produced in **y.tab.c** will not contain any **#line** constructs. Generally, this should only be used after **y.tab.c** has compiled successfully, since the **#line** directives allow the C compiler to give error messages that refer to the *yacc* source file rather than the **y.tab.c** file. This option is useful, however, for symbolic debugging, since some symbolic debuggers may be confused by line numbers that are not in order.

The **–N**<*secondary*><*n*> option allows the sizes of certain internal *yacc* tables to be reset. *Secondary* is one of the letters from the set { **a m s p n e c l w** } and specifies the table; *n* is the new size. Tables that can be reset by using secondary letters are as follows:

a	a-array size; default is 12 000.
m	mem array size; default is 12 000.
s	number of states; default is 1000.
p	number of productions; default is 800.
n	number of non-terminals; default is 600.
e	temp-space size; default is 1250.
c	name-space size; default is 5000.
l	look-ahead set table size; default is 650.
w	working set table size; default is 650.

If an array overflows, *yacc* issues a fatal error message including a suggestion of which table to reset. For example:

too many states, try –Ns option

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **–t** option is used, this debugging code will be compiled by default. Independent of whether the **–t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

EXTERNAL INFLUENCES

Environment Variables

LC_CTYPE determines the classification of characters as letters and digits for name fields.

LANG determines the language in which messages are displayed.

International Code Set Support

Single-byte character code sets are supported.

ERRORS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp,	
yacc.acts, yacc.debug	temporary files
/usr/lib/yaccpar	parser prototype for C programs

WARNINGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

The maximum number of terminal symbols is fixed at 2000 and cannot be reset using the **–N** option.

SEE ALSO

lex(1), setlocale(3C), malloc(3X).

YACC – Yet Another Compiler Compiler in *HP-UX Concepts and Tutorials: Programming Environment.*"

LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.

STANDARDS CONFORMANCE

yacc: SVID2, XPG2, XPG3

NAME

yes – be repetitively affirmative

SYNOPSIS

yes [*expletive*]

DESCRIPTION

Yes repeatedly outputs **y**, or if *expletive* is given, the *expletive* is output repeatedly. Termination is by interrupt.

AUTHOR

Yes was developed by the University of California, Berkeley.

Index to Volume 1

Index
Volume 1

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**

Description **Entry Name(Section)**