UNIX®
SYSTEM V
Release 4

Commands Reference Manual

VOLUME 1

(Commands a-l)

for
Motorola Processors

MOTOROLA

UNIX®
SYSTEM V
Release 4

Commands Reference Manual

VOLUME 1

(Commands a-l)

for
Motorola Processors

Published by PTR Prentice-Hall, Inc.
A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

10 9 8 7 6 5 4 3

ISBN 0-13-088832-X

IMPORTANT NOTE TO USERS (USL)

While every effort has been made to ensure the accuracy of all information in this documentation, UNIX System Laboratories, Inc. (USL) assumes no liabilities to any party for any loss or damage caused by errors or omissions or by statements of any kind in this documentation, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause. USL further assumes no liability arising out of the application or use of any product or system described herein, nor any liability for incidental or consequential damages arising from the use of this documentation. USL disclaims all warranties regarding the information contained herein, whether expressed, implied, or statutory, *including implied warranties of merchantability or fitness for a particular purpose.* USL makes no representation that the interconnection of products in the manner described herein will not infringe on existing or future patent rights, nor do the descriptions contained herein imply the granting of any license to make, use or sell equipment constructed in accordance with such descriptions. USL reserves the right to make changes without further notice to any products herein to improve reliability, function, or design.

PRODUCT AVAILABILITY

It is possible that this publication may contain reference to, or information about Motorola products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Motorola intends to announce such Motorola products, programming, or services in your country.

GNU C COMPILER

The GNU C compiler is a product of the Free Software Foundation, Inc. and is subject to the GNU General Public License as published by the Free Software Foundation. You should have received a copy of the GNU General Public License along with the GNU C compiler product; if not, contact:

Free Software Foundation
675 Massachusetts Ave.
Cambridge, Massachusetts 02139
U.S.A.

THIS PROGRAM IS PROVIDED WITHOUT ANY WARRANTY, INCLUDING THE IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Under the General Public License for GNU C you have the freedom to distribute copies of GNU C, obtain source code if you want it, change the software, or use pieces of it in new free programs.

The GNU C compiler has been modified by Motorola, Inc.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013.

MOTOROLA, INC.
Computer Group
2900 South Diablo Way
Tempe, Arizona 85282

TRADEMARKS

Table of Contents

Commands(1) and Miscellaneous Facilities(5)

Introduction

Reference Manuals

Description Manual pages provide technical reference information about the interfaces and execution behavior of each UNIX SYSTEM V Release 4 component.

Organization The *type* of component being described is indicated by the numerical section suffix. Within each section there may be subsections indicated by a single letter. Related sections are organized into reference manuals and alphabetized by name. The following table shows the contents of the reference manuals and their section suffixes.

Title and Contents	Sections
Commands Reference Manual Volumes 1 and 2	
General-purpose user commands	1
Basic networking commands	1C
Form and Menu Language Interpreter (FMLI)	1F
System maintenance commands	1M
Enhanced networking commands	1N
Miscellaneous reference information related to commands.	5
System Calls and Library Functions Reference Manual	
System calls	2
BSD system compatibility library	3
Standard C library	3C
Executable and linking format library	3E

Continued on next page

Reference Manuals, Continued

Contents	Sections
System Calls and Library Functions Reference Manual (continued)	
General-purpose library	3G
Math library	3M
Networking library	3N
Standard I/O library	3S
Specialized library	3X
Miscellaneous reference information related to programming.	5
System Files and Devices Reference Manual	
System file formats	4
Special files (devices)	7
Device Driver Interface/Driver - Kernel Interface Reference Manual	
Driver Data Definitions	D1
Driver Entry Point Routines	D2
Kernel Utility Routines	D3
Kernel Data Structures	D4
Kernel Defines	D5
Master Permuted Index	
Permuted index of all manual pages	All

Retitled Reference Manuals

Background Four reference manuals for this release have been restructured and/or retitled to more accurately describe their contents. The following table shows these changes.

Previous Titles	Current Titles	Current Sections
User's Reference Manual/ System Administrator's Reference Manual (Commands a - l) (Commands m - z)	*Commands Reference Manual (Volume 1, a - l) (Volume 2, m - z)*	1, 1C, 1F, 1M, 1N, 5
Programmer's Reference Manual: Operating System API Part 1: Programming Commands and System Calls Part 2: Functions	*System Calls and Library Functions Reference Manual*	2, 3, 3C, 3E, 3G, 3M, 3N, 3S, 3X, 5
System Files and Devices Reference Manual	*System Files and Devices Reference Manual* (section 5 removed)	4, 7
Permuted Index	*Master Permuted Index*	All

Manual Page Format

Main headings used

All UNIX manual pages have a common format. The following main headings are used:

Heading	Section Contents
NAME	Name of the component and brief statement of its purpose
SYNOPSIS	Syntax of the component
DESCRIPTION	General discussion of functionality
EXAMPLE	Example(s) of usage
FILES	File names built into the component
SEE ALSO	Cross-references to related components

Note: Not all manual pages use all headings.

Typographical Conventions

Style and conventions used

The following typographical and formatting conventions are used.

Convention	Indicates ...
`Constant width`	a literal that should be entered just as it appears
Italic	a substitutable argument
Square brackets around an argument [ ]	an optional argument
name or *file*	a file name
Ellipses ...	previous argument may be repeated
Argument beginning with - minus + plus = equal	a flag argument

Permuted Index

Definition

A permuted index is an alphabetical listing of all the keywords in the **NAME** line of a manual page.

Certain common words are not considered keywords and are not recognized. In the example below, the common words *of*, *to*, and *the* are not recognized.

Example

The **NAME** line of the `adjtime`(2) manual page appears below.

> **adjtime(2)** **adjtime(2)**
>
> **NAME**
>
> `adjtime`- correct the time to allow synchronization of the system clock

The `adjtime`(2) entries from the permuted index are shown below. These entries appear in the a, c, and s sections of the permuted index respectively.

Remainder of NAME line	Keyword and NAME line	Manual Page
synchronization of the system/	adjtime correct the time to allow.....	adjtime(2)
clock adjtime correct the time to	allow synchronization of the system...	adjtime(2)
allow synchronization of the system	clock adjtime correct the time to...	adjtime(2)
synchronization of the/ adjtime	correct the time to allow.....	adjtime(2)
adjtime correct the time to allow	synchronization of the system clock...	adjtime(2)
to allow synchronization of the	system clock / correct the time.....	adjtime(2)

Continued on next page

Permuted Index, Continued

How a permuted index is constructed

The center column lists each keyword followed by all or a portion of the **NAME** line, as space permits. The left column lists the remainder of the **NAME** line. The right column indicates the manual page being referenced.

Omitted words are indicated with a slash (/).

Identification of entries

Manual page entries are identified with their section suffixes shown in parentheses.

Example: `man`(1) and `man`(5)

Section suffixes eliminate confusion caused by duplication of names among the sections.

Master Permuted Index

Each reference manual has a permuted index for the manual pages contained in that book.

The *Master Permuted Index* covers all the manual pages of this documentation library.

Request for Comment

Description

A Request for Comment (RFC) is a document that describes some aspect of networking technology. The RFCs cited in the **SEE ALSO** section of these manual pages are available in hard copy for a small fee from:

Network Information System Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
415-859-6387 fax: 415-859-6028
email:`nisc@nisc.sri.com`

Online versions of RFCs

Online versions of the RFCs are available by `ftp` from `nic.ddn.mil`.To retrieve an on-line RFC, do the following:

Step	Action

1	Connect to the RFC host by entering:
	
	`ftp nic.ddn.mil`
	`user name: anonymous`
	`password: guest`

2	Retrieve the RFC by entering:
	`get rfc/rfc`*num*
	
	`where` *num* `is the number of the RFC`
	
	Example:
	`get rfc:rfc1171.txt`

3	End the `ftp` session by entering: `quit`

NAME

intro - introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands, including user commands, programming commands and commands used chiefly for maintenance and administration (1M commands).

Because of command restructuring for the Virtual File System architecture, there are several instances of multiple manual pages with the same name. For example, there are four manual pages called mount(1M). In each such case the first of the multiple pages describes the syntax and options of the generic command, that is, those options applicable to all FSTypes (file system types). The succeeding pages describe the functionality of the FSType-specific modules of the command. These pages all display the name of the FSType to which they pertain centered and in parentheses at the top of the page. Note that the administrator should not attempt to call these modules directly. The generic command provides a common interface to all of them. Thus the FSType-specific manual pages should not be viewed as describing distinct commands, but rather as detailing those aspects of a command that are specific to a particular FSType.

Manual Page Command Syntax

Unless otherwise noted, commands described in the **SYNOPSIS** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [*-option*...] [*cmdarg*...]
where:

[ ]	Surround an *option* or *cmdarg* that is not required.
. . .	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "-".) *noargletter*... or, *argletter optarg*[,...]
noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "-" (Rule 5, below).
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or separated by white space and quoted (Rule 8, below).
cmdarg	Path name (or other command argument) *not* beginning with "-", or "-" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. getopts(1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command

itself.

1. Command names (*name* above) must be between two and nine characters long.
2. Command names must include only lower-case letters and digits.
3. Option names (*option* above) must be one character long.
4. All options must be preceded by "-".
5. Options with no arguments may be grouped after a single "-".
6. The first option-argument (*optarg* above) following an option must be preceded by white space.
7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (for example, `-o xxx,z,yy` or `-o "xxx z yy"`).
9. All options must precede operands (*cmdarg* above) on the command line.
10. "--" may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. "-" preceded and followed by white space should only be used to mean standard input.

SEE ALSO

`getopts`(1), `exit`(2), `wait`(2), `getopt`(3C).

How to Get Started in the "Introduction" to this document

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 [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, or bad or inaccessible data. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

NOTES

Throughout the manual pages there are references to *TMPDIR, BINDIR, INCDIR,* and *LIBDIR*. These represent directory names whose value is specified on each manual page as necessary. For example, *TMPDIR* might refer to `/var/tmp`. These are not environment variables and cannot be set. [There is an environment variable called `TMPDIR` which can be set. See `tmpnam`(3S).] There are also references to *LIBPATH*, the default search path of the link editor and other tools.

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.

NAME

intro - introduction to miscellany

DESCRIPTION

This section describes miscellaneous information related to commands.

NAME

accept, reject - accept or reject print requests

SYNOPSIS

accept *destinations*
reject [-r *reason*] *destinations*

DESCRIPTION

accept allows the queueing of print requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Run lpstat -a to find the status of *destinations*.

reject prevents queueing of print requests for the named *destinations*. A *destination* can be either a printer or a class of printers. (Run lpstat -a to find the status of *destinations*.) The following option is useful with reject.

-r *reason* Assign a *reason* for rejection of requests. This *reason* applies to all *destinations* specified. *Reason* is reported by lpstat -a. It must be enclosed in quotes if it contains blanks. The default reason is unknown reason for existing destinations, and new destination for destinations just added to the system but not yet accepting requests.

FILES

/var/spool/lp/*

SEE ALSO

lpadmin(1M), lpsched(1M)
enable(1), lp(1), lpstat(1).

NAME

acct: acctdisk, acctdusg, accton, acctwtmp closewtmp, utmp2wtmp - overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [-u *file*] [-p *file*]

/usr/lib/acct/accton [*file*]

/usr/lib/acct/acctwtmp "*reason*"

/usr/lib/acct/closewtmp

/usr/lib/acct/utmp2wtmp

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. acctsh(1M) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into /var/adm/wtmp, as described in utmp(4). The programs described in acctcon(1M) convert this file into session and charging records, which are then summarized by acctmerg(1M).

Process accounting is performed by the UNIX system kernel. Upon termination of a process, one record per process is written to a file (normally /var/adm/pacct). The programs in acctprc(1M) summarize this data for charging purposes; acctcms(1M) is used to summarize command usage. Current process data may be examined using acctcom(1).

Process accounting and connect time accounting (or any accounting records in the tacct format described in acct(4)) can be merged and summarized into total accounting records by acctmerg (see tacct format in acct(4)). prtacct (see acctsh(1M)) is used to format any or all accounting records.

acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

acctdusg reads its standard input (usually from find / -print) and computes disk resource consumption (including indirect blocks) by login. If -u is given, records consisting of those filenames for which acctdusg charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If -p is given, *file* is the name of the password file. This option is not needed if the password file is /etc/passwd. (See diskusg(1M) for more details.)

accton alone turns process accounting off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records (see acct(2) and acct(4)).

acctwtmp writes a utmp(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned (see utmp(4)). *reason* must be a string of 11 or fewer characters, numbers, $, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp "acctg on" >> /var/adm/wtmp
acctwtmp "acctg off" >> /var/adm/wtmp
```

For each user currently logged on, `closewtmp` puts a false `DEAD_PROCESS` record in the `/var/adm/wtmp` file. `runacct` (see `runacct(1M)`) uses this false `DEAD_PROCESS` record so that the connect accounting procedures can track the time used by users logged on before `runacct` was invoked.

For each user currently logged on, `runacct` uses `utmp2wtmp` to create an entry in the file `/var/adm/wtmp`, created by `runacct`. Entries in `/var/adm/wtmp` enable subsequent invocations of `runacct` to account for connect times of users currently logged in.

FILES

`/etc/passwd`	used for login name to user ID conversions
`/usr/lib/acct`	holds all accounting commands listed in sub-class 1M of this manual
`/var/adm/pacct`	current process accounting file
`/var/adm/wtmp`	login/logoff history file

SEE ALSO

`acctcms`(1M), `acctcom`(1), `acctcon`(1M), `acctmerg`(1M), `acctprc`(1M), `acctsh`(1M), `diskusg`(1M), `fwtmp`(1M), `runacct`(1M), `acct`(2), `acct`(4), `utmp`(4).

NAME

acctcms - command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [-a [-p] [-o]] [-c] [-j] [-n] [-s] [-t] *files*

DESCRIPTION

acctcms reads one or more *files*, normally in the form described in acct(4). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The options are:

- -a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in acctcom(1). Output is normally sorted by total kcore-minutes.
- -c Sort by total CPU time, rather than total kcore-minutes.
- -j Combine all commands invoked only once under "***other".
- -n Sort by number of command invocations.
- -s Any filenames encountered hereafter are already in internal summary format.
- -t Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old (that is, pre-UNIX System V Release 4.0) style acctcms internal summary format records.

The following options may be used only with the -a option.

- -p Output a prime-time-only command summary.
- -o Output a non-prime (offshift) time only command summary.

When -p and -o are used together, a combination prime and non-prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes, which will be split into prime and non-prime.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file ... > today
cp total previoustotal
acctcms -s today previoustotal > total
acctcms -a -s today
```

SEE ALSO

acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NOTES

Unpredictable output results if -t is used on new style internal summary format files, or if it is not used with old style internal summary format files.

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

acctcom [*options*] [*file* . . .]

DESCRIPTION

acctcom reads *file*, the standard input, or /var/adm/pacct, in the form described by acct(4) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE (K), and optionally, F (the fork/exec flag: 1 for fork without exec), STAT (the system exit status), HOG FACTOR, KCORE MIN, CPU FACTOR, CHARS TRNSFD, and BLOCKS READ (total blocks read and written).

A # is prepended to the command name if the command was executed with superuser privileges. If a process is not associated with a known terminal, a ? is printed in the TTYNAME field.

If no *files* are specified, and if the standard input is associated with a terminal or /dev/null (as is the case when using & in the shell), /var/adm/pacct is read; otherwise, the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file /var/adm/pacct is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in /var/adm/pacct*incr*.

The *options* are:

-a Show some average statistics about the processes selected. The statistics will be printed after the output records.

-b Read backwards, showing latest commands first. This option has no effect when the standard input is read.

-f Print the fork/exec flag and system exit status columns in the output. The numeric output for this option will be in octal.

-h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as (total CPU time)/(elapsed time).

-i Print columns containing the I/O counts in the output.

-k Instead of memory size, show total kcore-minutes.

-m Show mean core size (the default).

-r Show CPU factor (user-time/(system-time + user-time)).

-t Show separate system and user CPU times.

-v Exclude column headings from the output.

-l *line* Show only processes belonging to terminal /dev/term/*line*.

-u *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a #, which designates only those processes executed with superuser privileges, or ?, which designates only those processes associated with unknown user IDs.

-g *group*	Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.
-s *time*	Select processes existing at or after *time*, given in the format *hr* [:*min* [:*sec*]].
-e *time*	Select processes existing at or before *time*.
-S *time*	Select processes starting at or after *time*.
-E *time*	Select processes ending at or before *time*. Using the same *time* for both -S and -E shows the processes that existed at *time*.
-n *pattern*	Show only commands matching *pattern* that may be a regular expression as in regcmp(3G), except + means one or more occurrences.
-q	Do not print any output records, just print the average statistics as with the -a option.
-o *ofile*	Copy selected process records in the input data format to *ofile*; suppress printing to standard output.
-H *factor*	Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option -h above.
-O *sec*	Show only processes with CPU system time exceeding *sec* seconds.
-C *sec*	Show only processes with total CPU time (system-time + user-time) exceeding *sec* seconds.
-I *chars*	Show only processes transferring more characters than the cutoff number given by *chars*.

FILES

/etc/passwd
/var/adm/pacct*incr*
/etc/group

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), ps(1), runacct(1M), su(1), acct(2), regcmp(3G), acct(4), utmp(4).

NOTES

acctcom reports only on processes that have terminated; use ps(1) for active processes.

If *time* exceeds the present time, then *time* is interpreted as occurring on the previous day.

NAME

acctcon, acctcon1, acctcon2 - connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon [*options*]

/usr/lib/acct/acctcon1 [*options*]

/usr/lib/acct/acctcon2

DESCRIPTION

acctcon converts a sequence of login/logoff records to total accounting records (see the tacct format in acct(4)). login/logoff records are read from standard input. The file /var/adm/wtmp is usually the source of the login/logoff records, however, because it may contain corrupted records or system date changes, it should first be fixed using wtmpfix. The fixed version of file /var/adm/wtmp can then be redirected to acctcon. The tacct records are written to standard output. Here are the options for acctcon:

-l *file* *file* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hangup, termination of login(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See init(1M) and utmp(4).

-o *file* *file* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

acctcon is a combination of the programs acctcon1 and acctcon2. acctcon1 converts login/logoff records, taken from the fixed /var/adm/wtmp file, to ASCII output. acctcon2 reads the ASCII records produced by acctcon1 and converts them to tacct records. acctcon1 can be used with the -l and -o options, described above, as well as with the following options:

-p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

-t acctcon1 maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The -t flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

EXAMPLES

The acctcon command is typically used as follows:

```
acctcon -l lineuse -o reboots < tmpwtmp > ctacct
```

The acctcon1 and acctcon2 commands are typically used as follows:

```
acctcon1 -l lineuse -o reboots < tmpwtmp | sort +1n +2 > ctmp
acctcon2 < ctmp > ctacct
```

FILES

/var/adm/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), runacct(1M), login(1), acct(2), acct(4), utmp(4).

NOTES

The line usage report is confused by date changes. Use wtmpfix (see fwtmp(1M)), with the /var/adm/wtmp file as an argument, to correct this situation.

NAME

acctdisk, acctdusg, accton, acctwtmp - overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [-u *file*] [-p *file*]

/usr/lib/acct/accton [*file*]

/usr/lib/acct/acctwtmp *reason*

/usr/lib/acct/closewtmp

/usr/lib/acct/utmp2wtmp

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. acctsh(1M) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into /var/adm/utmp, as described in utmp(4). The programs described in acctcon(1M) convert this file into session and charging records, which are then summarized by acctmerg(1M).

Process accounting is performed by the system kernel. Upon termination of a process, one record per process is written to a file (normally /var/adm/pacct). The programs in acctprc(1M) summarize this data for charging purposes; acctcms(1M) is used to summarize command usage. Current process data may be examined using acctcom(1).

Process accounting and connect time accounting (or any accounting records in the tacct format described in acct(4)) can be merged and summarized into total accounting records by acctmerg [see tacct format in acct(4)]. prtacct [see acctsh(1M)] is used to format any or all accounting records.

acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

acctdusg reads its standard input (usually from find / -print) and computes disk resource consumption (including indirect blocks) by login. If -u is given, records consisting of those filenames for which acctdusg charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If -p is given, *file* is the name of the password file. This option is not needed if the password file is /etc/passwd. [See diskusg(1M) for more details.]

accton alone turns process accounting off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records [see acct(2) and acct(4)].

acctwtmp writes a utmp(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned [see utmp(4)]. *reason* must be a string of 11 or fewer characters, numbers, $, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp acctgon >> /var/adm/wtmp
acctwtmp "file acctgoff" >> /var/adm/wtmp
```

For each user currently logged on, `closewtmp` puts a false `DEAD_PROCESS` record in the `/var/adm/wtmp` file. `runacct` [see `runacct(1M)`] uses this false `DEAD_PROCESS` record so that the connect accounting procedures can track the time used by users logged on before `runacct` was invoked.

For each user currently logged on, `runacct` uses `utmp2wtmp` to create an entry in the file `/var/adm/wtmp`, created by `runacct`. Entries in `/var/adm/wtmp` enable subsequent invocations of `runacct` to account for connect times of users currently logged in.

FILES

`/var/adm/passwd`	used for login name to user ID conversions
`/usr/lib/acct`	holds all accounting commands listed in section 1M
`/var/adm/pacct`	current process accounting file
`/var/adm/wtmp`	login/logoff history file

SEE ALSO

`acctcms`(1M), `acctcom`(1), `acctcon`(1M), `acctmerg`(1M), `acctprc`(1M), `acctsh`(1M), `diskusg`(1M), `fwtmp`(1M), `runacct`(1M), `acct`(2), `acct`(4), `utmp`(4).

NAME

acctmerg - merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [-a] [-i] [-p] [-t] [-u] [-v] [*file*] ...

DESCRIPTION

acctmerg reads its standard input and up to nine additional files, all in the tacct format (see acct(4)) or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys. Options are:

- -a Produce output in ASCII version of tacct.
- -i Input files are in ASCII version of tacct.
- -p Print input with no processing.
- -t Produce a single record that totals all input.
- -u Summarize by user ID, rather than user ID and name.
- -v Produce output in verbose ASCII format, with more precise notation for floating-point numbers.

EXAMPLES

The following sequence is useful for making "repairs" to any file kept in this format:

```
acctmerg -v <file1 > file2
```

Edit *file2* as desired ...

```
acctmerg -i <file2 > file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NAME

acctprc, acctprc1, acctprc2 - process accounting

SYNOPSIS

/usr/lib/acct/acctprc
/usr/lib/acct/acctprc1 [*ctmp*]
/usr/lib/acct/acctprc2

DESCRIPTION

acctprc reads standard input, in the form described by acct(4), and converts it to total accounting records (see the tacct record in acct(4)). acctprc divides CPU time into prime time and non-prime time and determines mean memory size (in memory segment units). acctprc then summarizes the tacct records, according to user IDs, and adds login names corresponding to the user IDs. The summarized records are then written to standard output. acctprc1 reads input in the form described by acct(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If *ctmp* is given, it is expected to contain a list of login sessions sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file, just as acctprc does. The information in *ctmp* helps it distinguish between different login names sharing the same user ID.

From standard input, acctprc2 reads records in the form written by acctprc1, summarizes them according to user ID and name, then writes the sorted summaries to the standard output as total accounting records.

EXAMPLES

The acctprc command is typically used as shown below:

```
acctprc < /var/adm/pacct > ptacct
```

The acctprc1 and acctprc2 commands are typically used as shown below:

```
acctprc1 ctmp </var/adm/pacct | acctprc2 >ptacct
```

FILES

/etc/passwd

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NOTES

Although it is possible for acctprc1 to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from cron(1M), for example. A more precise conversion can be done using the acctwtmp program in acct(1M). acctprc does not distinguish between users with identical user IDs.

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor.

NAME

acctsh: chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct - shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee *login-name number*

/usr/lib/acct/ckpacct [*blocks*]

/usr/lib/acct/dodisk [-o] [*files ...*]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct *number*

/usr/lib/acct/nulladm *file*

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [-l] [-c] [*mmdd*]

/usr/lib/acct/prtacct *file* ["*heading*"]

/usr/lib/acct/runacct [*mmdd*] [*mmdd state*]

/usr/lib/acct/shutacct ["*reason*"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | off | switch

DESCRIPTION

chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to /var/adm/fee, to be merged with other accounting records by runacct.

ckpacct should be initiated via cron(1M) to periodically check the size of /var/adm/pacct. If the size exceeds *blocks*, 1000 by default, turnacct will be invoked with argument *switch*. If the number of free disk blocks in the /var file system falls below 500, ckpacct will automatically turn off the collection of process accounting records via the *off* argument to turnacct. When at least 500 blocks are restored, the accounting will be activated again on the next invocation of ckpacct. This feature is sensitive to the frequency at which ckpacct is executed, usually by cron.

dodisk should be invoked by cron to perform the disk accounting functions. By default, it will use diskusg (see diskusg(1M)) to do disk accounting on the S5 file system in /etc/vfstab. If the -o flag is used, it will use acctdusg (see acct(1M)) to do a slower version of disk accounting by login directory. *files* specifies the one or more filesystem names where disk accounting will be done. If *files* are used, disk accounting will be done on these filesystems only. If the -o flag is used, *files* should be mount points of mounted filesystems. If the -o option is omitted, *files* should be the special file names of mountable filesystems.

lastlogin is invoked by runacct to update /var/adm/acct/sum/loginlog, which shows the last date on which each person logged in.

monacct should be invoked once each month or each accounting period. *number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01-12). This default is useful if monacct is to executed via cron(1M) on the first day of each month. monacct creates summary files in

/var/adm/acct/fiscal and restarts the summary files in /var/adm/acct/sum.

nulladm creates *file* with mode 664 and ensures that owner and group are adm. It is called by various accounting shell procedures.

prctmp can be used to print the session record file (normally /var/adm/acct/nite/ctmp created by acctcon1 (see acctcon(1M)).

prdaily is invoked by runacct to format a report of the previous day's accounting data. The report resides in /var/adm/acct/sum/rprt/mmdd where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing prdaily. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The -l flag prints a report of exceptional usage by login id for the specified date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of monacct. The -c flag prints a report of exceptional resource usage by command, and may be used on current day's accounting data only.

prtacct can be used to format and print any total accounting (tacct) file.

runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see runacct(1M).

shutacct is invoked during a system shutdown to turn process accounting off and append a "reason" record to /var/adm/wtmp.

startup can be invoked when the system is brought to a multi-user state to turn process accounting on.

turnacct is an interface to accton (see acct(1M)) to turn process accounting on or off. The switch argument moves the current /var/adm/pacct to the next free name in /var/adm/pacct*incr* (where *incr* is a number starting with 1 and incrementing by one for each additional pacct file), then turns accounting back on again. This procedure is called by ckpacct and thus can be taken care of by the cron and used to keep pacct to a reasonable size. shutacct uses turnacct to stop process accounting. startup uses turnacct to start process accounting.

FILES

/var/adm/fee	accumulator for fees
/var/adm/pacct	current file for per-process accounting
/var/adm/pacct*incr*	used if pacct gets large and during execution of daily accounting procedure
/var/adm/wtmp	login/logoff summary
/usr/lib/acct/ptelus.awk	contains the limits for exceptional usage by login ID
/usr/lib/acct/ptecms.awk	contains the limits for exceptional usage by command name
/var/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in section 1M of this manual

`/var/adm/acct/sum`	summary directory contains information for `monacct`
`var/adm/acct/fiscal`	fiscal reports directory

SEE ALSO

`acct`(1M), `acctcms`(1M), `acctcom`(1), `acctcon`(1M), `acctmerg`(1M), `acctprc`(1M), `cron`(1M), `diskusg`(1M), `fwtmp`(1M), `runacct`(1M), `acct`(2). `acct`(4), `utmp`(4)

NAME

addbib - create or extend a bibliographic database

SYNOPSIS

/usr/ucb/addbib [-a] [-p *promptfile*] *database*

DESCRIPTION

When addbib starts up, answering y to the initial Instructions? prompt yields directions; typing n or RETURN skips them. addbib then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to *database.* A null response (RETURN) means to leave out that field. A '-' (minus sign) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating Continue? prompt allows the user either to resume by typing y or RETURN, to quit the current session by typing n or q, or to edit *database* with any system editor (vi, ex, ed).

The following options are available:

-a Suppress prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL-D.

-p *promptfile*
Use a new prompting skeleton, defined in *promptfile*. This file should contain prompt strings, a TAB, and the key-letters to be written to the *database*.

USAGE

Bibliography Key Letters

The most common key-letters and their meanings are given below. addbib insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

%A	Author's name
%B	Book containing article referenced
%C	City (place of publication)
%D	Date of publication
%E	Editor of book containing article referenced
%F	Footnote number or label (supplied by refer(1))
%G	Government order number
%H	Header commentary, printed before reference
%I	Issuer (publisher)
%J	Journal containing article
%K	Keywords to use in locating reference
%L	Label field used by -k option of refer(1)
%M	Bell Labs Memorandum (undefined)
%N	Number within volume
%O	Other commentary, printed at end of reference

%P	Page number(s)
%Q	Corporate or Foreign Author (unreversed)
%R	Report, paper, or thesis (unpublished)
%S	Series title
%T	Title of article or book
%V	Volume number
%X	Abstract — used by roffbib, not by refer
%Y,Z	Ignored by refer

SEE ALSO

ed(1), ex(1), indxbib(1), lookbib(1), refer(1), roffbib(1), sortbib(1), vi(1).

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] *files*

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 (that begin with -) and named files (note that SCCS filenames 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 unchanged.

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 listed below. Each argument is explained as if only one named file were to be processed because the effect of each argument applies 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 filename is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, the SCCS file is created empty. 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. Characters from supplementary code sets can be used for the *name* of the file from which the text is to be taken. The file may also include characters from supplementary code sets.

-r*rel* The *rel* (release) 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 filename must also be supplied. In the case of existing SCCS files: (1) a -t keyletter without a filename causes removal of the descriptive text (if any) that is currently in the SCCS file, and (2) a -t keyletter with a filename causes text (if any) in the named file to replace the descriptive

text (if any) that is currently in the SCCS file. Characters from supplementary code sets can be used for the *name* of the file from which the text is to be taken. The file may also include characters from supplementary code sets.

`-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` command to create branch deltas.

`c`*ceil* The highest release (that is, ceiling): a number greater than 0 but less than or equal to 9999 that may be retrieved by a `get` command for editing. The default value for an unspecified `c` flag is 9999.

`f`*floor* The lowest release (that is, floor): a number greater than 0 but less than 9999 that may be retrieved by a `get` 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` command.

`i`[*str*] Causes the `No id keywords (ge6)` message issued by `get` or `delta` 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. The string must contain a keyword, and no embedded newlines.

`j` Allows concurrent `get` commands for editing on the same SID of an SCCS file. This flag 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.

`n` Causes `delta` to create a null delta in each of those releases (if any) being skipped when a delta is made in a new release (for example, 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 nonexistent 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. Characters from supplementary code sets can be used for the substitution text *text*.

m*mod* *mod* (module) name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by get. If the m flag is not specified, the value assigned is the name of the SCCS file with the leading s. removed. Characters from supplementary code sets can be used for the module name *mod*.

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

v[*pgm*] Causes delta 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)]. This program will receive as arguments the module name, the value of the type flag (see t*type* above), and the *mrlist*. (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 in a single admin command. See the -f keyletter for allowable *flag* names.

(l*list* used with -d indicates 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 UNIX System group ID, to be added to the list of users who 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 logins or numerical group IDs as desired may be on the list simultaneously. If the list of users is empty, 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.

-m[*mrlist*] The list of Modification Request (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to delta. 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.

-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`. Omission of the `-y` keyletter results in a default comment line being inserted.

  The `-y` keyletter is valid only if the `-i` and/or `-n` keyletters are specified (that is, a new SCCS file is being created). The *comment* text including characters from supplementary code sets can be inserted into the SCCS file as a comment.

`-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 to the file, nullifying the effect of any other keyletters supplied; therefore, it is 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.

The last component of all SCCS filenames must be of the form `s.`*file*. 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*, [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 renaming process 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.

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

FILES

x-file	see `delta`(1)
z-file	see `delta`(1)
`bdiff`	program to compute differences between the "gotten" file and the g-file [see `get`(1)]

INTERNATIONAL FUNCTIONS

Characters from supplementary code sets can be used for SCCS filenames, *files*. However, they must begin with the `s.` ASCII characters. SCCS files may also include characters from supplementary code sets.

SEE ALSO

`bdiff`(1), `delta`(1), `ed`(1), `get`(1), `help`(1), `prs`(1), `what`(1), `sccsfile`(4).

DIAGNOSTICS

Use the `help` command for explanations.

NOTES

If it is necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of a text editor. You must run `admin -h` on the edited file 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.

NAME

alpq - query the ALP *STREAMS* module

SYNOPSIS

`alpq`

DESCRIPTION

The `alpq` command takes no arguments or options. It presents, on its standard output, a list of the functions currently registered with the `alp` *STREAMS* module. Information on building and using these functions is contained in the manual entry `alp`(7).

The output list contains entries like the following:

```
1  Ucase            (Upper to lower case converter)
```

The first field is a sequence number. The second field is the function's name (by which it may be accessed), and the third field is the function's explanation string, enclosed in parentheses.

CAVEATS

The `alpq` command works by pushing the `alp` *STREAMS* module querying it via `ioctl`(2) and then popping it immediately; its standard input (normally the user's tty) must thus be a *STREAM*.

SEE ALSO

kbdcomp(1M), kbdload(1M), alp(7), kbd(7).

NAME

apropos - locate commands by keyword lookup

SYNOPSIS

/usr/ucb/apropos *keyword* . . .

DESCRIPTION

apropos shows which manual sections contain instances of any of the given keywords in their title. Each word is considered separately and the case of letters is ignored. Words which are part of other words are considered; thus, when looking for 'compile', apropos will find all instances of 'compiler' also.

Try

```
apropos password
```

and

```
apropos editor
```

If the line starts '*filename*(*section*) . . .' you can do 'man *section filename*' to get the documentation for it. Try

```
apropos format
```

and then

```
man 3s printf
```

to get the manual page on the subroutine printf.

apropos is actually just the -k option to the man(1) command.

FILES

/usr/share/man/whatis data base

SEE ALSO

man(1), whatis(1), catman(1M)

NAME

ar - maintain portable archive or library

SYNOPSIS

ar [-V] - *key* [*arg*] [*posname*] *afile* [*name*...]

DESCRIPTION

The ar command maintains groups of files combined into a single archive file. Its main use is to create and update library files. However, it can be used for any similar purpose. The magic string and the file headers used by ar consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When ar creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure are described in detail in ar(4). The archive symbol table [described in ar(4)] is used by the link editor ld to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by ar when there is at least one object file in the archive. 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 command is used to create or update the contents of such an archive, the symbol table is rebuilt. The s option described below will force the symbol table to be rebuilt.

The -V option causes ar to print its version number on standard error.

Unlike command options, the *key* is a required part of the ar command line. The *key* is formed with one of the following letters: drqtpmx. Arguments to the *key*, alternatively, are made with one of more of the following set: vuaibcls. *posname* is an archive member name used as a reference point in positioning other files in the archive. *afile* is the archive file. The *name*s are constituent files in the archive file. The meanings of the *key* characters are as follows:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character u is used with r, then 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, then the *posname* argument must be present and specifies that new files are to be placed after (a) or before (b or i) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a large archive piece-by-piece.

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

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in r, specifies where the files are to be moved.

`x` Extract the named files. If no names are given, all files in the archive are extracted. In neither case does `x` alter the archive file.

The meanings of the other key arguments are as follows:

`v` 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`, give a long listing of all information about the files. When used with `x`, print the filename preceding each extraction.

`c` Suppress the message that is produced by default when *afile* is created.

`l` This option is obsolete. It is recognized, but ignored, and will be removed in the next release.

`s` Force the regeneration of the archive symbol table even if `ar`(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the `strip`(1) command has been used on the archive.

SEE ALSO

`ld`(1), `lorder`(1), `strip`(1), `a.out`(4), `ar`(4)

NOTES

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

Since the archiver no longer uses temporary files, the `-l` option is obsolete and will be removed in the next release.

By convention, archives are suffixed with the characters `.a`.

NAME

arch - display the architecture of the current host

SYNOPSIS

/usr/ucb/arch

DESCRIPTION

The arch command displays the architecture of the current host system.

SEE ALSO

mach(1), uname(1).

NAME

arp - address resolution display and control

SYNOPSIS

arp *hostname*

arp -a [*unix* [*kmem*]]

arp -d *hostname*

arp -s *hostname ether_address* [temp] [pub] [trail]

arp -f *filename*

DESCRIPTION

The arp program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol [arp(7)].

With no flags, the program displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation.

The following options are available:

-a Display all of the current ARP entries by reading the table from the file *kmem* (default /dev/kmem) based on the kernel file *unix* (default /stand/unix).

-d Delete an entry for the host called *hostname*. This option may only be used by the super-user.

-s Create an ARP entry for the host called *hostname* with the Ethernet address *ether_address*. The Ethernet address is given as six hexadecimal bytes separated by colons. The entry will be permanent unless the word temp is given in the command. If the word pub is given, the entry will be published, for instance, this system will respond to ARP requests for *hostname* even though the hostname is not its own. The word trail indicates that trailer encapsulations may be sent to this host.

-f Read the file named *filename* and set multiple entries in the ARP tables. Entries in the file should be of the form

hostname ether_address [temp] [pub] [trail]

with argument meanings as given above.

SEE ALSO

ifconfig(1M), arp(7)

NAME

as - assembler

SYNOPSIS

as [*options*] *file*

DESCRIPTION

The as command creates object files from assembly language source *files*. The following flags may be specified in any order:

-o *objfile*	Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the .s suffix, if there is one, from the input file name and appending a .o suffix.
-n	Turn off long/short address optimization. By default, address optimization takes place.
-m	Run the m4 macro processor on the input to the assembler.
-R	Remove (unlink) the input file after assembly is completed.
-dl	Obsolete. Assembler issues a warning saying that it is ignoring the -dl option.
-T	Accept obsolete assembler directives.
-V	Write the version number of the assembler being run on the standard error output.
-Q{y | n}	If -Qy is specified, place the version number of the assembler being run in the object file. The default is -Qn.
-Y [md],*dir*	Find the m4 preprocessor (m) and/or the file of predefined macros (d) in directory *dir* instead of in the customary place.

FILES

By default, as creates its temporary files in /var/tmp. This location can be changed by setting the environment variable TMPDIR [see tempnam in tmpnam(3S)].

SEE ALSO

cc(1), ld(1), m4(1), nm(1), strip(1), tmpnam(3S), a.out(4)

NOTES

If the -m (m4 macro processor invocation) option is used, keywords for m4 [see m4(1)] cannot be used as symbols (variables, functions, labels) in the input file since m4 cannot determine which keywords are assembler symbols and which keywords are real m4 macros.

The .align assembler directive may not work in the .text section when long/short address optimization is performed.

Arithmetic expressions may only have one forward referenced symbol per expression.

Whenever possible, you should access the assembler through a compilation system interface program such as cc.

NAME

ascii - map of ASCII character set

DESCRIPTION

ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. It contains:

```
|000 nul |001 soh |002 stx |003 etx |004 eot |005 enq |006 ack |007 bel |
|010 bs  |011 ht  |012 nl  |013 vt  |014 np  |015 cr  |016 so  |017 si  |
|020 dle |021 dc1 |022 dc2 |023 dc3 |024 dc4 |025 nak |026 syn |027 etb |
|030 can |031 em  |032 sub |033 esc |034 fs  |035 gs  |036 rs  |037 us  |
|040 sp  |041 !   |042 "   |043 #   |044 $   |045 %   |046 &   |047 ´   |
|050 (   |051 )   |052 *   |053 +   |054 ,   |055 -   |056 .   |057 /   |
|060 0   |061 1   |062 2   |063 3   |064 4   |065 5   |066 6   |067 7   |
|070 8   |071 9   |072 :   |073 ;   |074 <   |075 =   |076 >   |077 ?   |
|100 @   |101 A   |102 B   |103 C   |104 D   |105 E   |106 F   |107 G   |
|110 H   |111 I   |112 J   |113 K   |114 L   |115 M   |116 N   |117 O   |
|120 P   |121 Q   |122 R   |123 S   |124 T   |125 U   |126 V   |127 W   |
|130 X   |131 Y   |132 Z   |133 [   |134 \   |135 ]   |136 ^   |137 _   |
|140 `   |141 a   |142 b   |143 c   |144 d   |145 e   |146 f   |147 g   |
|150 h   |151 i   |152 j   |153 k   |154 l   |155 m   |156 n   |157 o   |
|160 p   |161 q   |162 r   |163 s   |164 t   |165 u   |166 v   |167 w   |
|170 x   |171 y   |172 z   |173 {   |174 |   |175 }   |176 ~   |177 del |

| 00 nul | 01 soh | 02 stx | 03 etx | 04 eot | 05 enq | 06 ack | 07 bel |
| 08 bs  | 09 ht  | 0a nl  | 0b vt  | 0c np  | 0d cr  | 0e so  | 0f si  |
| 10 dle | 11 dc1 | 12 dc2 | 13 dc3 | 14 dc4 | 15 nak | 16 syn | 17 etb |
| 18 can | 19 em  | 1a sub | 1b esc | 1c fs  | 1d gs  | 1e rs  | 1f us  |
| 20 sp  | 21 !   | 22 "   | 23 #   | 24 $   | 25 %   | 26 &   | 27 ´   |
| 28 (   | 29 )   | 2a *   | 2b +   | 2c ,   | 2d -   | 2e .   | 2f /   |
| 30 0   | 31 1   | 32 2   | 33 3   | 34 4   | 35 5   | 36 6   | 37 7   |
| 38 8   | 39 9   | 3a :   | 3b ;   | 3c <   | 3d =   | 3e >   | 3f ?   |
| 40 @   | 41 A   | 42 B   | 43 C   | 44 D   | 45 E   | 46 F   | 47 G   |
| 48 H   | 49 I   | 4a J   | 4b K   | 4c L   | 4d M   | 4e N   | 4f O   |
| 50 P   | 51 Q   | 52 R   | 53 S   | 54 T   | 55 U   | 56 V   | 57 W   |
| 58 X   | 59 Y   | 5a Z   | 5b [   | 5c \   | 5d ]   | 5e ^   | 5f _   |
| 60 `   | 61 a   | 62 b   | 63 c   | 64 d   | 65 e   | 66 f   | 67 g   |
| 68 h   | 69 i   | 6a j   | 6b k   | 6c l   | 6d m   | 6e n   | 6f o   |
| 70 p   | 71 q   | 72 r   | 73 s   | 74 t   | 75 u   | 76 v   | 77 w   |
| 78 x   | 79 y   | 7a z   | 7b {   | 7c |   | 7d }   | 7e ~   | 7f del |
```

FILES

/usr/pub/ascii

NAME

at, batch - execute commands at a later time

SYNOPSIS

at [-f *script*] [-m] *time* [*date*] [+ *increment*]
at -l [*job* . . .]
at -r *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 should be executed, while jobs queued with batch will execute when system load level permits. at may be used with the following options:

-f *script* Reads commands to be executed from the named *script* file.

-l [*job*] Reports all jobs scheduled for the invoking user, or just the *job*s specified.

-m Sends mail to the user after the job has been completed, indicating that the job is finished, even if the job produces no output. Mail is sent only if the job has not already generated a mail message.

-r *job* Removes specified *job*s previously scheduled using at.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use at if their name appears in the file /usr/sbin/cron.d/at.allow. If that file does not exist, the file /usr/sbin/cron.d/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 and is empty, global usage is permitted. The allow/deny files consist of one user name per line. These files can only be modified by the privileged user.

If the DATEMSK environment variable is set, it points to a template file that at will use to determine the valid *time* and *date* values instead of the values described below. For more information about using DATEMSK, see the last paragraph of the DESCRIPTION section.

time may be specified as follows, where *h* is hours and *m* is minutes: *h*, *hh*, *hhmm*, *h*:*m*, *h*:*mm*, *hh*:*m*, *hh*:*mm*. A 24-hour clock is assumed, unless am or pm is appended to *time*. If zulu is appended to *time*, it means Greenwich Mean Time (GMT). *time* can also take on the values: noon, midnight, and now. at now responds with the error message too late; use now with the *increment* argument, such as: at now + 1 minute.

An optional *date* may be specified as either a month name followed by a day number (and possibly a year number preceded by a comma) or a day of the week. (Both the month name and the day of the week may be spelled out or abbreviated to three characters.) Two special "days", today and tomorrow are recognized. If no *date* is given, today is assumed if the given hour is greater than the current hour

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.

The optional *increment* is simply a number suffixed by one of the following: minutes, hours, days, weeks, months, or years. (The singular form is also accepted.) The modifier next may precede the *increment;* it means "+ 1."

Thus valid commands include:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at now next day
at 5 pm Friday
```

at and batch write the job number and schedule time to standard error.

at -r removes jobs previously scheduled by at or batch. The job number is the number returned to you previously by the at or batch command. You can also get job numbers by typing at -l. You can only remove your own jobs unless you are the privileged user.

If the environment variable DATEMSK is set, at will use its value as the full path name of a template file containing format strings. The strings consist of field descriptors and text characters and are used to provide a richer set of allowable date formats in different languages by appropriate settings of the environment variable LANG or LC_TIME (see *environ*(5)). (See *getdate*(3C) for the allowable list of field descriptors; this list is a subset of the descriptors allowed by calendar(1) that are listed on the *date*(1) manual page.) The formats described above for the *time* and *date* arguments, the special names noon, midnight, now, next, today, tomorrow, and the *increment* argument are not recognized when DATEMSK is set.

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.

This sequence can be used at a terminal:

```
batch
sort filename > outfile
```
CTRL-d (hold down 'control' and depress 'd')

This sequence, which shows redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
sort filename 2>&1 > outfile | mail loginid
!
```

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 example shows the possible contents of a template file AT.TEMPL in /var/tmp.

```
%I %p, the %est of %B of the year %Y run the following job
%I %p, the %end of %B of the year %Y run the following job
%I %p, the %erd of %B of the year %Y run the following job
%I %p, the %eth of %B of the year %Y run the following job
%d/%m/%y
%H:%M:%S
%I:%M%p
```

The following are examples of valid invocations if the environment variable DATEMSK is set to /var/tmp/AT.TEMPL.

```
at 2 PM, the 3rd of July of the year 2000 run the following job
at 3/4/99
at 10:30:30
at 2:30PM
```

FILES

/usr/sbin/cron.d	main cron directory
/usr/sbin/cron.d/at.allow	list of allowed users
/usr/sbin/cron.d/at.deny	list of denied users
/usr/sbin/cron.d/queuedefs	scheduling information
/var/spool/cron/atjobs	spool area

SEE ALSO

atq(1), atrm(1), calendar(1), cron(1M), crontab(1), date(1), kill(1), mail(1), nice(1), ps(1), sh(1), sort(1), getdate(3C), environ(5).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

atq - display the jobs queued to run at specified times

SYNOPSIS

atq [-c] [-n] [*username...*]

DESCRIPTION

atq displays the current user's queue of jobs submitted with at to be run at a later date. If invoked by the privileged user, atq will display all jobs in the queue.

If no options are given, the jobs are displayed in chronological order of execution.

When a privileged user invokes atq without specifying *username*, the entire queue is displayed; when a *username* is specified, only those jobs belonging to the named user are displayed.

The atq command can be used with the following options:

-c Display the queued jobs in the order they were created (that is, the time that the at command was given).

-n Display only the total number of jobs currently in the queue.

FILES

/var/spool/cron spool area

SEE ALSO

at(1), atrm(1), cron(1M).

NAME

atrm - remove jobs spooled by at or batch

SYNOPSIS

atrm [-afi] *arg* . . .

DESCRIPTION

atrm removes delayed-execution jobs that were created with the at(1) command, but not yet executed. The list of these jobs and associated job numbers can be displayed by using atq(1).

arg a user name or job-number. atrm removes each job-number you specify, and/or all jobs belonging to the user you specify, provided that you own the indicated jobs.

Jobs belonging to other users can only be removed by the privileged user.

The atrm command can be used with the following options:

-a All. Remove all unexecuted jobs that were created by the current user. If invoked by the privileged user, the entire queue will be flushed.

-f Force. All information regarding the removal of the specified jobs is suppressed.

-i Interactive. atrm asks if a job should be removed. If you respond with a y, the job will be removed.

FILES

/var/spool/cron spool area

SEE ALSO

at(1), atq(1), cron(1M).

NAME

automount - automatically mount NFS file systems

SYNOPSIS

automount [-nTv] [-D *name=value*] [-M *mount-directory*]
[-t *sub-options*] [*directory map* [*-mount-options*]] . . .

DESCRIPTION

automount is a daemon that automatically and transparently mounts an NFS file system as needed. It monitors attempts to access directories that are associated with an automount map, along with any directories or files that reside under them. When a file is to be accessed, the daemon mounts the appropriate NFS file system. You can assign a map to a directory using an entry in a direct automount map, or by specifying an indirect map on the command line.

automount uses a map to locate an appropriate NFS file server, exported file system, and mount options. It then mounts the file system in a temporary location, and replaces the file system entry for the directory or subdirectory with a symbolic link to the temporary location. If the file system is not accessed within an appropriate interval (five minutes by default), the daemon unmounts the file system and removes the symbolic link. If the indicated directory has not already been created, the daemon creates it, and then removes it upon exiting.

Since the name-to-location binding is dynamic, updates to an automount map are transparent to the user. This obviates the need to pre-mount shared file systems for applications that have hard coded references to files.

If you specify the dummy directory /-, automount treats the *map* argument that follows as the name of a direct map. In a direct map, each entry associates the full pathname of a mount point with a remote file system to mount.

If the directory argument is a pathname, the *map* argument points to a file called an indirect map. An indirect map contains a list of the subdirectories contained within the indicated directory. With an indirect map, it is these subdirectories that are mounted automatically. The *map* argument must be a full pathname.

The *-mount-options* argument, when supplied, is a comma-separated list of mount(1M) options, preceded by a hyphen (-). If mount options are specified in the indicated map, however, those in the map take precedence.

The following options are available:

-n Disable dynamic mounts. With this option, references through the automount daemon only succeed when the target file system has been previously mounted. This can be used to prevent NFS servers from cross-mounting each other.

-T Trace. Expand each NFS call and display it on the standard output.

-v Verbose. Log status messages to the console.

-D *name=value*
Assign *value* to the indicated automount (environment) variable.

-M *mount-directory*
Mount temporary file systems in the named directory, instead of /tmp_mnt.

`-t` *sub-options*
Specify *sub-options* as a comma-separated list that contains any combination of the following:

`l` *duration*
Specify a *duration*, in seconds, that a file system is to remain mounted when not in use. The default is 5 minutes.

`m` *interval*
Specify an *interval*, in seconds, between attempts to mount a file system. The default is 30 seconds.

`w` *interval*
Specify an *interval*, in seconds, between attempts to unmount file systems that have exceeded their cached times. The default is 1 minute.

ENVIRONMENT

Environment variables can be used within an `automount` map. For instance, if `$HOME` appeared within a map, `automount` would expand it to its current value for the `HOME` variable.

If a reference needs to be protected from affixed characters, enclose the variable name within braces.

USAGE

Direct/Indirect Map Entry Format

A simple map entry (mapping) takes the form:

`directory` [*-mount-options*] *location* . . .

where `directory` is the full pathname of the directory to mount when used in a direct map, or the basename of a subdirectory in an indirect map. *mount-options* is a comma-separated list of `mount` options, and *location* specifies a remote file system from which the directory may be mounted. In the simple case, *location* takes the form:

host : *pathname*

Multiple *location* fields can be specified, in which case `automount` sends multiple `mount` requests; `automount` mounts the file system from the first host that replies to the `mount` request. This request is first made to the local net or subnet. If there is no response, any connected server may respond.

If *location* is specified in the form:

host : *path* : *subdir*

host is the name of the host from which to mount the file system, *path* is the pathname of the directory to mount, and *subdir*, when supplied, is the name of a subdirectory to which the symbolic link is made. This can be used to prevent duplicate mounts when multiple directories in the same remote file system may be accessed. With a map for `/home` such as:

```
able  homeboy:/home/homeboy:able
baker homeboy:/home/homeboy:baker
```

and a user attempting to access a file in /home/able, automount mounts homeboy:/home/homeboy, but creates a symbolic link called /home/able to the able subdirectory in the temporarily mounted file system. If a user immediately tries to access a file in /home/baker, automount needs only to create a symbolic link that points to the baker subdirectory; /home/homeboy is already mounted. With the following map:

```
able  homeboy:/home/homeboy/able
baker homeboy:/home/homeboy/baker
```

automount would have to mount the file system twice.

A mapping can be continued across input lines by escaping the NEWLINE with a backslash. Comments begin with a # and end at the subsequent NEWLINE.

Directory Pattern Matching

The & character is expanded to the value of the directory field for the entry in which it occurs. In this case:

```
able  homeboy:/home/homeboy:&
```

the & expands to able.

The * character, when supplied as the directory field, is recognized as the catch-all entry. Such an entry resolves to any entry not previously matched. For instance, if the following entry appeared in the indirect map for /home:

```
*     &:/home/&
```

this would allow automatic mounts in /home of any remote file system whose location could be specified as:

hostname:/home/*hostname*

Hierarchical Mappings

A hierarchical mapping takes the form:

directory [/ [*subdirectory*]] [*-mount-options*] *location* ...
[/ [*subdirectory*] [*-mount-options*] *location* ...].

The initial /[*subdirectory*] is optional for the first location list and mandatory for all subsequent lists. The optional *subdirectory* is taken as a filename relative to the directory. If *subdirectory* is omitted in the first occurrence, the / refers to the directory itself.

Given the direct map entry:

```
/arch/src   \
/         -ro,intr  arch:/arch/src          alt:/arch/src   \
/1.0      -ro,intr  alt:/arch/src/1.0       arch:/arch/src/1.0   \
/1.0/man  -ro,intr  arch:/arch/src/1.0/man  alt:/arch/src/1.0/man
```

automount would automatically mount /arch/src, /arch/src/1.0 and /arch/src/1.0/man, as needed, from either arch or alt, whichever host responded first.

Direct Maps

A direct map contains mappings for any number of directories. Each directory listed in the map is automatically mounted as needed. The direct map as a whole is not associated with any single directory.

Indirect Maps

An indirect map allows you to specify mappings for the subdirectories you wish to mount under the `directory` indicated on the command line. It also obscures local subdirectories for which no mapping is specified. In an indirect map, each `directory` field consists of the basename of a subdirectory to be mounted as needed.

Included Maps

The contents of another map can be included within a map with an entry of the form

+mapname

where *mapname* is a filename.

Special Maps

The `-null` map is the only special map currently available. The `-null` map, when indicated on the command line, cancels a previous map for the directory indicated.

FILES

`/tmp_mnt`	parent directory for dynamically mounted file systems

SEE ALSO

`df`(1M), `mount`(1M), `passwd`(4)

NOTES

When it receives signal number 1, `automount` rereads the `/etc/mnttab` file to update its internal record of currently-mounted file systems. If a file system mounted with `automount` is unmounted by a `umount` command, `automount` should be forced to reread the file.

Shell filename expansion does not apply to objects not currently mounted.

Since `automount` is single-threaded, any request that is delayed by a slow or non-responding NFS server will delay all subsequent automatic mount requests until it completes.

Programs that read `/etc/mnttab` and then touch files that reside under automatic mount points will introduce further entries to the file.

NAME

autopush - configure lists of automatically pushed STREAMS modules

SYNOPSIS

```
autopush -f file
autopush -r -M major -m minor
autopush -g -M major -m minor
```

DESCRIPTION

This command allows one to configure the list of modules to be automatically pushed onto the stream when a device is opened. It can also be used to remove a previous setting or get information on a setting.

The following options apply to autopush:

-f This option sets up the autopush configuration for each driver according to the information stored in the specified file. An autopush file consists of lines of at least four fields each where the fields are separated by a space as shown below:

```
maj_  min_  last_min_  mod1 mod2 ... modn
```

The first three fields are integers that specify the major device number, minor device number, and last minor device number. The fields following represent the names of modules. If *min_* is -1, then all minor devices of a major driver specified by *maj_* are configured and the value for *last_min_* is ignored. If *last_min_* is 0, then only a single minor device is configured. To configure a range of minor devices for a particular major, *min_* must be less than *last_min_*.

The last fields of a line in the autopush file represent the list of module names where each is separated by a space. The maximum number of modules that can be automatically pushed on a stream is defined to be eight. The modules are pushed in the order they are specified. Comment lines start with a # sign.

-r This option removes the previous configuration setting of the particular *major* and *minor* device number specified with the -M and -m options respectively. If the values of *major* and *minor* correspond to a setting of a range of minor devices, where *minor* matches the first minor device number in the range, the configuration would be removed for the entire range.

-g This option gets the current configuration setting of a particular *major* and *minor* device number specified with the -M and -m options respectively. It will also return the starting minor device number if the request corresponds to a setting of a range (as described with the -f option).

SEE ALSO

streamio(7).

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [-Fc] [*prog*] [*parameters*] [*files*]

DESCRIPTION

awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as -f *file*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

Parameters, in the form x=... y=... etc., may be passed to *awk.*

Files are read in order; if there are no files, the standard input is read. The file name - means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS; see 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 ; conditional4; expression ) 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 whole 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 (").

The *print* statement prints its arguments on the standard output (or on 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 in the printf(3S) manpage.

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions `exp`, `log`, `sqrt`, and *int*. The last truncates its argument to an integer; *substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. 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 are as in `egrep`(1). 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 an occurrence of the first pattern and the next 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 `~` (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 may be used to capture control before the first input line is read and after the last. 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 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`).

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*

SEE ALSO

`grep`(1), `lex`(1), `nawk`(1), `sed`(1), `printf`(3S).

NOTES

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.

NAME

backup - initiate or control a system backup session

SYNOPSIS

backup -i [-t *table*] [-o *name*] [-m *user*] [-ne] [-s | -v] [-c *week:day* | demand]

backup [-a] [-t *table*] [-o *name*] [-m *user*] [-ne] [-c *week:day* | demand]

backup -S | -R | -C [-u *user* | -A | -j *jobid*]

DESCRIPTION

Without options, the backup command performs all backup operations specified for the current day and week of the backup rotation in the backup register. This set of backup operations is considered a single job and is assigned a backup job id which can be used to control the progress of the session. As backup operations are processed, their status is tracked [See bkstatus(1M)]. As backup operations are completed, they are recorded in the backup history log.

backup may only be executed by a user with superuser privilege.

A backup job can be controlled in three ways. It can be canceled, suspended or resumed (after being suspended).

Modes of Operator Intervention

Backup operations may require operator intervention to perform such tasks as inserting volumes into devices or confirming proper volume labels. backup provides three modes of operator interaction.

backup with no options assumes that an operator is present, but not at the terminal where the backup command was issued. This mode sends a mail message to the operator. The mail identifies the device requiring service and the volume required. The operator reads the mail message, invokes the bkoper command, responds to the prompts, and the backup operation continues.

backup -i establishes interactive mode, which assumes that an operator is present at the terminal where the backup command was issued. In this mode, bkoper is automatically invoked at the terminal where the backup command was entered. The operator responds to the prompts as they arrive.

backup -a establishes automatic mode, which assumes that no operator is available. In this mode, any backup operation that requires operator intervention fails. Backups that can be satisfied by mounted volume proceed.

Register Validations

A number of backup service databases must be consistent before the backups listed in a backup register can be performed. These consistencies can only be validated at the time backup is initiated. If any of them fail, backup will terminate. Invoking backup -ne performs the validation checks in addition to displaying the set of backup operations to be performed. The validations are:

1. The backup method must be a default method or be an executable file in /bkup/method .
2. The dependencies for an entry are all defined in the register. Circular dependencies (eg., entry abc depends on entry def; entry def depends on entry abc) are allowed.

3. The device group for a destination must be defined in the device group table, `/dgroup.tab` [See *"Device Management"*).

Options

`-a` Initiates all backup operations in automatic mode; does not prompt an operator to service media.

`-c` *week:day* | `demand`
Selects from the backup register only those backup operations for the specified week and day of the backup rotation, instead of the current day and week of the rotation. If `demand` is specified, selects only those backup operations scheduled to be performed on demand.

`-e` This option displays an estimate of the number of volumes required to perform each backup operation.

`-i` Selects interactive operation

`-j` *jobid* Controls only the backup job identified by *jobid*. *jobid* is a `backup` job id.

`-m` *user* Sends mail to the named *user* when all backup operations for the backup job are complete.

`-n` Displays the set of backup operations that would be performed but does not actually perform the backup operations. The display is ordered according to the dependencies and priorities specified in the backup register.

`-o` *name* Initiates backup operations only on the named originating object. *name* is an item in the following form:
oname | *odevice*

`-s` Displays a "." for each 100 (512-byte) blocks transferred to the destination device. The dots are displayed while each backup operation is progressing.

`-t` *table* Initiates backup operations described in the specified backup register instead of the default register, `etc/bkup/bkreg.tab` . *table* is a backup register.

`-u` *user* Controls backup jobs started by the named *user* instead of those started by the user invoking the command. *user* is a valid login id.

`-v` While each backup operation is progressing, display the name of each file or directory as soon as it has been transferred to the destination device.

`-A` Controls backup jobs for all users instead of those started by the user invoking the command.

`-C` Cancels backup jobs.

`-R` Resumes suspended backup jobs.

`-S` Suspends backup jobs.

DIAGNOSTICS

The exit codes for the `backup` command are the following:

0 = successful completion of the task
1 = one or more parameters to backup are invalid.
2 = an error has occurred which caused backup to fail to complete *all* portions of its task.

EXAMPLES

Example 1:

```
backup -i -v -c 2:1 -m admin3
```

initiates those backups scheduled for Monday of the second week in the rotation period instead of backups for the current day and week. Performs the backup in interactive mode and displays on standard output the name of each file, directory, file system slice, or data slice as soon as it is transferred to the destination device. When all backups are completed, sends mail notification to the user with login id admin3.

Example 2:

```
backup -o /usr
```

initiates only those backups from the *usr* file system that is mounted on the originating device /dev/rdsk/m328_c1d0s2 and is labeled usr.

Example 3:

```
backup -S
```

Suspends the backup jobs requested by the invoking user.

Example 4:

```
backup -R -j back-359
```

resumes the backup operations included in backup job id back-359.

FILES

/etc/bkup/method/*
/etc/bkup/bkreg.tab
/etc/device.tab
/etc/dgroup.tab

SEE ALSO

bkhistory(1M), bkoper(1M), bkreg(1M), bkstatus(1M)

NAME

banner - make posters

SYNOPSIS

banner *strings*

DESCRIPTION

banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

WARNING

Non-ASCII characters specified in *strings* will not be output correctly.

SEE ALSO

echo(1)

NAME

basename, dirname - deliver 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 substitution marks (` `) within shell procedures. The *suffix* is a pattern as defined on the ed(1) manual page.

dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, invoked with the argument /home/sms/personal/mail sets the environment variable NAME to the file named mail and the environment variable MYMAILPATH to the string /home/sms/personal.

```
NAME=`basename $HOME/personal/mail`
MYMAILPATH=`dirname $HOME/personal/mail`
```

This shell procedure, invoked with the argument /usr/src/bin/cat.c, compiles the named file and moves the output to cat in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

SEE ALSO

ed(1), sh(1)

NAME

basename - display portions of pathnames

SYNOPSIS

/usr/ucb/basename *string* [*suffix*]

DESCRIPTION

basename deletes any prefix ending in '/' and the *suffix*, if present in *string*. It directs the result to the standard output, and is normally used inside substitution marks (` `) within shell procedures. The *suffix* is a pattern as defined on the ed(1) manual page.

EXAMPLE

This shell procedure invoked with the argument /usr/src/bin/cat.c compiles the named file and moves the output to cat in the current directory:

```
cc $1
mv a.out `basename $1 .c`
```

SEE ALSO

ed(1), sh(1).

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. bc is actually a preprocessor for the desk calculator program dc, which it invokes automatically unless the -c option is present. In this case the dc input is sent to the standard output instead. The options are as follows:

-c Compile only. The output is sent to the standard output.

-l Argument stands for the name of an arbitrary precision math library.

The syntax for bc programs is as follows: *L* means letter a-z, *E* means expression, *S* means statement.

Comments
are enclosed in /* and */.

Names
simple variables: *L*
array elements: *L* [*E*]
the words ibase, obase,and scale

Other operands
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*)

Operators
+ - * / % ^
(% is remainder; ^ is power)
++ -- (prefix and postfix; apply to names)
== <= >= != < >
= =+ =- =* =/ =% =^

Statements
E
{ *S* ; . . . ; *S* }
if (*E*) *S*
while (*E*) *S*
for (*E* ; *E* ; *E*) *S*
null statement
break
quit

Function definitions

```
define L ( L , ... , L ) {
        auto L , ... , L
"       S" ; ... S
        return ( E )
}
```

Functions in `-l` math library

`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.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to `scale` influences the number of digits to be retained on arithmetic operations in the manner of `dc`. Assignments to `ibase` or `obase` set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. 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.

EXAMPLE

```
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.

FILES

`/usr/lib/lib.b`	mathematical library

`/usr/bin/dc` desk calculator proper

SEE ALSO

`dc`(1)

NOTES

The `bc` command does not recognize the logical operators `&&` and `||`.

The `for` statement must have all three expressions (*E*'s).

The `quit` statement is interpreted when read, not when executed.

NAME

bdiff - big diff

SYNOPSIS

bdiff *file1 file2* [*n*] [-s]

DESCRIPTION

bdiff is used in a manner analogous to diff to find which lines in *file1* and *file2* must be changed to bring the files into agreement. Its purpose is to allow processing of files too large for diff. If *file1 (file2)* is -, the standard input is read.

Valid options to bdiff are:

n The number of line 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.

-s Specifies that no diagnostics are to be printed by bdiff (silent option). Note, however, that this does not suppress possible diagnostic messages from diff, which bdiff calls.

bdiff ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes diff on corresponding segments. 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)

NAME

bfs - big file scanner

SYNOPSIS

bfs [-] *file*

DESCRIPTION

The bfs command is similar to ed except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including newline, per line (255 for 16-bit machines). 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 the csplit command 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. Messages are given in response to errors if prompting is turned on.

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, 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 both print the first ten lines. The f command only prints the name of the file being scanned; there is no remembered filename. 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. The xf commands may be nested to a depth of 10.

xn
: List the marks currently in use (marks are set by the k command).

xo [*file*]
: Further output from the p and null commands is diverted to the named *file*, which, if necessary, is created with mode 666 (readable and writable by everyone), unless your umask setting dictates otherwise; see umask(1). 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 newline, 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 does not 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 following command is an unconditional jump:

```
xb/^/ label
```

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.

`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`. The commands `xv5100` or `xv5 100` both assign the value `100` to the variable `5`. The command `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` prints the first 100 lines:

```
1,%5p
1,%5
%6
```

The following globally searches for the characters `100` and prints each line containing a match:

```
g/%5/p
```

To escape the special meaning of `%`, a `\` must precede it.

```
g/".*\%[cds]/p
```

could be used to match and list lines containing a `printf` of characters, decimal integers, or strings.

Another feature of the `xv` command is that the first line of output from a UNIX system command can be stored into a variable. The only requirement is that the first character of *value* be an `!`. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

puts the current line into variable `5`, prints it, and increments 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 test the last saved *return code* from the execution of a UNIX system command (`!`*command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string `size`.

```
xv55
: l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
xv45
: l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz l
```

`xc` [*switch*]

If *switch* is `1`, output from the `p` and null commands is crunched; if *switch* is `0` it is not. 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.

INTERNATIONAL FUNCTIONS

`bfs` can process characters from supplementary code sets in the text as well as ASCII characters.

`bfs` can also recognize *labels* containing characters from supplementary code sets for `:`, `xb`, `xbn` and `xbz` commands.

Regular expression searches are performed on characters, not on individual bytes. Refer to `ed`(1).

The value designated by *number* with the `xt` command must be the number of displayed columns, not the number of characters.

Marks set by the k command must be ASCII characters in the range of *a* to z, and all 26 marks are remembered.

Size Indication
The size of the file displayed at first and after read/write by the e or w commands is in bytes, not characters.

SEE ALSO

csplit(1), ed(1), umask(1), regexp(5).

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

NAME

biff - give notice of incoming mail messages

SYNOPSIS

/usr/ucb/biff [y | n]

DESCRIPTION

biff turns mail notification on or off for the terminal session. With no arguments, biff displays the current notification status for the terminal.

The y option allows mail notification for the terminal. The n option disables notification for the terminal.

If notification is allowed, the terminal rings the bell and displays the header and the first few lines of each arriving mail message. biff operates asynchronously. For synchronized notices, use the MAIL variable of sh(1) or the mail variable of csh(1).

A 'biff y' command can be included in your ~/.login or ~/.profile file for execution when you log in.

FILES

~/.login
~/.profile

SEE ALSO

csh(1), mail(1), sh(1).

NAME

`binmail` - an early program for processing mail messages

SYNOPSIS

`/usr/ucblib/binmail` [`-ipq`] [`-f` *filename*] *address*
`/usr/ucblib/binmail` *recipient ...*

DESCRIPTION

This is the old version 7 UNIX system mail program. The default mail command, `/bin/mail` is described in `mail`(1).

`binmail` is installed on the system to facilitate the local delivery of mail for `sendmail`. It is intended to be used only by `sendmail`. It is not suitable for use by users.

SEE ALSO

`mail`(1), `sendmail`(1M), `vacation_bsd`(1).

NAME

biod - NFS daemon

SYNOPSIS

biod [*nservers*]

DESCRIPTION

biod starts *nservers* asynchronous block I/O daemons. This command is used on an NFS client to buffer read-ahead and write-behind. Four is the usual number for *nservers*.

The biod daemons are automatically invoked in run level 3.

SEE ALSO

mountd(1M), nfsd(1M), sharetab(4)

NAME

bkexcept - change or display an exception list for incremental backups

SYNOPSIS

bkexcept [-t *file*] [-d *patterns*]
bkexcept [-t *file*] -a|-r *patterns*
bkexcept -C [*files*]

DESCRIPTION

The bkexcept command displays a list of patterns describing files that are to be excluded when backup operations occur using incfile. The list is known as the "exception list."

bkexcept may be executed only by a user with superuser privilege.

bkexcept -a adds patterns to the list.

bkexcept -d displays patterns from the list.

bkexcept -r removes patterns from the list.

Patterns

Patterns describe individual pathnames or sets of pathnames. Patterns must conform to pathname naming conventions specified under **DEFINITIONS** on the intro(2) page. A pattern is taken as a filename and is interpreted in the manner of cpio. A pattern can include the shell special characters *, ?, and []. Asterisk (*) and question mark (?) will match period (.) and slash(/). Because these are shell special characters, they must be escaped on the command line.

There are three general methods of specifying entries to the exception list:

- To specify all files under a particular directory, specify the directory name (and any desired subdirectories) followed by an asterisk:

  */directory/subdirectories/**

- To specify all instances of a filename regardless of its location, specify the filename preceded by an asterisk:

  /filename

- To specify one instance of a particular file, specify the entire pathname to the file:

  /directory/subdirectories/filename

If *pattern* is a dash (-), standard input is read for a list of patterns (one per line until EOF) to be added or deleted.

Compatibility

Prior versions of the backup service created exception lists using ed syntax. bkexcept -C provides a translation facility for exception lists created by ed. The translation is not perfect; not all ed patterns have equivalents in cpio. For those patterns that have no automatic translation, an attempt at translation is made, and the translated version is flagged with the word QUESTIONABLE. The exception list translation is directed to standard output. Redirect the standard output to a translation file, review the contents of the translation file (correcting entries that were not translated properly and deleting the QUESTIONABLE flags), and then use the resulting file as input to a subsequent bkexcept -a. For example, if the translated

file was named `checkfile` the `-a` option would appear as follows:

```
bkexcept -a - < checkfile
```

Options

`-t` *file* The filename used in place of the default file.

`-a` *pattern...*

Adds *pattern* to the exception list where *pattern* is one or more patterns (comma-separated or blank-separated and enclosed in quotes) describing sets of paths.

`-d` *pattern...*

Displays entries in the exception list. If *pattern* begins with a slash (/), `-d` displays all entries whose names begin with *pattern*. If *pattern* does not begin with a slash, `-d` displays all entries that include *pattern* anywhere in the entry. If *pattern* is a dash (-), input is taken from standard input. *pattern* is not a pattern -- it matches patterns. *pattern* `a*b` matches `/a*b` but does not match `/adb`. For files containing a carriage return, a null exception list is returned. For files of zero length (no characters), an error is returned (search of table failed).

The entries are displayed in ASCII collating sequence order (special characters, numbers, then alphabetical order).

`-r` *pattern...*

Removes *pattern* from the exception list. *pattern* is one or a list of patterns (comma-separated or blank-separated and enclosed in quotes) describing sets of paths. *pattern* must be an exact match of an entry in the exception list for *pattern* to be removed. Patterns that are removed are echoed to standard output, `stdout`.

`-C` [*files*]

Displays on standard output the translation of each *file* (a prior version's exception list) to the new syntax. Each *file* contains `ed` patterns, one per line.

If *file* is omitted, the default UNIX exception list, `/etc/save.d/except`, is translated. If *file* is a dash (-), input is taken from standard input, one per line.

DIAGNOSTICS

The exit codes for the `bkexcept` command are the following:

`0` = the task completed successfully
`1` = one or more parameters to `bkexcept` are invalid
`2` = an error has occurred, causing `bkexcept` to fail to complete all portions of its task

EXAMPLES

Example 1:

```
bkexcept -a /tmp/*,/var/tmp/*,/usr/rje/*,*/trash,
```

adds the four sets of files to the exception list, (all files under `/tmp`, all files under `/var/tmp`, all files under `/usr/rje`, and any file on the system named `trash`).

Example 2:

```
bkexcept -d /tmp
```

displays the following patterns from those added to the exception list in Example 1.

```
/tmp/*
```

```
bkexcept -d tmp
```

displays the following patterns from those added to the exception list in Example 1.

```
/tmp/*, /var/tmp/*
```

displays one per line, with a heading.

Example 3:

```
bkexcept -r /var/tmp/*,/usr/rje/*
```

removes the two patterns from the exception list.

Example 4:

```
bkexcept -C /save.d/old.except > trans.except
```

translates the file `/save.d/old.except` from its `ed` format to `cpio` format and sends the translations to the file `trans.except`. The translations of `/save.d/old.except` may be added to the current exception list by using `bkexcept -a` as follows:

```
bkexcept -a - < trans.except
```

FILES

`/etc/bkup/bkexcept.tab`	the default exception list for UNIX System V Release 4.
`/etc/save.d/except`	the default exception list for pre-UNIX System V Release 4.

SEE ALSO

`backup`(1M), `cpio`(1), `ed`(1), `incfile`(1M), `sh`(1), `intro`(2).

NAME

bkhistory - report on completed backup operations

SYNOPSIS

bkhistory [-hl] [-f *field_separator*] [-d *dates*] [-o *names*] [-t *tags*]
bkhistory -p *period*

DESCRIPTION

bkhistory without options reports a summary of the contents of the backup history log, bkhist.tab. Backup operations are sorted alphabetically by tag. For each tag, operations are listed from most to least recent. backup(1M) updates this log after each successful backup operation.

bkhistory may be executed only by a user with the superuser privilege.

bkhistory -p assigns a rotation *period* (in weeks) for the history log; all entries older than the specified number of weeks are deleted from the log. The default rotation period is one (1) week.

Options

-d *dates*
: Restricts the report to backup operations performed on the specified dates. *dates* are in the date format. *day, hour, minute,* and *year,* are optional and will be ignored. The list of *dates* is either comma-separated or blank-separated and surrounded by quotes.

-f *field_separator*
: Suppresses field wrap on the display and specifies an output field separator to be used. The value of *c* is the character that will appear as the field separator on the display output. For clarity of output, do not use a separator character that is likely to occur in a field. For example, do not use the colon as a field separator character if the display will contain dates that use a colon to separate hours from minutes. To use the default field separator (tab), specify the null character ("") for *c*.

-h
: Suppresses header for the reports.

-l
: Displays a long form of the report. This produces an ls -l listing of the files included in the backup archive (if backup tables of contents are available on-line).

-o *names*
: Restricts the report to the specified originating objects (file systems or data slices). *names* is a list of *onames* and/or *odevices*. [See bkreg(1M)].

  The list of names is either comma-separated or blank-separated and surrounded by quotes.

-p *period*
: Sets the number of weeks of information that will be saved in the backup history table. The minimum value of *period* is 1, which is also the default value. the size of int. By default, *period* is 1.

-t *tags*
: Restricts the report to backups with the specified *tags*. *tags* is a list of tag values as specified in the backup register. The list of *tags* is either comma-separated or blank-separated and surrounded by quotes.

DIAGNOSTICS

The exit codes for the `bkhistory` command are the following:

`0` =	the task completed successfully
`1` =	one or more parameters to `bkhistory` are invalid
`2` =	an error has occurred, causing `bkhistory` to fail to complete all portions of its task

EXAMPLES

Example 1:

```
bkhistory -p 3
```

sets the rotation period for the history log to three weeks. Entries older than three weeks are deleted from the log.

Example 2:

```
bkhistory -t SpoolDai,UsrDaily,TPubsWed
```

displays a report of completed backup operations for the three tags listed.

Example 3:

```
bkhistory -l -o /usr
```

Displays an `ls -l` listing of the files that were backed up from `/usr` (the originating object) if there is a table of contents.

FILES

`/etc/bkup/bkhist.tab`	the backup history log that contains information about successfully completed backup operations
`/etc/bkup/bkreg.tab`	description of the backup policy established by the administrator
`/var/sadm/bkup/toc`	list of directories with on-line tables of contents

SEE ALSO

`backup`(1M), `bkreg`(1M), `date`(1), `ls`(1).

NAME

bkoper - interact with backup operations to service media insertion prompts

SYNOPSIS

bkoper [-u *users*]

DESCRIPTION

Backup operations may require an operator to insert media and to confirm proper volume labels. The bkoper command provides a mailx-like interface for these operator interactions. It begins by printing a list of headers. Each header describes a backup operation requiring interaction, the device requiring attention including the media type and label of the volume to be inserted (see EXAMPLE). The system displays prompts and the operator issues commands to resolve the backup operation. Typing a carriage return invokes the current header. If no headers have been serviced, the current header is the first header on the list. If a header has been selected and serviced, the current header is the next one following.

bkoper may be executed only by a user with superuser privilege. By default, the operator may interact only with backup operations that were started by the same user ID .

If the -u *users* option is given, the operator interacts only with backup operations started by the specified *user*(s).

Commands

!*shell-command*
: Escapes to the shell. The remainder of the line after the ! is sent to the UNIX system shell (sh) to be interpreted as a command.

=
: Prints the current backup operation number.

?
: Prints this summary of commands.

[p|t] [*n*]
: Both the p and t options operate in the same way. Either option will interact with the backup operation described by the *n*'th header. *n* defaults to the current header number.

h
: Prints the list of backup operations.

q
: Quits from bkoper.

DIAGNOSTICS

The exit codes for bkoper are the following:

0 = successful completion of the task
1 = one or more parameters to bkoper are invalid.
2 = an error has occurred which caused bkoper to fail to complete *all* portions of its task.

EXAMPLE

A sample header is shown below. Items appearing in the header are listed in the following order: header number, job-ID, tag, originating device, destination group, destination device, destination volume labels. [See bkreg(1M) for descriptions of items.] Not every header contains values for all these fields; if a destination group is not specified in /etc/bkup/bkreg.tab, then no value for "destination group" appears in the header.

```
1 back-111 usrsun /dev/dsk/c1d0s1 disk /dev/dsk/c2d1s9 usrsave
2 back-112 fs2daily /dev/dsk/c1d0s8 ctape /dev/ctape/c4d0s2 -
```

Backup headers are numbered on the basis of arrival; the oldest header has the lowest number. If the destination device does not have a volume label, a dash is displayed in the header.

SEE ALSO

`bkreg`(1M), `bkstatus`(1M), `getvol`(1M), `mailx`(1)

NAME

bkreg - change or display the contents of a backup register

SYNOPSIS

bkreg -p *period* [-w *cweek*] [-t *table*]

bkreg -a *tag* -o *orig* -c *weeks:days*|demand -d *ddev* -m *method*|migration [-b *moptions*] [-t *table*] [-D *depend*] [-P *prio*]

bkreg -e *tag* [-o *orig*] [-c *weeks:days*|demand] [-m *method*|migration] [-d *ddev*] [-t *table*] [-b *moptions*] [-D *depend*] [-P *prio*]

bkreg -r *tag* [-t *table*]

bkreg [-A|-O|-R] [-hsv] [-t *table*] [-c *weeks*[:*days*]|demand]

bkreg -C *fields* [-hv] [-t *table*] [-c *weeks*[:*days*]|demand] [-f *c*]

DESCRIPTION

A backup register is a file containing descriptions of backup operations to be performed on a system. The default backup register is located in /etc/bkup/bkreg.tab. Other backup registers may be created.

The bkreg command may be executed only by a user with superuser privilege.

Each entry in a backup register describes backup operations to be performed on a given disk object (called the originating object) for some set of days and weeks during a rotation period. There may be several register entries for an object, but only one entry may specify backup operations for an object on a specific day and week of the rotation period. The entry describes the object, the backup method to be used to archive the object, and the destination volumes to be used to store the archive. Each entry has a unique *tag* that identifies it. *Tag*s must conform to file naming conventions.

Rotation Period

Backups are performed in a rotation period specified in weeks. When the end of a rotation period is reached, a new period begins. Rotation periods begin on Sundays. The default rotation period is one week.

Originating Objects

An originating object is either a raw data slice or a filesystem. An originating object is described by its originating object name, its device name, and optional volume labels.

Several backup operations for different originating objects may be active concurrently by specifying priorities and dependencies. During a backup session, higher priority backup operations are attempted before lower priority backup operations. All backup operations of a given priority may proceed concurrently unless dependencies are specified. If one backup is declared to be dependent on others, it will not be started until all of its antecedents have completed successfully.

Destination Devices

Each backup archive is written to a set of storage volumes inserted into a destination device. A destination device can have destination device group, a destination device name, media characteristics, and volume labels. Default characteristics for a medium (as specified in the device table) may be overridden.

Backup Methods

An originating object is backed up to a destination device archive using a method. The method determines the amount of information backed up and the representation of that information. Different methods may be used for a given originating object on different days of the rotation. Each method accepts a set of options that are specific to the method.

Several default methods are provided with the Backup service. Others methods may be added by a system site. For descriptions of the default methods, see `incfile`(1M), `ffile`(1M), `fdisk`(1M), `fimage`(1M), and `fdp`(1M).

A backup archive may be migrated to a different destination by specifying `migration` as the backup method. The device name of the originating object for a migration must have been the destination device for a previously successful backup operation. This form of backup does not re-archive the originating object. It copies an archive from one destination to another, updating the backup service's databases so that restores can still be done automatically.

Register Validations

There are items in a single backup register entry and items across register entries that must be consistent for the backup service to conduct a backup session correctly. Some of these consistencies are checked at the time the backup register is created or changed. Others can be checked only at the time the backup register is used by `backup`(1M). See `backup`(1M) for a complete list of validations.

Modes

The `bkreg` command has two modes: changing the contents of a backup register and displaying the contents of a backup register.

Changing Contents

`bkreg -p` changes the rotation period for a backup register. The default rotation period is one week.

`bkreg -a` adds an entry to a backup register. This option requires other options to be specified. These are listed below under **Options**.

`bkreg -e` edits an existing entry in a backup register.

`bkreg -r` removes an existing entry from a backup register.

Displaying Contents

`bkreg -C` produces a customized display of the contents of a backup register.

`bkreg [-A|-R|-O]`
produces a summary display of the contents of a backup register.

Options

`-a` Adds a new entry to the default backup register. Options required with `-a` are: *tag, originating device, weeks:days, destination device,* and *method*. If other options are not specified, the following defaults are used: the default backup register is used, no method options are specified, the priority is 0, and no dependencies exist between entries.

`-b` *moptions*
Each backup method supports a specific set of options that modify its behavior. *moptions* is specified as a list of options that are blank-separated and enclosed in quotes. The argument string provided here is passed to the

method exactly as entered, without modification. See the following entries for a listing of options. `fdisk`(1M), `fdp`(1M), `ffile`(1M), `fimage`(1M), and `incfile`(1M).

`-c` *weeks*:*days* | `demand`
Sets the week(s) and day(s) of the rotation period during which a backup entry should be performed or for which a display should be generated. *weeks* is a set of numbers including 1 and 52. The value of *weeks* cannot be greater than the value of `-p`*period*. *weeks* is specified as a combination of lists or ranges (either comma-separated or blank-separated and enclosed in quotes). An example set of weeks is

```
''1 3-10,13''
```

indicating the first week, each of the third through tenth weeks, and the thirteenth week of the rotation period.

days is either a set of numbers between 0 (Sunday) and 6 (Saturday), or a set of abbreviations between s (Sunday) and sa (Saturday). In addition, *days* are specified as a combination of lists or ranges (either comma-separated or blank-separated and enclosed in quotes).

`demand` indicates that an entry is used only when explicitly requested by

```
backup -c demand
```

`-d` *ddev*
Specifies *ddev* as the destination device for the backup operation. *ddev* is of the form:

[*dgroup*] [: [*ddevice*] [: *dchar*] [: *dmname*]]

where either *dgroup* or *ddevice* must be specified and *dchar* and *dmname* are optional. (Both *dgroup* and *ddev* may be specified together.) Colons delineate field boundaries and must be included as indicated above.

dgroup is the device group for the destination device. [See `devgroup.tab`(4).] If omitted, *ddevice* must be specified.

ddevice is the device name of a specific destination device. [See `device.tab`(4).] If omitted, *dgroup* must be specified and any available device in *dgroup* may be used.

dchar describes media characteristics. If specified, they override the default characteristics for the device and group. *dchar* is of the form:

`keyword=`*value*

where `keyword` is a valid device characteristic keyword (as it appears in the device table.) *dchar* entries may be separated by commas or blanks. If separated by blanks, the entire string of arguments to *ddev* must be enclosed in quotes.

dlabels is a list of volume names of the destination volumes. The list of *dlabels* must be either comma-separated or blank-separated. If blank-separated, the entire *ddev* argument must be surrounded by quotes. Each *dlabel* corresponds to a *volumename* specified on the `labelit` command. If *dlabels* is omitted, `backup` and `restore` do not validate the volume labels on this entry.

`-e` Edits an existing entry. If any of the options `-b`, `-c`, `-d`, `-m`, `-o`, `-D`, or `-P` are present, they replace the current settings for the specified entry in the register.

`-f` *c* Overrides the default output field separator. *c* is the character that will appear as the field separator on the display output. The default output field separator is colon (`:`).

`-h` Suppresses headers when generating displays.

`-m` *method* | `migration`
Performs the backup using the specified *method.* Default methods are: `incfile`, `ffile`, `fdisk`, `fimage`, and `fdp`. If the method to be used is not a default method, it must appear as the executable file in the standard method directory `/etc/bkup/method`. `migration` indicates that the value of *orig* (following the `-o` option) matches the value of *ddev* during a prior backup operation. The originating object is not rearchived; it is simply copied to the location specified by *ddev* (following the `-d` option). The backup history (if any) and tables of contents (if any) are updated to reflect the changed destination for the original archive.

`-o` *orig*
Specifies *orig* as the originating object for the backup operation. *orig* is specified in the following format:

oname`:`*odevice*`[:`*omname*`]`

where *oname* is the name of an originating object. For file system slices, it is the nodename on which the file system is usually mounted, `mount`. For data slices, it is any valid path name. This value is provided to the backup method and validated by `backup`. The default data slice backup methods, `fdp` and `fdisk`, do not validate this name.

odevice is the device name for the originating object. In all cases, it is a raw disk slice device name. This name is specified in the following format: `/dev/rdsk/m328_c?d?s?`.

olabel is the volume label for the originating object. For file system slices, it corresponds to the *volumename* displayed by the `labelit` command. A data slice may have an associated volume name that appears nowhere except on the outside of the volume (where it is taped); `getvol` may be used to have an operator validate the name.

The special data slice (e.g., `/dev/rdsk/m328_c0d2s7`) names an entire disk and is used when disk formatting or reslicing is done to reference the disk's volume table of contents (VTOC). [See `fmthard`(1M) and `prtvtoc`(1M).] `backup` validates this special full disk slice with the disk volume name specified when the disk was sliced. [See `fmthard`(1M).] If the disk volume name is omitted, `backup` does not validate the volume labels for this originating object.

`-p` *period*
Sets the rotation period (in weeks) for the backup register to *period*. The minimum value is 1; the maximum value is 52. By default the current week of the rotation is set to 1.

`-r` Removes the specified entries from the register.

`-s` Suppresses wrap-around behavior when generating displays. Normal behavior is to wrap long values within each field.

`-t` *table*
Uses *table* instead of the default register, `bkreg.tab`.

`-v` Generates displays using (vertical) columns instead of (horizontal) rows. This allows more information to be displayed without encountering problems displaying long lines.

`-w` *cweek*
Overrides the default behavior by setting the current week of the rotation period to *cweek*. *cweek* is an integer between 1 and the value of *period*. The default is `1`.

`-A` Displays a report describing all fields in the register. The display produced by this option is best suited as input to a filter, since in horizontal mode it produces extremely long lines.

`-C` *fields*
Generates a display of the contents of a backup register, limiting the display to the specified fields. The output is a set of lines, one per register entry. Each line consists of the desired fields, separated by a field separator character. *fields* is a list of field names (either comma-separated or blank-separated and enclosed in quotes) for the fields desired. The valid field names are `period`, `cweek`, `tag`, `oname`, `odevice`, `olabel`, `weeks`, `days`, `method`, `moptions`, `prio`, `depend`, `dgroup`, `ddevice`, `dchar`, and `dlabel`.

`-D` *depend*
Specifies a set of backup operations that must be completed successfully before this operation may begin. *depend* is a list of *tag*(s) (either comma-separated or blank-separated and enclosed in quotes) naming the antecedent backup operations.

`-f` *c* Overrides the default output field separator. *c* is the character that will appear as the field separator on the display output. The default output field separator is colon (":").

`-O` Displays a summary of all originating objects with entries in the register.

`-P` *prio*
Sets a priority of *prio* for this backup operation. The default priority is 0; the highest priority is 100. All backup operations with the same priority may run simultaneously, unless the priority is 0. All backups with priority 0 run sequentially in an unspecified order.

`-R` Displays a summary of all destination devices with entries in the register.

DIAGNOSTICS

The exit codes for `bkreg` are the following:

`0` = the task completed successfully
`1` = one or more parameters to `bkreg` are invalid
`2` = an error has occurred, causing `bkreg` to fail to complete *all* portions of its task

Errors are reported on standard error if any of the following occurs:

1. The `tag` specified in `bkreg -e` or `bkreg -r` does not exist in the backup register.
2. The tag specified in `bkreg -a` already exists in the register.

EXAMPLES

Example 1:

```
bkreg -p 15 -w 3
```

establishes a 15-week rotation period in the default backup register and sets the current week to the 3rd week of the rotation period.

Example 2:

```
bkreg -a acct5 -t wklybu.tab \
-o /usr:/dev/rdsk/m328_c1d0s2:usr -c "2 4-6 8 10:0,2,5" \
-m incfile -b -txE \
-d ctape:capacity=1404:acctwkly1,acctwkly2,acctwkly3 \
```

adds an entry named *acct5* to the backup register named `wklybu.tab`. If `wklybu.tab` does not already exist, it will be created. The originating object to be backed up is the `/usr` file system on the `/dev/rdsk/m328_c1d0s2` device which is known as `usr`. The backup will be performed each Sunday, Tuesday, and Friday of the second, fourth through sixth, eighth, and tenth weeks of the rotation period using the `incfile` (incremental file) method. The method options specify that a table of contents will be created on additional media instead of in the backup history log, the exception list is to be ignored, and an estimate of the number of volumes for the archive is to be provided before performing the backup. The backup will be done to the next available cartridge tape device using the three cartridge tape volumes `acctwkly1`, `acctwkly2`, and `acctwkly3`. These volumes have a capacity of 1404 blocks each.

Example 3:

```
bkreg -e services2 -t wklybu.tab \
-o /back:/dev/rdsk/m328_c1d0s8:back -m migration \
-c demand -d ctape:/dev/rdsk/m328_c4d0s3 \
```

changes the specifications for the backup operation named `services2` on the backup table `wklybu.tab` so that whenever the command `backup -c demand` is executed, the backup that was performed to the destination device `back:dev/rdsk/m328_c1d0s2:back` will be migrated from that device (now serving as the originating device) to a cartridge tape.

Example 4:

```
bkreg -e pubsfri -P 10 -D develfri,marketfri,acctfri
```

changes the priority level for the backup operation named `pubsfri` to 10 and makes this backup operation dependent on the three backup operations `develfri`, `marketfri`, and `acctfri`. The `pubsfri` operation will be done only after all backup operations with priorities greater than 10 have begun and after the `develfri`, `marketfri`, and `acctfri` operations have been completed successfully.

Example 5:

```
bkreg -c 1-8:0-6
```

provides the default display of the contents of the default backup register, for all weekdays for the first through eighth weeks of the rotation period. The information in the register will be displayed in the following format:

```
Rotation Period = 10    Current Week = 4


Originating Device: / /dev/root
Tag       Weeks   Days   Method   Options   Pri   Dgroup
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
rootsp    1-8     0      ffile    -bxt      20    ctape

Originating Device:  /usr /dev/dsk/m328_c1d0s2

Tag       Weeks   Days   Method   Options   Pri   Dgroup
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
usrsp     1-8     0      ffile    -bxt      15    ctape
```

FILES

`/etc/bkup/method/*`	
`/etc/bkup/bkreg.tab`	describes the backup policy established by the administrator
`/etc/dgroup.tab`	lists logical groupings of devices as determined by the administrator
`/etc/device.tab`	describes specific devices and their attributes

SEE ALSO

`backup`(1M), `fdisk`(1M), `fdp`(1M), `incfile`(1M), `ffile`(1M), `fimage`(1M), `fmthard`(1M), `getvol`(1M), `labelit`(1M), `mkfs`(1M), `mount`(1M), `prtvtoc`(1M), `restore`(1M)

NAME

bkstatus - display the status of backup operations

SYNOPSIS

bkstatus [-h] [-f *field_separator*] [-j *jobids*] [-s *states* | -a] [-u *users*]

bkstatus -p *period*

DESCRIPTION

Without options, the bkstatus command displays the status of backup operations that are in progress: either active, pending, waiting or suspended. When used with the -a option, the backup command includes failed and completed backup operations in the display.

bkstatus -p defines the amount of status information that is saved for display.

bkstatus may only be executed by a user with superuser privilege.

Each backup operation goes through a number of states as described below. The keyletters listed in parentheses after each state are used with the -s option and also appear on the display.

pending(p)
: backup has been invoked and the operations in the backup register for the specified day are scheduled to occur.

active(a)
: The backup operation has been assigned a destination device and archiving is currently underway; or a suspended backup has been resumed.

waiting(w)
: The backup operation is waiting for operator interaction, such as inserting the correct volume.

suspended(s)
: The backup operation has been suspended by an invocation of backup -S.

failed(f)
: The backup operation failed or has been cancelled.

completed(c)
: The backup operation has completed successfully.

The -a and -s options are mutually exclusive.

Options

-a
: Include failed and completed backup operations in the display. All backup operations that have occurred within the rotation period are displayed.

-f *field_separator*
: Suppresses field wrap on the display and specifies an output field separator to be used. The value of *c* is the character that will appear as the field separator on the display output. For clarity of output, do not use a separator character that is likely to occur in a field. For example, do not use the colon as a field separator character if the display will contain dates that use a colon to separate hours from minutes. To use the

default field separator (tab), specify the null character ("") for *c*.

`-h` Suppress header on the display.

`-j` *jobids* Restrict the display to the specified list of `backup` job ids (either comma-separated or blank-separated and enclosed in quotes). [See `backup`(1M)].

`-p` *period* Define the amount of backup status information that is saved and made available for display as *period*. *period* is the number of weeks that information is saved in `/bkup/bkstatus.tab`. Status information that is older than the number of weeks specified in *period* is deleted from the status table. The minimum valid entry is 1. The maximum valid entry is 52. The default is 1 week.

`-s` *states* Restrict the report to backup operations with the specified *states*. *states* is a list of state key-letters (concatenated, comma-separated or blank-separated and surrounded by quotes). For example,

```
apf
a,p,f
"a p f"
```

all specify that the report should only include backup operations that are `active`, `pending` or `failed`.

`-u` *users* Restrict the display to backup operations started by the specified list of *users* (either comma-separated or blank-separated and enclosed in quotes). *users* must be in the `passwd` file.

DIAGNOSTICS

The exit codes for the `bkstatus` command are the following:

`0` = successful completion of the task
`1` = one or more parameters to `bkstatus` are invalid.
`2` = an error has occurred which caused `bkstatus` to fail to complete *all* portions of its task.

EXAMPLES

Example 1:

```
bkstatus -p 4
```

specifies that backup status information is to be saved for four weeks. Any status information older than four weeks is deleted from the system.

Example 2:

```
bkstatus -a -j back-459,back-395
```

produces a display that shows status for the two backup jobs specified, even if they have `completed` or `failed`.

Example 3:

```
bkstatus -s a,c -u "oper3 oper4"
```

produces a display that shows only those backup jobs issued by users `oper3` and `oper4` that have a status of either `active` or `completed`.

FILES

`/etc/bkup/bkstatus.tab`	lists the current status of backups that have occurred or are still in progress
`/etc/bkup/bkreg.tab`	describes the backup policy decided on by the System Administrator

SEE ALSO

`backup`(1M), `bkhist`(1M), `bkreg`(1M)

NAME

boot - bootstrap procedures

DESCRIPTION

Bootstrapping is the process of loading and executing a standalone program. The term bootstrapping is used to describe the process of loading and executing the bootable operating system, but any standalone program can be booted instead. The diagnostic monitor for a machine is a good example of a standalone program other than the operating system that can be booted.

The bootstrap procedure on most machines consists of the following basic phases.

First, the machine is either turned on, or brought down to firmware mode in any of a number of ways (hardware reset button, a shutdown or init command, and so on). On powerup, the boot process is generally begun automatically: a small firmware program is loaded and executed, and the process moves into the second phase.

From firmware mode, however, the boot process is not automatic and the user can request the running of a firmware command, a standalone program (such as the bootable operating system), or the reconfiguration of the operating system.

Assume that an operating system boot is requested from firmware. The firmware boot code loads and executes a disk (or other storage media) based boot program.

Next, the boot program loads and executes the bootable operating system. It is at this point that the UNIX system is started, necessary file systems are mounted [see vfstab(4)], and init is run to bring the system to the initdefault state specified in /etc/inittab [see inittab(4)].

For the Motorola reference platform, the boot program is called boot. These programs are taken from the boot slice on disk, and loaded and executed at boot time. A copy of this program exists in the directory /usr/lib, for the purpose of copying it to another hard disk using the dinit command.

The default bootable operating system file is /stand/unix. The /stand slice is defined in the disk's VTOC table.

BOOT COMMANDS AND OPTIONS

For more information about what commands and options the boot supports, issue a boot command from the BUG but specify a file name of ;help. This help message will produce output which looks something like that shown below.

```
188-Bug>bo 6 0;help
Booting from: VME328, Controller 6, Drive 0
Loading: ;help

Volume: $00000000

IPL loaded at: $00700000


System VR4.0 M88K Boot Loader
Boot commands: "bo x y ;command"

make-kernel     force a new unix to be built
```

```
help            obtain these helpful messages
ls              list the files in the V_STAND slice
l               an alias for ls
?               an alias for help

Boot options: "bo x y file;option[;option] ..."

root-slice      specify root FS slice; keyword=hex
boot-slice      specify V_STAND slice; keyword=hex
noprobe         do not run the device probes
one-cpu         use only the boot mpu
kdb             stop in kernel debugger on boot (if possible)
debug           specify the probe output; keyword=hex
halt            return to the BUG after loading kernel
h               an alias for halt

188-Bug>
```

The `make-kernel` command is used to force the system through the auto-configuration process. It is useful when you change a configurable parameter, forget to touch `/stand/system`, and shut the system down.

The `help` command lists the help message shown above.

The `ls` and `l` commands are used to view the contents of the V_STAND slice to determine what kernels are available.

The `root-slice` option specifies the slice on the boot device which is to be used as the root filesystem (this command is used only in special configurations). Slices are designated using a single hexadecimal digit [0-9a-fA-F].

The `boot-slice` option is used to select from the available V_STAND slices on a disk to locate the probes and the kernel (this command is used only in special configurations). Slices are designated using a single hexadecimal digit [0-9a-fA-F].

The `noprobe` option is used to avoid the time spent probing for new devices and subsequently checking whether the system needs to be reconfigured. No reconfiguration will occur if noprobe is specified even if one is needed.

On multiprocessor systems, one of the CPUs is used to initialize the majority of the system and then the remainder of the CPUs are started. Using the `one-cpu` option inhibits the starting of the remainder of the CPUs in the system. This option is useful for eliminating multiprocessor affects on the system (e.g., during driver debugging).

The `kdb` option causes the system to halt in the kernel debugger after the system has been initialized but before any additional CPUs are started and before the devices are initialized. If the kernel debugger is not installed, this option has no effect. See the `kdb`(1M) manpage for further details about the kernel debugger.

The `debug` option controls how much output is sent to the console terminal. The `hex` value must be between 0 and 7 inclusive and consists of three bits in which 0 indicates `off` and 1 indicates `on`. Bit 0 controls diagnostics for successful operations. Bit 1 controls diagnostics for failed operations. Bit 2 controls diagnostics for operations which are skipped (e.g., the ignore option in the EDT data file causes the entry to be skipped).

The `halt` and `h` options return to the ROM debugger after the image is loaded (this option requires intimate knowledge of how the image operates and should be used carefully). See the note below when using this option.

DIAGNOSTICS

The following table describes the diagnostics which may be seen when the system is booted.

`equal sign is missing`
: The option being executed requires an equal sign to separate the keyword from the value and no equal sign was found.

`invalid hex value`
: The option being executed requires a valid hexadecimal value. These valid

`debug value must be <= 7`
: The debug value must be between 0 and 7 inclusive.

`unknown option <option>`
: An unknown option was specified.

`file name too long`
: Due to limitations of the BFS file system, all files have names containing less than 14 characters. The file specified contains more than 14 characters.

`Unable to open <name>`
: The file name specified doesn't exist or can't be opened.

`file <filename>: bad magic`
: The file specified is not a COFF or ELF executable and thus can not be loaded.

`no compiled in configuration information found`
: Standard kernels have default configuration information compiled into the ELF file itself to allow the system to be booted without any probes. The file specified contained no such information.

`cannot open file system`
: The file /stand/system either doesn't exist or can't be opened.

`system file more recent than unix`
: The file /stand/system has been modified more recently than the kernel being booted and thus a reconfiguration of the kernel is required.

`cannot open file edt_data`
: The file /stand/edt_data either doesn't exist or can't be opened.

`edt_data file more recent than unix`
: The file /stand/edt_data has been modified more recently than the kernel being booted and thus a reconfiguration of the kernel is required.

`help may used only with list`
: The help command was attempted with a command which is not one of the list commands.

`slice <slice> isn't tagged V_STAND`
: An attempt was made to boot from a slice which isn't tagged as a V_STAND file system.

`No /stand slice on device`
There are no slices on this device which contain any bootable files.

`noprobe and debug may not be used together`
The commands `noprobe` and `debug` may not be used together.

`must not specify a file name`
A command or option was specified which does not take a file name but a file name was given.

`must specify a file name`
A command which requires a file was executed without any file specified.

`Name = <kernel_name>, start address = <start_address>`
This diagnostic message shows which object is booted and what the address in the object where execution will begin.

`EDT table overflow`
The number of devices specified in the EDT data file or found by the probes exceeds the boot program's internal table size.

`Using In-core EDT built by probe programs`
This diagnostic message is issued when the information gathered by the probes is used (as opposed to the EDT information in the kernel itself).

`Program load point <addr> is too near the boot loader`
The program, when loaded, would cause the boot loader to be overwritten.

`Incore and probed EDT entries differ for <name>, board <num>`
A difference between the probe and the previously configuration information was found. The kernel should be reconfigured.

`Adding required device <device> to EDT`
This diagnostic is printed when a required device is added to the configuration information regardless of whether the device is actually present in the system.

`No probe programs!: using builtin devices`
No probe programs were found so the previously configuration information will be used by the kernel.

Boot: unable to run this kernel on this CPU
The kernel is not configured to run on the CPU it was booted on.

Boot: kernel configuration information missing
One or more the compiled in configuration sections of the kernel file were missing from the kernel.

Boot: inappropriate kernel CPU support
The kernel if either configured for a different CPU than the one it was booted on or it has support for more CPUs than are necessary. The kernel should be reconfigured.

NOTES

The boot program is not smart enough to differentiate between a bootable kernel or stand-alone program and a UNIX executable. Booting a UNIX executable will result in unpredictable results.

When booting a UNIX kernel using the ;halt option, register 9 must be manually set to the value of 0xf00D before jumping to the start address of the kernel.

SEE ALSO

dinit(1M), init(1M), kdb(1M), fmthard(1M), prtvtoc(1M), shutdown(1M), inittab(4), vfstab(4).

NAME

`bootparamd` - boot parameter server

SYNOPSIS

`bootparamd` [`-d`]

DESCRIPTION

`bootparamd` is a server process that provides information to diskless clients necessary for booting. It obtains its information from the `/etc/bootparams` file.

`bootparamd` can be invoked either by `inetd`(1M) or by the user.

The `-d` option displays the debugging information.

FILES

`/etc/bootparams`

SEE ALSO

`inetd`(1M)

NAME

bootpd, in.bootpd - Internet Boot Protocol server

SYNOPSIS

bootpd [-s -t *timeout* -d] [*configfile* [*dumpfile*]]

DESCRIPTION

bootpd implements an Internet Boot Protocol server as defined in RFC 951 and RFC 1048. It is normally run by /etc/inetd by including the following line in the file /etc/inetd.conf:

```
bootps      dgram udp   wait  root  /etc/bootpd      bootpd
```

This causes bootpd to be started only when a boot request arrives. If bootpd does not receive another boot request within fifteen minutes of the last one it received, it will exit to conserve system resources. The -t switch may be used to specify a different timeout value in minutes (for example -t20). A timeout value of zero means forever.

It is also possible to run bootpd in a standalone configuration using the -s switch (for example, at boot time from /etc/rc.local). This is probably the desired mode of operation for large network installations with many hosts. In this case, the -t switch has no effect since bootpd will never exit.

Each instance of the -d switch increases the level of debugging output.

Upon startup, bootpd first reads its configuration file, /etc/bootptab, and then begins listening for BOOTREQUEST packets. The configuration file has a format similar to that of termcap(3X) in which two-character case-sensitive tag symbols are used to represent host parameters. These parameter declarations are separated by colons (:). The general format is:

hostname:*tg*=*value*:*tg*=*value*:*tg*=*value* . . .

where *hostname* is the actual name of a bootp client and tg is a two-character tag symbol. Most tags must be followed by an equals-sign and a value as above. Some may also appear in a boolean form with no value (that is, : tg :). The currently recognized tags are:

bf	Bootfile
bs	Bootfile size in 512-octet blocks
cs	Cookie server address list
ds	Domain name server address list
gw	Gateway address list
ha	Host hardware address
hd	Bootfile home directory
hn	Send hostname
ht	Host hardware type (see Assigned Numbers RFC)
im	Impress server address list
ip	Host IP address
lg	Log server address list
lp	LPR server address list
ns	IEN-116 name server address list
rl	Resource location protocol server address list
sm	Host subnet mask
tc	Table continuation (points to similar "template" host entry)

to	Time offset in seconds from UTC
ts	Time server address list
vm	Vendor magic cookie selector

There is also a generic tag, `Tn` , where *n* is an RFC 1048 vendor field tag number. Thus it is possible to immediately take advantage of future extensions to RFC 1048 without being forced to modify `bootpd` first. Generic data may be represented as either a stream of hexadecimal numbers or as a quoted string of ASCII characters. The length of the generic data is automatically determined and inserted into the proper field(s) of the RFC 1048-style bootp reply.

The following tags take a whitespace-separated list of IP addresses: `cs`, `ds`, `gw`, `im`, `lg`, `lp`, `ns`, `rl`, and `ts`. The `ip` and `sm` tags each take a single IP address. All IP addresses are specified in standard Internet "dot" notation and may use decimal, octal, or hexadecimal numbers (octal numbers begin with 0, hexadecimal numbers begin with '0x' or '0X').

The `ht` tag specifies the hardware type code as either an unsigned decimal, octal, or hexadecimal integer or one of the following symbolic names: `ethernet` or `ether` for 10Mb Ethernet, `ethernet3` or `ether3` for 3Mb experimental Ethernet, `ieee802`, `tr`, or `token-ring` for IEEE 802 networks, `pronet` for Proteon ProNET Token Ring, or `chaos`, `arcnet`, or `ax.25` for Chaos, ARCNET, and AX.25 Amateur Radio networks, respectively. The `ha` tag takes a hardware address which *must* be specified in hexadecimal; optional periods and/or a leading '0x' may be included for readability. The `ha` tag must be preceded by the `ht` tag (either explicitly or implicitly; see `tc` below).

The hostname, home directory, and bootfile are ASCII strings which may be optionally surrounded by double quotes ("). The client's request and the values of the `hd` and `bf` symbols determine how the server fills in the bootfile field of the bootp reply packet.

If the client specifies an absolute pathname and that file exists on the server machine, that pathname is returned in the reply packet. If the file cannot be found, the request is discarded; no reply is sent. If the client specifies a relative pathname, a full pathname is formed by prepending the value of the `hd` tag and testing for existence of the file. If the `hd` tag is not supplied in the configuration file or if the resulting boot file cannot be found, then the request is discarded.

Clients which specify null boot files will always elicit a reply from the server. The exact reply will again depend upon the `hd` and `bf` tags. If the `bf` tag gives an absolute pathname and the file exists, that pathname is returned in the reply packet. Otherwise, if the `hd` and `bf` tags together specify an accessible file, that filename is returned in the reply. If a complete filename cannot be determined or the file does not exist, the reply will contain a zeroed-out bootfile field.

In all these cases, existence of the file means that, in addition to actually being present, the file must have its public read access bit set, since this is required by `tftpd`(1M) to permit the file transfer. Also, all filenames are first tried as *filename.hostname* and then simply as *filename*, thus providing for individual per-host bootfiles.

The time offset to may be either a signed decimal integer specifying the client's time zone offset in seconds from UTC, or the keyword auto which uses the server's time zone offset. Specifying the to symbol as a boolean has the same effect as specifying auto as its value.

The bootfile size bs may be either a decimal, octal, or hexadecimal integer specifying the size of the bootfile in 512-octet blocks, or the keyword auto which causes the server to automatically calculate the bootfile size at each request. As with the time offset, specifying the bs symbol as a boolean has the same effect as specifying auto as its value.

The vendor magic cookie selector (the vm tag) may take one of the following keywords: auto (indicating that vendor information is determined by the client's request), rfc1048 (which always forces an RFC 1048-style reply), or cmu (which always forces a CMU-style reply).

The hn tag is strictly a boolean tag; it does not take the usual equals-sign and value. It's presence indicates that the hostname should be sent to RFC 1048 clients. bootpd attempts to send the entire hostname as it is specified in the configuration file; if this will not fit into the reply packet, the name is shortened to just the host field (up to the first period, if present) and then tried. In no case is an arbitrarily-truncated hostname sent (if nothing reasonable will fit, nothing is sent).

Often, many host entries share common values for certain tags (such as name servers, etc.). Rather than repeatedly specifying these tags, a full specification can be listed for one host entry and shared by others via the tc (table continuation) mechanism. Often, the template entry is a dummy host which doesn't actually exist and never sends bootp requests. This feature is similar to the tc feature of termcap(3X) for similar terminals. Note that bootpd allows the tc tag symbol to appear anywhere in the host entry, unlike termcap which requires it to be the last tag. Information explicitly specified for a host always overrides information implied by a tc tag symbol, regardless of its location within the entry. The value of the tc tag may be the hostname or IP address of any host entry previously listed in the configuration file.

Sometimes it is necessary to delete a specific tag after it has been inferred via tc. This can be done using the construction tag@ which removes the effect of tag as in termcap(3X). For example, to completely undo an IEN-116 name server specification, use ":ns@:" at an appropriate place in the configuration entry. After removal with @, a tag is eligible to be set again through the tc mechanism.

Blank lines and lines beginning with "#" are ignored in the configuration file. Host entries are separated from one another by newlines; a single host entry may be extended over multiple lines if the lines end with a backslash (\). It is also acceptable for lines to be longer than 80 characters. Tags may appear in any order, with the following exceptions: the hostname must be the very first field in an entry, and the hardware type must precede the hardware address.

An example `/etc/bootptab` file follows:

```
# Sample bootptab file
default1:\
      :hd=/usr/boot:bf=null:\
      :ds=128.2.35.50 128.2.13.21:\
      :ns=0x80020b4d 0x80020ffd:\
      :ts=0x80020b4d 0x80020ffd:\
      :sm=255.255.0.0:gw=0x8002fe24:\
      :hn:vm=auto:to=-18000:\
      :T37=0x12345927AD3BCF:T99="Special ASCII string":

carnegie:ht=6:ha=7FF8100000AF:ip=128.2.11.1:tc=default1:
baldwin:ht=1:ha=0800200159C3:ip=128.2.11.10:tc=default1:
wylie:ht=1:ha=00DD00CADF00:ip=128.2.11.100:tc=default1:
arnold:ht=1:ha=0800200102AD:ip=128.2.11.102:tc=default1:
bairdford:ht=1:ha=08002B02A2F9:ip=128.2.11.103:tc=default1:
bakerstown:ht=1:ha=08002B0287C8:ip=128.2.11.104:tc=default1:

# Special domain name server for next host
butlerjct:ht=1:ha=08002001560D:ip=128.2.11.108:ds=128.2.13.42:tc=default1:

gastonville:ht=6:ha=7FFF81000A47:ip=128.2.11.115:tc=default1:
hahntown:ht=6:ha=7FFF81000434:ip=128.2.11.117:tc=default1:
hickman:ht=6:ha=7FFF810001BA:ip=128.2.11.118:tc=default1:
lowber:ht=1:ha=00DD00CAF000:ip=128.2.11.121:tc=default1:
mtoliver:ht=1:ha=00DD00FE1600:ip=128.2.11.122:tc=default1:
```

`bootpd` looks in `/etc/services` to find the port numbers it should use. Two entries are extracted: `bootps` -- the bootp server listening port, and `bootpc` -- the destination port used to reply to clients. If the port numbers cannot be determined this way, they are assumed to be 67 for the server and 68 for the client.

`bootpd` rereads its configuration file when it receives a hangup signal, SIGHUP, or when it receives a bootp request packet and detects that the file has been updated. Hosts may be added, deleted or modified when the configuration file is reread. If `bootpd` is compiled with the -DDEBUG option, receipt of a SIGUSR1 signal causes it to dump its memory-resident database to the file `/etc/inet/bootpd.dump` or the command-line-specified dumpfile.

USER CONSIDERATIONS

Individual host entries must not exceed 1024 characters.

FILES

`/etc/bootptab`
`/etc/inet/bootpd.dump`
`/etc/services`

SEE ALSO

`inetd`(1M)
RFC 951, RFC 1048, RFC 1084

NAME

brc, bcheckrc - system initialization procedures

SYNOPSIS

/sbin/brc

/sbin/bcheckrc

DESCRIPTION

These shell procedures are executed via entries in /etc/inittab by init whenever the system is booted.

First, the bcheckrc procedure checks the status of the root file system. If the root file system is found to be bad, bcheckrc repairs it.

Then, bckeckrc mounts the /stand, /proc, and /var (if it exists) file systems (/var may exist as a directory in the root file system, or as a separate file system).

The brc script performs administrative tasks related to file sharing.

After these two procedures have executed, init checks for the initdefault value in /etc/inittab. This tells init in which run level to place the system. If, for example, initdefault is set to 2, the system will be placed in the multi-user state via the rc2 procedure.

Note that bcheckrc should always be executed before brc. Also, these shell procedures may be used for several run-level states.

SEE ALSO

fsck(1M), init(1M), rc2(1M), shutdown(1M), inittab(4), mnttab(4)

NAME

buildsys - operating system configuration script

SYNOPSIS

/sbin/buildsys [-s]

DESCRIPTION

The buildsys shell script performs the activities necessary to build a new bootable operating system from single user mode. buildsys is executed by the shell script rc6 or during a powerup if the configuration of a new bootable operating system is necessary. Normally, buildsys will not be called if the file /stand/noautoconfig exists. The bootable operating system resides in /stand and is generally referred to as unix.

Building a new operating system is usually required by hardware and system software changes made to your system. These changes must be incorporated into the bootable operating system so that it has complete and correct knowledge of the system configuration.

buildsys performs the following activities:

checks and mounts the file systems listed in /etc/boot_tab

optionally saves the current bootable unix (see NOTES below)

runs cunix to create a new unix

unmounts all file systems previously mounted

optionally reboots the system; a reboot is requested if buildsys was run during a powerup (that is, the -s option was specified); if it was run by rc6 (no -s option), control is returned to rc6

If an error occurs during the configuration of a new unix, buildsys exits to a shell; this gives the user a chance to fix any problems that might have caused the configuration process to fail, or to copy a version of unix to /stand/unix that is known to work in order to reboot the system. Exiting this shell (using ctrl-d or exit), puts the machine in firmware mode. The machine can then be rebooted from firmware.

NOTES

If the kernel debugger module, KDB, is listed in the system file, buildsys will automatically run dbsym(1M) and dbcmd(1M). These programs will load the kernel symbols and macros into the new kernel so they will be accessible to the debugger.

buildsys overwrites /stand/unix. To prevent loss of the bootable unix corresponding to a crash dump when an autoconfigure runs during a crash recovery, a procedure to save the current bootable unix may be enabled by editing /sbin/buildsys and following the instructions contained therein.

SEE ALSO

crashconf(1M), cunix(1M), dbcmd(1M), dbsym(1M), init(1M), kdb(1M), rc6(1M), shutdown(1M), vfstab(4).

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. The *month* is a number between 1 and 12. The *year* can be between 1 and 9999. The calendar produced is that for England and the United States.

NOTES

An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type: `cal 9 1752`

The command `cal 83` refers to the year 83, not 1983.

The year is always considered to start in January even though this is historically naive.

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. Most reasonable month-day dates such as Aug. 24, august 24, 8/24, and so on, are recognized, but not 24 August or 24/8. On weekends "tomorrow" extends through Monday. calendar can be invoked regularly by using the crontab(1) or at(1) commands.

When an argument is present, calendar does its job for every user who has a file calendar in his or her login directory and sends them any positive results by mail(1). Normally this is done daily by facilities in the UNIX operating system (see cron(1M)).

If the environment variable DATEMSK is set, calendar will use its value as the full path name of a template file containing format strings. The strings consist of field descriptors and text characters and are used to provide a richer set of allowable date formats in different languages by appropriate settings of the environment variable LANG or LC_TIME (see environ(5)). (See date(1) for the allowable list of field descriptors.)

EXAMPLES

The following example shows the possible contents of a template:

```
%B %eth of the year %Y
```

%B represents the full month name, %e the day of month and %Y the year (4 digits).

If DATEMSK is set to this template, the following calendar file would be valid:

```
March 7th of the year 1989 < Reminder>
```

FILES

/usr/lib/calprog program used to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*

SEE ALSO

at(1), cron(1M), crontab(1), date(1), mail(1), environ(5).

NOTES

Appropriate lines beginning with white space will not be printed.
Your calendar must be public information for you to get reminder service.
calendar's extended idea of "tomorrow" does not account for holidays.

NAME

captoinfo — convert a *termcap* description into a *terminfo* description

SYNOPSIS

captoinfo [-v...] [-V] [-1] [-w *width*] *file*...

DESCRIPTION

captoinfo looks in *file* for termcap descriptions. For each one found, an equivalent terminfo description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the termcap tc = field) will be reduced to the minimum superset before being output.

If no *file* is given, then the environment variable TERMCAP is used for the filename or entry. If TERMCAP is a full pathname to a file, only the terminal whose name is specified in the environment variable TERM is extracted from that file. If the environment variable TERMCAP is not set, then the file /usr/share/lib/termcap is read.

-v print out tracing information on standard error as the program runs. Specifying additional -v options will cause more detailed information to be printed.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

-w change the output to *width* characters.

FILES

/usr/share/lib/terminfo/?/* Compiled terminal description database.

NOTES

captoinfo should be used to convert termcap entries to terminfo entries because the termcap database (from earlier versions of UNIX System V) may not be supplied in future releases.

SEE ALSO

curses(3X), infocmp(1M), terminfo(4)

NAME

cat - concatenate 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 file on your terminal, and:

```
cat file1 file2 >file3
```

concatenates file1 and file2, and writes the results in file3.

If no input file is given, or if the argument - is encountered, cat reads from the standard input file.

The following options apply to cat:

-u The output is not buffered. (The default is buffered output.)

-s cat is silent about non-existent files.

-v Causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. ASCII control characters (octal 000 - 037) are printed as ^*n*, where *n* is the corresponding ASCII character in the range octal 100 - 137 (@, A, B, C, . . ., X, Y, Z, [, \,], ^, and _); the DEL character (octal 0177) is printed ^?. Other non-printable characters are printed as M-*x*, where *x* is the ASCII character specified by the low-order seven bits.

When used with the -v option, the following options may be used:

-t Causes tabs to be printed as ^I's and formfeeds to be printed as ^L's.

-e 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.

SEE ALSO

cp(1), pg(1), pr(1)

NOTES

Redirecting the output of cat onto one of the files being read will cause the loss of the data originally in the file being read. For example,

```
cat file1 file2 >file1
```

causes the original data in file1 to be lost.

INTERNATIONAL FUNCTIONS

cat can read and write files containing characters from supplementary code sets.

When invoked with the -v option, cat considers all characters from supplementary code sets to be printable.

NAME

catman - create the cat files for the manual

SYNOPSIS

/usr/ucb/catman [-nptw] [-M *directory*] [-T *tmac.an*] [*sections*]

DESCRIPTION

The catman commands creates the preformatted versions of the on-line manual from the nroff(1) input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, catman recreates the whatis database.

If there is one parameter not starting with a '-', it is taken to be a list of manual sections to look in. For example

```
catman 123
```

only updates manual sections 1, 2, and 3.

The following options are available:

-n Do not (re)create the whatis database.

-p Print what would be done instead of doing it.

-t Create troffed entries in the appropriate fmt subdirectories instead of nroffing into the cat subdirectories.

-w Only create the whatis database. No manual reformatting is done.

-M Update manual pages located in the specified directory (/usr/share/man by default).

-T Use tmac.an in place of the standard manual page macros.

ENVIRONMENT

TROFF The name of the formatter to use when the -t flag is given. If not set, 'troff' is used.

FILES

/usr/share/man	default manual directory location
/usr/share/man/man?/*.*	raw (nroff input) manual sections
/usr/share/man/cat?/*.*	preformatted nroffed manual pages
/usr/share/man/fmt?/*.*	preformatted troffed manual pages
/usr/share/man/whatis	whatis database location
/usr/ucblib/makewhatis	command script to make whatis database

SEE ALSO

man(1), nroff(1), troff(1), whatis(1)

DIAGNOSTICS

man?/xxx.? (.so'ed from man?/yyy.?): No such file or directory
The file outside the parentheses is missing, and is referred to by the file inside them.

target of .so in man?/xxx.? must be relative to /usr/man
catman only allows references to filenames that are relative to the directory /usr/share/man.

`opendir:man?: No such file or directory`
A harmless warning message indicating that one of the directories `catman` normally looks for is missing.

`*.*: No such file or directory`
A harmless warning message indicating `catman` came across an empty directory.

NAME

cb - C program beautifier

SYNOPSIS

cb [-s] [-j] [-l *leng*] [-V] [*file* . . .]

DESCRIPTION

The cb comand reads syntactically correct C programs either from its arguments or from the standard input, and writes them on the standard output with spacing and indentation that display the structure of the C code. By default, cb preserves all user new-lines.

cb accepts the following options.

-s	Write the code in the style of Kernighan and Ritchie found in *The C Programming Language*.
-j	Put split lines back together.
-l *leng*	Split lines that are longer than *leng*.
-V	Print on standard error output the version of cb invoked.

NOTES

cb treats asm as a keyword.

The format of structure initializations is unchanged by cb.

Punctuation that is hidden in preprocessing directives causes indentation errors.

SEE ALSO

cc(1)

Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Second Edition, Prentice-Hall, 1988

NAME

cc - configurable C compiler

SYNOPSIS

cc [*options*] *file* . . .

DESCRIPTION

/usr/bin/cc is the interface to a choice of C compilation systems. There are two supported compilation systems available: GNU C, and the ABI (Application Binary Interface) compilation system. Regardless of which underlying compilation system is used, the command line syntax will always be the syntax described in this manual page.

cc will use GNU C by default, or you may actively choose which compilation system to execute by setting the environment variable CCCOMPILER to either gnu or abi. Compilation systems other than GNU C or ABI can also be used with the configurable cc command.

Because cc usually creates files in the current directory during the compilation process, it is necessary to run cc in a directory in which the output files can be created.

Components

The compilation tools conceptually consist of a preprocessor, compiler, optimizer, basic block analyzer, assembler, and link editor. cc processes the supplied options and then executes the various tools with the proper arguments. cc accepts several types of files as arguments.

Files whose names end with .c are taken to be C source files and may be preprocessed, compiled, optimized, instrumented for profiling, assembled, and link edited. The compilation process may be stopped after the completion of any pass if the appropriate options are supplied. If the compilation process runs through the assembler, then an object file is produced whose name is that of the source with .o substituted for .c. However, the .o file is normally deleted if a single C file is compiled and then immediately link edited. In the same way, files whose names end in .s are taken to be assembly source files; they may be assembled and link edited. Files whose names end in .i are taken to be preprocessed C source files, and they may be compiled, optimized, instrumented for profiling, assembled, and link edited. Files whose names do not end in .c, .s, or .i are handed to the link editor, which produces a dynamically linked executable whose name by default is a.out.

When cc is put in a file *prefix*cc, the prefix will be recognized and used to prefix the names of each tool executed. For example, OLDcc will execute OLDacomp, OLDnewoptim, OLDbasicblk, OLDas, and OLDld. Therefore, be careful when moving cc around, and be sure you already have in place the prefixed versions of the compilation components. The prefix applies to the compiler, optimizer, basic block analyzer, assembler, and link editor.

Options

The following options are interpreted by cc:

-A *name*[(tokens)]
: Associate *name* as a predicate with the specified *tokens* as if by a #assert preprocessing directive.

Preassertions: `system(unix)`
`cpu (m68k or m88k)`
`machine(m68k or m88k)`

`-A -` Cause all predefined macros (other than those that begin with __) and predefined assertions to be forgotten.

`-B` *c* *c* can be either `dynamic` or `static`. `-B dynamic` causes the link editor to look for files named `lib`*x*`.so` and then for files named `lib`*x*`.a` when given the `-l`*x* option. `-B static` causes the link editor to look only for files named `lib`*x*`.a`. This option may be specified multiple times on the command line as a toggle. This option and its argument are passed to `ld`.

`-C` Cause the preprocessing phase to pass along all comments other than those on preprocessing directive lines.

`-c` Suppress the link editing phase of the compilation and do not remove any produced object files.

`-D` *name*[=*tokens*]
Associate *name* with the specified *tokens* as if by a `#define` preprocessing directive. If no =*tokens* is specified, the token `1` is supplied.

Predefinitions: `m68k or m88k`
`unix`

`-d` *c* `c` can be either `y` or `n`. `-dy` specifies dynamic linking, which is the default, in the link editor. `-dn` specifies static linking in the link editor. This option and its argument are passed to `ld`.

`-E` Only preprocess the named C files and send the result to the standard output. The output will contain preprocessing directives for use by the next pass of the compilation system.

`-f` This option is obsolete and will be ignored.

`-G` Direct the link editor to produce a shared object rather than a dynamically linked executable. This option is passed to `ld`. It cannot be used with the `-dn` option.

`-g` Cause the compiler to generate additional information needed for the use of `tbx`. Use of `tbx` on a program compiled with both the `-g` and `-O` options is not recommended unless you understand the behavior of optimization.

`-H` Print, one per line, the path name of each file included during the current compilation on the standard error output.

`-I` *dir* Alter the search for included files whose names do not begin with `/` to look in *dir* prior to the usual directories. The directories for multiple `-I` options are searched in the order specified.

`-J sfm`
Not supported on Motorola 68000 and 88000 systems. A warning to this effect will be printed.

`-K` [*mode,goal*, `PIC`, `minabi`]

`-K` *mode* Not supported on Motorola 68000 and 88000 systems. A warning to this effect will be printed.

`-K` *goal* Not supported on Motorola 68000 and 88000 systems. A warning to this effect will be printed.

`-K PIC` Cause position-independent code (PIC) to be generated.

`-K minabi` Direct the compilation system to use a version of the C library that minimizes dynamic linking, without changing the application's ABI conformance (or non-conformance, as the case may be). Applications that use the Network Services Library or the X library may not use `-K` minabi.

The `-K` option can accept multiple arguments. For example, `-K PIC, minabi` can be used instead of `-K PIC -K minabi`.

`-L` *dir* Add dir to the list of directories searched for libraries by `ld`. This option and its argument are passed to `ld`.

`-l` *name*
Search the library `lib`*name*`.so` or `lib`*name*`.a`. Its placement on the command line is significant as a library is searched at a point in time relative to the placement of other libraries and object files on the command line. This option and its argument are passed to `ld`.

`-O` Arrange for compilation phase optimization. This option has no effect on `.s` files.

`-o` *pathname*
Produce an output object file *pathname*, instead of the default `a.out`. This option and its argument are passed to `ld`.

`-P` Only preprocess the named C files and leave the result in corresponding files suffixed `.i`. The output will not contain any preprocessing directives, unlike `-E`.

`-p` Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing takes place, profiled versions of `libc.a` and `libm.a` (with the `-lm` option) are linked if the `-dn` option is used. A `mon.out` file will then be produced at normal termination of execution of the object program. An execution profile can then be generated by use of `prof`.

`-Q` *c* *c* can be either `y` or `n`. If *c* is `y`, identification information about each invoked compilation tool will be added to the output files (the default behavior). This can be useful for software administration. Giving `n` for *c* suppresses this information.

`-q` *c* *c* can be either `l` or `p`. `-ql` causes the invocation of the basic block analyzer and arranges for the production of code that counts the number of times each source line is executed. A listing of these counts can be generated by use of `lprof`. `-qp` is a synonym for `-p`.

`-S` Compile, optimize (if `-O` is present), and do not assemble or link edit the named C files. The assembler-language output is left in corresponding files suffixed `.s`.

-U *name*
Cause any definition of *name* to be forgotten, as if by a `#undef` preprocessing directive. If the same *name* is specified for both `-D` and `-U`, *name* is not defined, regardless of the order of the options.

`-V` Cause each invoked tool to print its version information on the standard error output.

`-v` Cause the compiler to perform more and stricter semantic checks, and to enable certain `lint-like` checks on the named C files.

`-W` *tool*, arg_1[, arg_2 . . .]
Hand off the argument(s) arg_i each as a separate argument to *tool*. Each argument must be separated from the preceding by only a comma. (A comma can be part of an argument by escaping it by an immediately preceding backslash (`\`) character; the backslash is removed from the resulting argument.) *tool* can be one of the following:

`p`	preprocessor
`0`	compiler
`2`	optimizer
`b`	basic block analyzer
`a`	assembler
`l`	link editor

For example, `-Wa,-o,`*objfile* passes `-o` and *objfile* to the assembler, in that order; also `-Wl,-I,`*name* causes the linking phase to override the default name of the dynamic linker, `/usr/lib/libc.so.1`.

The order in which the argument(s) are passed to a tool with respect to the other specified command line options may change.

`-X` *c* Specify the degree of conformance to the ANSI C standard. *c* can be one of the following:

`t` (transition)
The compiled language includes all new features compatible with older (pre-ANSI) C (the default behavior). The compiler warns about all language constructs that have differing behavior between the new and old versions and uses the pre-ANSI C interpretation. This includes, for example, warning about the use of trigraphs the new escape sequence `\a`, and the changes to the integral promotion rules.

`a` (ANSI)
The compiled language includes all new features of ANSI C and uses the new interpretation of constructs with differing behavior. The compiler continues to warn about the integral promotion rule changes, but does not warn about trigraph replacements or new escape sequences.

`c` (conformance)
The compiled language and associated header files are ANSI C conforming, but include all conforming extensions of `-Xa`. Warnings will be produced about some of these. Also, only ANSI defined identifiers are visible in the standard header files.

The predefined macro __STDC__ has the value 0 for -Xt and -Xa, and 1 for -Xc. All warning messages about differing behavior can be eliminated in -Xa through appropriate coding; for example, use of casts can eliminate the integral promotion change warnings.

An additional option, -Xn, is also recognized. This option is identical to -Xt except that the application is required to provide the definition of the run-time variable *_lib_version*. The provision of this option is to accommodate test suites. The usage of the -Xn option by applications is not encouraged, because it may not be portable to other Release 4 systems.

-Y *item*, *dir*

Specify a new directory *dir* for the location of *item*. *item* can consist of any of the characters representing tools listed under the -W option or the following characters representing directories containing special files:

F	obsolete. Use -YP instead.
I	directory searched last for include files: *INCDIR* (see -I)
S	directory containing the start-up object files: *LIBDIR*
L	obsolete. Use -YP instead.
U	obsolete. Use -YP instead.
P	Change the default directories used for finding libraries. *dir* is a colon-separated path list.

If the location of a tool is being specified, then the new path name for the tool will be *dir*/*tool*. If more than one -Y option is applied to any one item, then the last occurrence holds.

cc recognizes -a, -B, -e, -h -m, -o, -r, -s, -t, -u, and -z and passes these options and their arguments to ld. cc also passes any unrecognized options to ld without any diagnostic.

FILES

file.c	C source file
file.i	preprocessed C source file
file.o	object file
file.s	assembly language file
a.out	link-edited output
LIBDIR/*crti.o	startup initialization code
LIBDIR/*crt1.o	startup routine
LIBDIR/*crtn.o	last startup routine
TMPDIR/*	temporary files
/usr/bin/cc	configurable compiler driver
LIBDIR/.compilerc	default compiler configuration file
BINDIR/gcc	GNU C compiler driver
LIBDIR/gcc-cpp	GNU C preprocessor
LIBDIR/gcc-cc1	GNU C compiler
LIBDIR/gcc-include	GNU C include directory
LIBDIR/gcc-gnulib	GNU C library
LIBDIR/cpp	USL CI5 preprocessor
LIBDIR/basicblk	basic block analyzer

BINDIR/as	assembler
BINDIR/ld	link editor
LIBDIR/libc.so	shared standard C library
LIBDIR/libc.a	archive standard C library
INCDIR	usually /usr/include
LIBDIR	usually /usr/ccs/lib
BINDIR	usually /usr/ccs/bin
TMPDIR	usually /var/tmp but can be redefined by setting the environment variable TMPDIR (see tempnam in tmpnam(3S)).

Kernighan, B. W., and Ritchie, D. M., *The C Programming Language*, Second Edition, Prentice-Hall, 1988

American National Standard for Information Systems - Programming Language C, X3.159-1989

NOTES

Obsolescent but still recognized cc options include -f and -F. The -ql and -O options do not work together; -O will be ignored.

SEE ALSO

as(1), ld(1), lint(1), lprof(1), prof(1), tbx(1) monitor(3C), tmpnam(3S).

NAME

cc - C compiler

SYNOPSIS

/usr/ucb/cc [*options*]

DESCRIPTION

/usr/ucb/cc is the C compiler for the BSD Compatibility Package. The behavior of /usr/ucb/cc is identical to /usr/ccs/bin/cc (see cc(1)) except that BSD header files are used and BSD libraries are linked *before* System V libraries.

/usr/ucb/cc accepts the same options as /usr/ccs/bin/cc, with the following exceptions:

-I *dir* Search *dir* for included files whose names do not begin with a '/', prior to the usual directories. The directories for multiple -I options are searched in the order specified. The preprocessor first searches for #include files in the directory containing *sourcefile*, and then in directories named with -I options (if any), then /usr/ucbinclude, and finally, in /usr/include.

-L *dir* Add *dir* to the list of directories searched for libraries by /usr/bin/cc. This option is passed to /usr/bin/ld. Directories specified with this option are searched before /usr/ucblib and /usr/lib.

-Y LU,*dir* Change the default directory used for finding libraries.

FILES

/usr/ucblib
/bin/ld
/usr/ucblib/libucb.a
/usr/lib/libucb.a

NOTES

The -Y LU,*dir* option may have unexpected results, and should not be used. This option is not in the UNIX System V base.

SEE ALSO

ld(1), as(1), ar(1), cc(1), ld(1), lorder(1), ranlib(1), strip(1), tsort(1), a.out(4)

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 /, ., or . ., *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 by and is internal to the shell.

SEE ALSO

pwd(1), sh(1), chdir(2).

NAME

cdc - change the delta comment of an SCCS delta

SYNOPSIS

cdc -r *SID* [-m[*mrlist*]] [-y[*comment*]] *file* . . .

DESCRIPTION

cdc changes the delta comment, for the SID (SCCS identification string) specified by the -r keyletter, of each named SCCS file.

The delta comment is the Modification Request (MR) and comment information normally specified via the -m and -y keyletters of the delta command.

If *file* is a directory, cdc 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 the NOTES section) and each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to cdc, which may 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 comment is to be changed.

-m*mrlist* If the SCCS file has the v flag set [see admin(1)] then a list of MR numbers to be added and/or deleted in the delta comment of the SID specified by the -r keyletter may be supplied. A null MR list has no effect.

mrlist entries are added to the list of MRs in the same manner as that of delta. In order to delete an MR, precede the MR number with the character ! (see the EXAMPLES section). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a comment line. A list of all deleted MRs is placed in the comment section of the delta comment 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).

mrlist entries 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) 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 comment remains unchanged.

-y[*comment*] Arbitrary text used to replace the *comment*(s) already existing for the delta specified by the -r keyletter. The 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 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.

If you made the delta and have the appropriate file permissions, you can change its delta comment. If you own the file and directory you can modify the delta comment.

EXAMPLES

```
cdc -r1.6 -m"bl88-12345 !bl87-54321 bl89-00001" -ytrouble s.file
```

adds bl88-12345 and bl89-00001 to the MR list, removes bl87-54321 from the MR list, and adds the comment trouble to delta 1.6 of s.file.

Entering:

```
cdc -r1.6 s.file
MRs? !bl87-54321 bl88-12345 bl89-00001
comments? trouble
```

produces the same result.

FILES

x-file [see delta(1)]
z-file [see delta(1)]

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4)

DIAGNOSTICS

Use help for explanations.

NOTES

If SCCS file names are supplied to the cdc command via the standard input (- on the command line), then the -m and -y keyletters must also be used.

NAME

ce_bds - Common Environment board status

SYNOPSIS

`ce_bds` [`-s`] [`-h`]

DESCRIPTION

`ce_bds` prints information for each board in the Common Environment, in the format:

cpu type addr board state

where:

cpu is the board cpu id,

type is the board type (e.g. 374 = Ethernet controller)

addr is the base address for the board

board is an index from the first board in the Common Environment (i.e. first board is 0)

state is either "communicating" or "failure"

The options are defined as follows:

`-s` short format output:

cpu type board

`-h` no headers (prints the information without the column headers)

FILES

`/etc/ce_bds`

NAME

ce_reset - Common Environment reset utility

SYNOPSIS

ce_reset *dst_cpu*

DESCRIPTION

ce_reset recreates the Common Environment Message Channel to the destination CPU. It directs the Common Environment driver to gather all the shared memory resources (BPEs, EPBs, and BUFs), reinitialize them into their original resource pools, and then recreate the Message Channel with the destination CPU given by *dst_cpu*.

NAME

cflow - generate C flowgraph

SYNOPSIS

cflow [-r] [-ix] [-i_] [-d*num*] *files*

DESCRIPTION

The cflow command analyzes a collection of C, yacc, lex, assembler, and object files and builds a graph charting the external function references. Files suffixed with .y, .l, and .c are processed by yacc, lex, and the C compiler as appropriate. The results of the preprocessed files, and files suffixed with .i, are then run through the first pass of lint. Files suffixed with .s are assembled. Assembled files, and files suffixed with .o, have information extracted from their symbol tables. The results are collected and turned into a graph of external references that is written on the standard output.

Each line of output begins with a reference number, followed by a suitable number of tabs indicating the level, then the name of the global symbol followed by a colon and its definition. Normally only function names that do not begin with an underscore are listed (see the -i options below). 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, suppose the following code is 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 output of `cflow` can be piped to the `pr` command, using the `-e` option, to compress the tab expansion to something less than every eight spaces.

In addition to the `-D`, `-I`, and `-U` options [which are interpreted just as they are by `cc`], 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 also sorted in lexicographical order by callee.

`-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).

`-d`*num* The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this number is very large. Attempts to set the cutoff depth to a nonpositive integer will be ignored.

SEE ALSO

`as`(1), `cc`(1), `lex`(1), `lint`(1), `nm`(1), `pr`(1), `yacc`(1).

DIAGNOSTICS

Complains about multiple definitions and only believes the first.

NOTES

Files produced by `lex` and `yacc` cause the reordering of line number declarations, which can confuse `cflow`. To get proper results, feed `cflow` the `yacc` or `lex` input.

NAME

checkfsys - check a file system

SYNOPSIS

checkfsys

DESCRIPTION

The checkfsys command allows you to check for and optionally repair a damaged file system. The command invokes a visual interface (the check task available through the sysadm command). The initial prompt allows you to select the device that contains the filesystem. Then you are asked to specify the type of checking. The following choices are available:

check only
: Check the file system. No repairs are attempted.

interactive fix
: Repair the file system interactively. You are informed about each instance of damage and asked if it should be repaired.

automatic fix
: Repair the file system automatically. The program applies a standard repair to each instance of damage.

The identical function is available under the sysadm menu:

```
sysadm check
```

NOTES

While automatic and interactive checks are generally successful, they can occasionally lose a file or a file's name. Files with content but without names are put in the *file-system*/lost+found directory.

If it is important not to lose data, check the file system first to see if it appears to be damaged. If it does, use one of the repair options of the task.

DIAGNOSTICS

The checkfsys command exits with one of the following values:

0 Normal exit.

2 Invalid command syntax. A usage message is displayed.

7 The visual interface for this command is not available because it cannot invoke fmli. (The FMLI package is not installed or is corrupted.)

SEE ALSO

fsck(1M), makefsys(1M), mountfsys(1M), sysadm(1M)

NAME

checknr - check nroff and troff input files; report possible errors

SYNOPSIS

/usr/ucb/checknr [-fs] [-a *.x1 .y1 .x2 .y2 ... .xn .yn*]
[-c *.x1 .x2 .x3 ... .xn*] [*filename ...*]

DESCRIPTION

The checknr command checks a list of nroff 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:

Font changes using \f*x* ... \fP.

Size changes using \s*x* ... \s0.

Macros that come in open ... close forms, for example, the .TS and .TE macros which must always come in pairs.

checknr knows about the ms and me macro packages.

checknr is intended to be used on documents that are prepared with checknr in mind. It expects a certain document writing style for \f and \s commands, in that each \f*x* must be terminated with \fP and each \s*x* must be terminated with \s0. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from checknr. Since it is probably better to use the \fP and \s0 forms anyway, you should think of this as a contribution to your document preparation style.

The following options are available:

-f Ignore \f font changes.

-s Ignore \s size changes.

-a *.x1 .y1 ...*
Add pairs of macros to the list. The pairs of macros are assumed to be those (such as .DS and .DE) that should be checked for balance. The -a option 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

-c *.x1 ...*
Define commands which checknr would otherwise complain about as undefined.

SEE ALSO

eqn(1), nroff(1), troff(1), me(7), ms(7)

NOTES

There is no way to define a one-character macro name using the -a option.

NAME

chgrp - change the group ownership of a file

SYNOPSIS

chgrp [-R] [-h] *group file . . .*

DESCRIPTION

chgrp changes the group ID of the *files* given as arguments to *group*. The group may be either a decimal group ID or a group name found in the group ID file, /etc/group.

You must be the owner of the file, or be the super-user to use this command.

The operating system has a configuration option {_POSIX_CHOWN_RESTRICTED}, to restrict ownership changes. When this option is in effect, the owner of the file may change the group of the file only to a group to which the owner belongs. Only the super-user can arbitrarily change owner IDs whether this option is in effect or not.

chgrp has one option:

-R Recursive. chgrp descends through the directory, and any subdirectories, setting the specified group ID as it proceeds. When symbolic links are encountered, they are traversed.

-h If the file is a symbolic link, change the group of the symbolic link. Without this option, the group of the file referenced by the symbolic link is changed.

FILES

/etc/group

SEE ALSO

chmod(1), chown(1), id(1M), chown(2), group(4), passwd(4).

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the ls -l command leads you to believe.

NAME

chkey - change user encryption key

SYNOPSIS

chkey [-f]

DESCRIPTION

The chkey command prompts for a password and uses it to encrypt a new user encryption key. The encrypted key is stored in the publickey(4) database.

The -f option is used to force an encryption key entry into the database for a user who doesn't already have an entry.

This command should be executed only on the master server for the publickey(4) database.

SEE ALSO

keylogin(1), keylogout(1), publickey(4), keyserv(1M), newkey(1)

NAME

chkyn - get yes/no response from user or check answer to question

SYNOPSIS

chkyn [-ynfeid!] [-D *defaultvalue*] [-k *pid*]
[-q *inputvalue*] [-Q *aliasname*]
[-h *helpstring*] [-H *helpmessage*] [-m *number*] *Prompt String*

chkyn [-c] *Prompt String answer . . .*

chkyn [-r] *Prompt String answer message . . .*

chkyn [-R] *Prompt String answer message . . .*

DESCRIPTION

chkyn is primarily used by the sysadm (system administration) package. This utility is useful in a variety of shell script applications.

chkyn prints out the *Prompt String*, followed by the possible answers, and asks the user for input. The handling of user input is controlled by the command line options.

chkyn exits with an exit value of 0 for yes or 1 for no.

The following command line options are supported by chkyn:

-y Default answer to prompt is yes.

-n Default answer to prompt is no.

-f Force an answer. If the answer supplied by the user is not valid, ask the question again, and continue asking until a valid answer is obtained.

-e Echo the answer. This allows a shell script to capture the response which the user typed in.

-c *Prompt String answer . . .*
Check a list. This is often used in conjunction with the -f option, which forces an answer. chkyn prompts the user with the prompt string (which does not automatically include a question mark), then expects a reply that matches one of the *answer* fields. The exit code of chkyn indicates which item from the list of answers was chosen; an exit code higher than the number of items in the list indicates the user's entry did not match any of the list items.

-r *Prompt answer message . . .*
Accept matching regular expressions. This is most useful with the -f option, which forces an answer. The user supplies a regular expression in *answer*; chkyn prints *message* if a nonmatching answer is given.

-R Reject matching regular expressions. This works the same as -r, except that if an *answer* from the list is matched, chkyn prints the corresponding *message* and considers this an invalid answer.

-i Match an integer expression. This requires the *bs* utility, which is not supported on SYSTEM V/88.

`-d` Match a decimal expression. This requires the *bs* utility, which is not supported on SYSTEM V/88.

`-D` *value* The default value for an empty input. Must be accompanied by a `-c`, `-r`, `-i` or `-d` option.

`-k` *pid* Quit sends SIGTERM to this process id. Used with the `-q` or `-Q` options, which define *quit*.

`-q` *input* Input that causes quit.

`-Q` *alias* Alias name for quit.

`-h` *helpstring*
String to request help. `-h` and `-H` must both be specified.

`-H` *helpmessage*
Help message to display. `-h` and `-H` must both be specified.

`-m` *number*
Maximum number of times to ask a question.

`-!` Permit shell escapes from prompt.

DIAGNOSTICS

Exit value from `chkyn` is basically `0` for true or `1` for false. When a list of acceptable answers is provided (as with the `-c` option), the exit code points to the matched item in the list. A higher exit code indicates no item was matched.

NOTES

Since `chkyn` is part of the `sysadm` package, the options and command syntax may change in future releases to support the requirements of `sysadm`. User scripts that depend on certain functions of `chkyn` would need to be rewritten at that time.

The meaning of multiple answer/message pairs with the `-r` option is not clear.

NAME

chmod - change file mode

SYNOPSIS

chmod [-R] *mode file* . . .
chmod [ugoa]{ + | - | =}[rwxlstugo] *file* . . .

DESCRIPTION

chmod changes or assigns the mode of a file. The mode of a file specifies its permissions and other attributes. The mode may be absolute or symbolic.

An absolute *mode* is specified using octal numbers:

chmod *nnnn file* . . .

where *n* is a number from 0 to 7. An absolute mode is constructed from the OR of any of the following modes:

4000	Set user ID on execution.
20#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory locking if # is 6, 4, 2, or 0. This bit is ignored if the file is a directory; it may be set or cleared only using the symbolic mode.
1000	Turn on sticky bit [(see chmod(2)].
0400	Allow read by owner.
0200	Allow write by owner.
0100	Allow execute (search in directory) by owner.
0070	Allow read, write, and execute (search) by group.
0007	Allow read, write, and execute (search) by others.

A symbolic *mode* is specified in the following format:

chmod [*who*] *operator* [*permission(s)*] *file* . . .

who is zero or more of the characters u, g, o, and a specifying whose permissions are to be changed or assigned:

u	user's permissions
g	group's permissions
o	others' permissions
a	all permissions (user, group, and other)

If *who* is omitted, it defaults to a.

operator is one of +, –, or =, signifying how permissions are to be changed:

+	Add permissions.
-	Take away permissions.
=	Assign permissions absolutely.

Unlike other symbolic operations, = has an absolute effect in that it resets all other bits. Omitting *permission*(s) is useful only with = to take away all permissions.

permission(s) is any compatible combination of the following letters:

r	read permission
w	write permission

x	execute permission
s	user or group set-ID
t	sticky bit
l	mandatory locking
u, g, o	indicate that *permission* is to be taken from the current user, group or other mode respectively.

Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (user, group, and others all have permission to read, write, and execute a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously.

The letter s is only meaningful with u or g, and t only works with u.

Mandatory file and record locking (l) refers to a file's ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID bit and enable a file to be locked on execution at the same time. The following examples, therefore, are invalid and elicit error messages:

```
chmod g+x,+l file
chmod g+s,+l file
```

Only the owner of a file or directory (or the super-user) may change that file's or directory's mode. Only the super-user may set the sticky bit on a non-directory file. If you are not super-user, chmod will mask the sticky-bit but will not return an error. In order to turn on a file's set-group-ID bit, your own group ID must correspond to the file's and group execution must be set.

The -R option recursively descends through directory arguments, setting the mode for each file as described above.

EXAMPLES

Deny execute permission to everyone:

```
chmod a-x file
```

Allow read permission to everyone:

```
chmod 444 file
```

Make a file readable and writable by the group and others:

```
chmod go+rw file
chmod 066 file
```

Cause a file to be locked during access:

```
chmod +l file
```

Allow everyone to read, write, and execute the file and turn on the set group-ID.

```
chmod =rwx,g+s file
chmod 2777 file
```

Absolute changes don't work for the set-group-ID bit of a directory. You must use `g+s` or `g-s`.

SEE ALSO

`ls`(1) `chmod`(2).

NOTES

`chmod` permits you to produce useless modes so long as they are not illegal (for example, making a text file executable).

NAME

chown - change file owner

SYNOPSIS

chown [-R] [-h] *owner file* ...

DESCRIPTION

chown changes the owner of the *file*s to *owner*. The *owner* may be either a decimal user ID or a login name found in /etc/passwd file.

If chown is invoked by other than the super-user, the set-user-ID bit of the file mode, 04000, is cleared.

Only the owner of a file (or the super-user) may change the owner of that file.

Valid options to chown are:

-R Recursive. chown descends through the directory, and any subdirectories, setting the ownership ID as it proceeds. When symbolic links are encountered, they are traversed.

-h If the file is a symbolic link, change the owner of the symbolic link. Without this option, the owner of the file referenced by the symbolic link is changed.

The operating system has a configuration option {_POSIX_CHOWN_RESTRICTED}, to restrict ownership changes. When this option is in effect the owner of the file is prevented from changing the owner ID of the file. Only the super-user can arbitrarily change owner IDs whether this option is in effect or not.

FILES

/etc/passwd

SEE ALSO

chgrp(1), chmod(1), chown(2), passwd(4).

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the ls -l command leads you to believe.

NAME

chown - change file owner

SYNOPSIS

/usr/ucb/chown [-fhR] *owner*[.*group*] *file* . . .

DESCRIPTION

chown changes the owner of the *files* to *owner*. The *owner* may be either a decimal user ID or a login name found in /etc/passwd file. The optional .*group* suffix may be used to change the group at the same time.

If chown is invoked by other than the super-user, the set-user-ID bit of the file mode, 04000, is cleared.

Only the super-user may change the owner of a file.

Valid options to chown are:

-f Suppress error reporting

-h If the file is a symbolic link, change the owner of the symbolic link. Without this option, the owner of the file referenced by the symbolic link is changed.

-R Descend recursively through directories setting the ownership ID of all files in each directory entered.

FILES

/etc/group
/etc/passwd

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the ls -l command leads you to believe.

SEE ALSO

chgrp(1), chmod(1), chown(2), passwd(4).

NAME

`chroot` - change root directory for a command

SYNOPSIS

`/usr/sbin/chroot` *newroot command*

DESCRIPTION

`chroot` causes the given command to be executed relative to the new root. The meaning of any initial slashes (`/`) in the path names is changed for the command and any of its child processes to *newroot* . Furthermore, upon execution, the initial working directory is *newroot* .

Notice, however, that if you redirect the output of the command to a file:

`chroot` *newroot command* `>x`

will create the file `x` relative to the original root of the command, not the new one.

The new root path name is always relative to the current root: even if a `chroot` is currently in effect, the *newroot* argument is relative to the current root of the running process.

This command can be run only by the super-user.

SEE ALSO

`cd`(1), `chroot`(2).

NOTES

One should exercise extreme caution when referencing device files in the new root file system.

NAME

chrtbl - generate character classification and conversion tables

SYNOPSIS

chrtbl [*file*]

DESCRIPTION

The chrtbl command creates two tables containing information on character classification, upper/lower-case conversion, character-set width, and numeric formatting. One table is an array of (257*2) + 7 bytes that is encoded so a table lookup can be used to determine the character classification of a character, convert a character [see ctype(3C)], and find the byte and screen width of a character in one of the supplementary code sets. The other table contains information about the format of non-monetary numeric quantities: the first byte specifies the decimal delimiter; the second byte specifies the thousands delimiter; and the remaining bytes comprise a null terminated string indicating the grouping (each element of the string is taken as an integer that indicates the number of digits that comprise the current group in a formatted non-monetary numeric quantity).

chrtbl reads the user-defined character classification and conversion information from *file* and creates three output files in the current directory. To construct *file*, use the file supplied in /usr/lib/locale/C/chrtbl_C as a starting point. You may add entries, but do not change the original values supplied with the system. For example, for other locales you may wish to add eight-bit entries to the ASCII definitions provided in this file.

One output file, ctype.c (a C-language source file), contains a (257*2)+7-byte array generated from processing the information from *file*. You should review the content of ctype.c to verify that the array is set up as you had planned. (In addition, an application program could use ctype.c.) The first 257 bytes of the array in ctype.c are used for character classification. The characters used for initializing these bytes of the array represent character classifications that are defined in /usr/include/ctype.h; for example, _L means a character is lower case and _S|_B means the character is both a spacing character and a blank. The second 257 bytes of the array are used for character conversion. These bytes of the array are initialized so that characters for which you do not provide conversion information will be converted to themselves. When you do provide conversion information, the first value of the pair is stored where the second one would be stored normally, and vice versa; for example, if you provide <0x41 0x61>, then 0x61 is stored where 0x41 would be stored normally, and 0x61 is stored where 0x41 would be stored normally. The last 7 bytes are used for character width information for up to three supplementary code sets.

The second output file (a data file) contains the same information, but is structured for efficient use by the character classification and conversion routines (see ctype(3C)). The name of this output file is the value you assign to the keyword LC_CTYPE read in from *file*. Before this file can be used by the character classification and conversion routines, it must be installed in the /usr/lib/locale/*locale* directory with the name LC_CTYPE by someone who is super-user or a member of group bin. This file must be readable by user, group, and other; no other permissions should be set. To use the character classification

and conversion tables in this file, set the LC_CTYPE environment variable appropriately (see environ(5) or setlocale(3C)).

The third output file (a data file) is created only if numeric formatting information is specified in the input file. The name of this output file is the value you assign to the keyword LC_NUMERIC read in from *file*. Before this file can be used, it must be installed in the /usr/lib/locale/*locale* directory with the name LC_NUMERIC by someone who is super-user or a member of group bin. This file must be readable by user, group, and other; no other permissions should be set. To use the numeric formatting information in this file, set the LC_NUMERIC environment variable appropriately (see environ(5) or setlocale(3C)).

The name of the locale where you install the files LC_CTYPE and LC_NUMERIC should correspond to the conventions defined in *file*. For example, if French conventions were defined, and the name for the French locale on your system is french, then you should install the files in /usr/lib/locale/french.

If no input file is given, or if the argument "-" is encountered, chrtbl reads from standard input.

The syntax of *file* allows the user to define the names of the data files created by chrtbl, the assignment of characters to character classifications, the relationship between upper and lower-case letters, byte and screen widths for up to three supplementary code sets, and three items of numeric formatting information: the decimal delimiter, the thousands delimiter and the grouping. The keywords recognized by chrtbl are:

LC_CTYPE	name of the data file created by chrtbl to contain character classification, conversion, and width information
isupper	character codes to be classified as upper-case letters
islower	character codes to be classified as lower-case letters
isdigit	character codes to be classified as numeric
isspace	character codes to be classified as spacing (delimiter) characters
ispunct	character codes to be classified as punctuation characters
iscntrl	character codes to be classified as control characters
isblank	character code for the blank (space) character
isxdigit	character codes to be classified as hexadecimal digits
ul	relationship between upper- and lower-case characters
cswidth	byte and screen width information (by default, each is one character wide)
LC_NUMERIC	name of the data file created by chrtbl to contain numeric formatting information
decimal_point	decimal delimiter
thousands_sep	thousands delimiter
grouping	string in which each element is taken as an integer that indicates the number of digits that comprise the current group in a formatted non-monetary numeric quantity.

Any lines with the number sign (#) in the first column are treated as comments and are ignored. Blank lines are also ignored.

Characters for `isupper`, `islower`, `isdigit`, `isspace`, `ispunct`, `iscntrl`, `isblank`, `isxdigit`, and `ul` can be represented as a hexadecimal or octal constant (for example, the letter `a` can be represented as `0x61` in hexadecimal or `0141` in octal). Hexadecimal and octal constants may be separated by one or more space and/or tab characters.

The dash character (-) may be used to indicate a range of consecutive numbers. Zero or more space characters may be used for separating the dash character from the numbers.

The backslash character (\) is used for line continuation. Only a carriage return is permitted after the backslash character.

The relationship between upper- and lower-case letters (`ul`) is expressed as ordered pairs of octal or hexadecimal constants: *<upper-case_character lower-case_character>*. These two constants may be separated by one or more space characters. Zero or more space characters may be used for separating the angle brackets (< >) from the numbers.

The following is the format of an input specification for `cswidth`:

n1:s1,n2:s2,n3:s3

where,

n1 byte width for supplementary code set 1, required *s1* screen width for supplementary code set 1 *n2* byte width for supplementary code set 2 *s2* screen width for supplementary code set 2 *n3* byte width for supplementary code set 3 *s3* screen width for supplementary code set 3

`decimal_point` and `thousands_sep` are specified by a single character that gives the delimiter. `grouping` is specified by a quoted string in which each member may be in octal or hex representation. For example, `\3` or `\x3` could be used to set the value of a member of the string to 3.

EXAMPLE

The following is an example of an input file used to create the USA-ENGLISH code set definition table in a file named `usa` and the non-monetary numeric formatting information in a file name `num-usa`.

```
LC_CTYPE  usa
isupper   0x41 - 0x5a
islower   0x61 - 0x7a
isdigit   0x30 - 0x39
isspace   0x20 0x9 - 0xd
ispunct   0x21 - 0x2f   0x3a - 0x40 \
          0x5b - 0x60   0x7b - 0x7e
iscntrl   0x0 - 0x1f    0x7f
isblank   0x20
isxdigit  0x30 - 0x39   0x61 - 0x66 \
          0x41 - 0x46
ul       <0x41 0x61> <0x42 0x62> <0x43 0x63>  \
         <0x44 0x64> <0x45 0x65> <0x46 0x66>  \
         <0x47 0x67> <0x48 0x68> <0x49 0x69>  \
         <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c>  \
```

```
                <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f>  \
                <0x50 0x70> <0x51 0x71> <0x52 0x72>  \
                <0x53 0x73> <0x54 0x74> <0x55 0x75>  \
                <0x56 0x76> <0x57 0x77> <0x58 0x78>  \
                <0x59 0x79> <0x5a 0x7a>
        cswidth          1:1,0:0,0:0
        LC_NUMERIC  num_usa
        decimal_point           .
        thousands_sep           ,
        grouping                "\3"
```

FILES

`/usr/lib/locale/`*locale*`/LC_CTYPE`
: data files containing character classification, conversion, and character-set width information created by `chrtbl`

`/usr/lib/locale/`*locale*`/LC_NUMERIC`
: data files containing numeric formatting information created by `chrtbl`

`/usr/include/ctype.h`
: header file containing information used by character classification and conversion routines

`/usr/lib/locale/C/chrtbl_C`
: input file used to construct `LC_CTYPE` and `LC_NUMERIC` in the default locale.

SEE ALSO

`ctype`(3C), `setlocale`(3C), `environ`(5).

DIAGNOSTICS

The error messages produced by `chrtbl` are intended to be self-explanatory. They indicate errors in the command line or syntactic errors encountered within the input file.

NOTES

Changing the files in `/usr/lib/locale/C` will cause the system to behave unpredictably.

Only lower case hex values are allowed in input specification.

NAME

ckbinarsys - determine whether remote system can accept binary messages

SYNOPSIS

ckbinarsys [-S] -s *remote_system_name* -t *content_type*

DESCRIPTION

Because rmail can transport binary data, it may be important to determine whether a particular remote system (typically the next hop) can handle binary data via the chosen transport layer agent (uux, SMTP, and so on)

ckbinarsys consults the file /etc/mail/binarsys for information on a specific remote system. ckbinarsys returns its results via an appropriate exit code. An exit code of zero implies that it is OK to send a message with the indicated content type to the system specified. An exit code other than zero indicates that the remote system cannot properly handle messages with binary content.

The absence of the binarsys file will cause ckbinarsys to exit with a non-zero exit code.

Command-line arguments are:

-s *remote_system_name*
: Name of remote system to look up in /etc/mail/binarsys

-t *content_type*
: Content type of message to be sent. When invoked by rmail, this will be one of two strings: text or binary, as determined by mail independent of any Content-Type: header lines that may be present within the message header. All other arguments are treated as equivalent to binary.

-S
: Normally, ckbinarsys will print a message (if the binary mail is rejected) which would be suitable for rmail to return in the negative acknowledgement mail. When -S is specified, no message will be printed.

FILES

/etc/mail/binarsys
/usr/lib/mail/surrcmd/ckbinarsys

SEE ALSO

mail(1), uux(1), binarsys(4), mailsurr(4).

NAME

ckbupscd - check file system backup schedule

SYNOPSIS

ckbupscd [-m]

DESCRIPTION

ckbupscd consults the file /etc/bupsched and prints the file system lists from lines with date and time specifications matching the current time. If the -m flag is present, an introductory message in the output is suppressed so that only the file system lists are printed. Entries in the bupsched file are printed under the control of cron.

The file bupsched should contain lines of four or more fields, separated by spaces or tabs. The first three fields (the schedule fields) specify a range of dates and times. The rest of the fields constitute a list of names of file systems to be printed if ckbupscd is run at some time within the range given by the schedule fields. The general format is:

time[,*time*] *day*[,*day*] *month*[,*month*] *fsyslist*

where:

time Specifies an hour of the day (0 through 23), matching any time within that hour, or an exact time of day (0:00 through 23:59).

day Specifies a day of the week (sun through sat) or day of the month (1 through 31).

month Specifies the month in which the time and day fields are valid. Legal values are the month numbers (1 through 12).

fsyslist The rest of the line is taken to be a file system list to print.

Multiple time, day, and month specifications may be separated by commas, in which case they are evaluated left to right.

An asterisk (*) always matches the current value for the field in which it appears.

A line beginning with a sharp sign (#) is interpreted as a comment and ignored.

The longest line allowed (including continuations) is 1024 characters.

EXAMPLES

The following are examples of lines which could appear in the /etc/bupsched file.

```
06:00-09:00  fri  1,2,3,4,5,6,7,8,9,10,11  /applic
```

Prints the file system name */applic* if ckbupscd is run between 6:00 A.M. and 9:00 A.M. any Friday during any month except December.

```
00:00-06:00,16:00-23:59  1,2,3,4,5,6,7  1,8  /
```

Prints a reminder to backup the root (/) file system if ckbupscd is run between the times of 4:00 P.M. and 6:00 A.M. during the first week of August or January.

FILES

/etc/bupsched specification file containing times and file system to back up

NOTES

`ckbupscd` will report file systems due for backup if invoked any time in the window. It does not know that backups may have just been done.

`ckbupscd` will be removed in the next release of System V.

SEE ALSO

`cron`(1M), `echo`(1), `sh`(1), `sysadm`(1).

NAME

ckdate, errdate, helpdate, valdate - prompts for and validates a date

SYNOPSIS

ckdate [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

errdate [-W] [-e *error*] [-f *format*]
helpdate [-W] [-h *help*] [-f *format*]
valdate [-f *format*] *input*

DESCRIPTION

ckdate prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a date, text for help and error messages, and a default value (which will be returned if the user responds with a carriage return). The user response must match the defined format for a date.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) will be displayed.

Three visual tool modules are linked to the ckdate command. They are errdate (which formats and displays an error message), helpdate (which formats and displays a help message), and valdate (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt. When *format* is defined in the errdate and helpdate modules, the messages will describe the expected format.

The options and arguments for this command are:

-Q Specifies that quit will not be allowed as a valid response.

-W *width* Specifies that prompt, help and error messages will be formatted to a line length of *width*.

-f *format* Specifies the format against which the input will be verified. Possible formats and their definitions are:
%b = abbreviated month name
%B = full month name
%d = day of month (01 - 31)
%D = date as %m/%d/%y (the default format)
%e = day of month (1 - 31; single digits are preceded by a blank)
%h = abbreviated month name (jan, feb, mar)
%m = month number (01 - 12)
%y = year within century (e.g. 89)
%Y = year as CCYY (e.g. 1989)

-d *default* Defines the default value as *default*.
The default does not have to meet the format criteria.

`-h` *help*	Defines the help messages as *help*.
`-e` *error*	Defines the error message as *error*.
`-p` *prompt*	Defines the prompt message as *prompt*.
`-k` *pid*	Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.
`-s` *signal*	Specifies that the process ID *pid* defined with the `-k` option is to be sent signal `signal` when quit is chosen. If no signal is specified, `SIGTERM` is used.
input	Input to be verified against format criteria.

EXIT CODES

`0` = Successful execution
`1` = EOF on input
`2` = Usage error
`3` = User termination (quit)
`4` = Garbled format argument

NOTES

The default prompt for `ckdate` is:

```
Enter the date [?,q]:
```

The default error message is:

```
ERROR - Please enter a date, using the following format: <format>.
```

The default help message is:

```
Please enter a date, using the following format: <format>.
```

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valdate` module will not produce any output. It returns zero for success and non-zero for failure.

NAME

`ckgid`, `errgid`, `helpgid`, `valgid` - prompt for and validate a group ID

SYNOPSIS

`ckgid` [`-Q`] [`-W` *width*] [`-m`] [`-d` *default*] [`-h` *help*] [`-e` *error*] [`-p` *prompt*] [`-k` *pid* [`-s` *signal*]]

`errgid` [`-W` *width*] [`-e` *error*]

`helpgid` [`-W` *width*] [`-m`] [`-h` *help*]

`valgid` *input*

DESCRIPTION

`ckgid` prompts a user and validates the response. It defines, among other things, a prompt message whose response should be an existing group ID, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The `-W` option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Three visual tool modules are linked to the `ckgid` command. They are `errgid` (which formats and displays an error message), `helpgid` (which formats and displays a help message), and `valgid` (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt.

The options and arguments for this command are:

`-Q` Do not allow quit as a valid response.

`-W` Use *width* as the line length for prompt, help, and error messages.

`-m` Display a list of all groups when help is requested or when the user makes an error.

`-d` The default value is *default*. The default is not validated and so does not have to meet any criteria.

`-h` The help message is *help*.

`-e` The error message is *error*.

`-p` The prompt message is *prompt*.

`-k` Send process ID *pid* a signal if the user chooses to abort.

`-s` When quit is chosen, send *signal* to the process whose *pid* is specified by the `-k` option. If no signal is specified, use `SIGTERM`.

input Input to be verified against `/etc/group`

EXIT CODES

0 = Successful execution
1 = EOF on input
2 = Usage error
3 = User termination (quit)

NOTES

The default prompt for `ckgid` is:

```
Enter the name of an existing group [?,q]
```

The default error message is:

```
ERROR - Please enter the name of an existing group.
```
(if the `-m` *option of* `ckgid` *is used, a list of valid groups is displayed here)*

The default help message is:

```
Please enter an existing group name.
```
(if the `-m` *option of* `ckgid` *is used, a list of valid groups is displayed here)*

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valgid` module does not produce any output. It returns zero for success and non-zero for failure.

NAME

ckint - display a prompt; verify and return an integer value

SYNOPSIS

ckint [-Q] [-W *width*] [-b *base*] [-d *default*] [-h *help*] [-e *error*]
[-p *prompt*] [-k *pid* [-s *signal*]]

errint [-W *width*] [-b *base*] [-e *error*]

helpint [-W *width*] [-b *base*] [-h *help*]

valint [-b *base*] *input*

DESCRIPTION

ckint prompts a user, then validates the response. It defines, among other things, a prompt message whose response should be an integer, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Three visual tool modules are linked to the ckint command. They are errint (which formats and displays an error message), helpint (which formats and displays a help message), and valint (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt. When *base* is defined in the errint and helpint modules, the messages includes the expected base of the input.

The options and arguments for this command are:

-Q Do not allow quit as a valid response.

-W Use *width* as the line length for prompt, help, and error messages.

-b The base for input is *base*. Must be 2 to 36, default is 10.

-d The default value is *default*. The default is not validated and so does not have to meet any criteria.

-h The help message is *help*.

-e The error message is *error*.

-p The prompt message is *prompt*.

-k Send process ID *pid* a signal if the user chooses to abort.

-s When quit is chosen, send *signal* to the process whose *pid* is specified by the -k option. If no signal is specified, use SIGTERM.

input Input to be verified against *base* criterion.

EXIT CODES

0 = Successful execution
1 = EOF on input
2 = Usage error
3 = User termination (quit)

NOTES

The default base 10 prompt for `ckint` is:

```
Enter an integer [?,q]
```

The default base 10 error message is:

```
ERROR - Please enter an integer.
```

The default base 10 help message is:

```
Please enter an integer.
```

The messages are changed from "`integer`" to "`base` *base* `integer`" if the base is set to a number other than 10.

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valint` module does not produce any output. It returns zero for success and non-zero for failure.

NAME

ckitem - build a menu; prompt for and return a menu item

SYNOPSIS

ckitem [-Q] [-W *width*] [-uno] [-f *file*] [-l *label*]
[[-i *invis*] [-i *invis*] . . .] [-m *max*] [-d *default*] [-h *help*] [-e *error*]
[-p *prompt*] [-k *pid* [-s *signal*]] [*choice1 choice2* . . .]

erritem [-W *width*] [-e *error*] [*choice1 choice2* . . .]

helpitem [-W *width*] [-h *help*] [*choice1 choice2* . . .]

DESCRIPTION

ckitem builds a menu and prompts the user to choose one item from a menu of items. It then verifies the response. Options for this command define, among other things, a prompt message whose response is a menu item, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

By default, the menu is formatted so that each item is prepended by a number and is printed in columns across the terminal. Column length is determined by the longest choice. Items are alphabetized.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Two visual tool modules are linked to the ckitem command. They are erritem (which formats and displays an error message) and helpitem (which formats and displays a help message). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt. When *choice* is defined in these modules, the messages describe the available menu choice (or choices).

The options and arguments for this command are:

-Q Do not allow quit as a valid response.

-W Use *width* as the line length for prompt, help, and error messages.

-u Display menu items as an unnumbered list.

-n Do not display menu items in alphabetical order.

-o Return only one menu token.

-f *file* contains a list of menu items to be displayed. [The format of this file is: *token<tab>description*. Lines beginning with a pound sign ("#") are comments and are ignored.]

-l Print *label* above the menu.

-i *invis* specifies invisible menu choices (choices not to be printed in the menu). For example, "all" used as an invisible choice would mean it is a valid option but does not appear in the menu. Any number of invisible choices may be defined. Invisible choices should be made known to a user

either in the prompt or in a help message.

`-m` The maximum number of menu choices allowed is *m*.

`-d` The default value is *default*. The default is not validated and so does not have to meet any criteria.

`-h` The help message is *help*.

`-e` The error message is *error*.

`-p` The prompt message is *prompt*.

`-k` Send process ID *pid* a signal if the user chooses to abort.

`-s` When quit is chosen, send *signal* to the process whose *pid* is specified by the `-k` option. If no signal is specified, use `SIGTERM`.

choice Defines menu items. Items should be separated by white space or newline.

EXIT CODES

`0` = Successful execution
`1` = EOF on input
`2` = Usage error
`3` = User termination (quit)
`4` = No choices from which to choose

NOTES

The user may input the number of the menu item if choices are numbered or as much of the string required for a unique identification of the item. Long menus are paged with 10 items per page.

When menu entries are defined both in a file (by using the `-f` option) and also on the command line, they are usually combined alphabetically. However, if the `-n` option is used to suppress alphabetical ordering, then the entries defined in the file are shown first, followed by the options defined on the command line.

The default prompt for `ckitem` is:

```
Enter selection [?,??,q]:
```

One question mark gives a help message and then redisplays the prompt. Two question marks gives a help message and then redisplays the menu label, the menu and the prompt.

The default error message is:

```
ERROR - Does not match an available menu selection.
Enter one of the following:
- the number of the menu item you wish to select
- the token associated withe the menu item,
- partial string which uniquely identifies the token
   for the menu item
- ?? to reprint the menu
```

The default help message is:

```
Enter one of the following:
- the number of the menu item you wish to select
- the token associated with the menu item,
- partial string which uniquely identifies the token
```

```
    for the menu item
— ?? to reprint the menu
```

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`.

NAME

ckkeywd - prompt for and validate a keyword

SYNOPSIS

ckkeywd [-Q] [-W *width*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*]
[-k *pid* [-s *signal*]] [*keyword* . . .]

DESCRIPTION

ckkeywd prompts a user and validates the response. It defines, among other things, a prompt message whose response should be one of a list of keywords, text for help and error messages, and a default value (which is returned if the user responds with a RETURN). The answer returned from this command must match one of the defined list of keywords.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

-Q	Do not allow quit as a valid response.
-W	Use *width* as the line length for prompt, help, and error messages.
-d	The default value is *default*. The default is not validated and so does not have to meet any criteria.
-h	The help message is *help*.
-e	The error message is *error*.
-p	The prompt message is *prompt*.
-k	Send process ID *pid* a signal if the user chooses to abort.
-s	When quit is chosen, send *signal* to the process whose *pid* is specified by the -k option. If no signal is specified, use SIGTERM.
keyword	The keyword, or list of keywords, against which the answer is to be verified is *keyword*.

EXIT CODES

0 = Successful execution
1 = EOF on input
2 = Usage error
3 = User termination (quit)
4 = No keywords from which to choose

NOTES

The default prompt for ckkeywd is:

```
Enter appropriate value [keyword[, . . . ],?,q]
```

The default error message is:

```
ERROR - Please enter one of the following keywords:
keyword[, ...]
```

The default help message is:

```
Please enter one of the following keywords:
keyword[, ...]
```

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`.

NAME

ckpath - display a prompt; verify and return a pathname

SYNOPSIS

ckpath [-Q] [-W *width*] [-a | l] [*file_options*] [-rtwx] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

errpath [-W *width*] [-a | l] [*file_options*] [-rtwx] [-e *error*]

helppath [-W *width*] [-a | l] [*file_options*] [-rtwx] [-h *help*]

valpath [-a | l] [*file_options*] [-rtwx] *input*

DESCRIPTION

ckpath prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a pathname, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

The pathname must obey the criteria specified by the first group of options. If no criteria are defined, the pathname must be for a normal file that does not yet exist. If neither -a (absolute) or -l (relative) is given, then either is assumed to be valid.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Three visual tool modules are linked to the ckpath command. They are errpath (which formats and displays an error message), helppath (which formats and displays a help message), and valpath (which validates a response). These modules should be used in conjunction with FACE objects. In this instance, the FACE object defines the prompt.

The options and arguments for this command are:

-Q Do not allow quit as a valid response.

-W Use *width* as the line length for prompt, help, and error messages.

-a Pathname must be an absolute path.

-l Pathname must be a relative path.

-r Pathname must be readable.

-t Pathname must be creatable (touchable). Pathname is created if it does not already exist.

-w Pathname must be writable.

-x Pathname must be executable.

-d The default value is *default*. The default is not validated and so does not have to meet any criteria.

`-h` The help message is *help*.

`-e` The error message is *error*.

`-p` The prompt message is *prompt*.

`-k` Send process ID *pid* a signal if the user chooses to abort.

`-s` When quit is chosen, send *signal* to the process whose *pid* is specified by the `-k` option. If no signal is specified, use `SIGTERM`.

input Input to be verified against validation options.

file_options are:

`-b` Pathname must be a block special file.

`-c` Pathname must be a character special file.

`-f` Pathname must be a regular file.

`-y` Pathname must be a directory.

`-n` Pathname must not exist (must be new).

`-o` Pathname must exist (must be old).

`-z` Pathname must be a file with the size greater than 0 bytes.

The following *file_options* are mutually exclusive: `-bcfy`, `-no`, `-nz`, `-bz`, `-cz`.

EXIT CODES

`0` = Successful execution
`1` = EOF on input
`2` = Usage error
`3` = User termination (quit)
`4` = Mutually exclusive options

NOTES

The text of the default messages for `ckpath` depends upon the criteria options that have been used. An example default prompt for `ckpath` (using the `-a` option) is:

```
Enter an absolute pathname [?,q]
```

An example default error message (using the `-a` option) is:

```
ERROR - Pathname must begin with a slash (/).
```

An example default help message is:

```
A pathname is a filename, optionally preceded by parent
directories.  The pathname you enter:
- must contain 1 to NAME_MAX characters
- must not contain a spaces or special characters
```

NAME_MAX is a system variable is defined in `limits.h`.

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valpath` module does not produce any output. It returns zero for success and non-zero for failure.

NAME

ckrange - prompts for and validates an integer

SYNOPSIS

ckrange [-Q] [-W *width*] [-l *lower*] [-u *upper*] [-b *base*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

errange [-W] [-l *lower*] [-u *upper*] [-e *error*]
helprange [-W] [-l *lower*] [-u *upper*] [-h *help*]
valrange [-l *lower*] [-u *upper*] [-b *base*] *input*

DESCRIPTION

ckrange prompts a user and validates the response. It defines, among other things, a prompt message whose response should be an integer in the range specified, text for help and error messages, and a default value (which will be returned if the user responds with a carriage return).

This command also defines a range for valid input. If either the lower or upper limit is left undefined, then the range is bounded on only one end.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text will be inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) will be displayed.

Three visual tool modules are linked to the ckrange command. They are errange (which formats and displays an error message), helprange (which formats and displays a help message), and valrange (which validates a response). These modules should be used in conjunction with FACE objects. In this instance, the FACE object defines the prompt.

The options and arguments for this command are:

-Q Specifies that quit will not be allowed as a valid response.

-W Specifies that prompt, help and error messages will be formatted to a line length of *width*.

-l Defines the lower limit of the range as *lower*. Default is the machine's largest negative integer or long.

-u Defines the upper limit of the range as *upper*. Default is the machine's largest positive integer or long.

-b Defines the base for input. Must be 2 to 36, default is 10.

-d Defines the default value as *default*. The default is not validated and so does not have to meet any criteria.

-h Defines the help messages as *help*.

-e Defines the error message as *error*.

`-p` Defines the prompt message as *prompt*.

`-k` Specifies that process ID *pid* is to be sent a signal if the user chooses to abort.

`-s` Specifies that the process ID *pid* defined with the `-k` option is to be sent signal `signal` when quit is chosen. If no signal is specified, `SIGTERM` is used.

input Input to be verified against upper and lower limits and base.

EXIT CODES

`0` = Successful execution
`1` = EOF on input
`2` = Usage error
`3` = User termination (quit)

NOTES

The default base 10 prompt for `ckrange` is:

`Enter an integer between` *lower_bound* `and` *upper_bound* [q,?]:

The default base 10 error message is:

`ERROR - Please enter an integer between` *lower_bound* `and` *upper_bound*.

The default base 10 help message is:

`Please enter an integer between` *lower_bound* `and` *upper_bound*.

The messages are changed from "integer" to "base *base* integer" if the base is set to a number other than 10.

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valrange` module will not produce any output. It returns zero for success and non-zero for failure.

NAME

ckstr - display a prompt; verify and return a string answer

SYNOPSIS

ckstr [-Q] [-W *width*] [[-r *regexp*] [-r *regexp*] . . .] [-l *length*]
[-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

errstr [-W *width*] [-e *error*] [[-r *regexp*] [-r *regexp*] . . .] [-l *length*]

helpstr [-W *width*] [-h *help*] [[-r *regexp*] [-r *regexp*] . . .] [-l *length*]

valstr *input* [[-r *regexp*] [-r *regexp*] . . .] [-l *length*]

DESCRIPTION

ckstr prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a string, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

The answer returned from this command must match the defined regular expression and be no longer than the length specified. If no regular expression is given, valid input must be a string with a length less than or equal to the length defined with no internal, leading or trailing white space. If no length is defined, the length is not checked. Either a regular expression or a length must be given with the command.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Three visual tool modules are linked to the ckstr command. They are errstr (which formats and displays an error message), helpstr (which formats and displays a help message), and valstr (which validates a response). These modules should be used in conjunction with FACE objects. In this instance, the FACE object defines the prompt.

The options and arguments for this command are:

-Q Do not allow quit as a valid response.

-W Use *width* as the line length for prompt, help, and error messages.

-r Validate the input against regular expression *regexp*. May include white space. If multiple expressions are defined, the answer need match only one of them.

-l The maximum length of the input is *length*.

-d The default value is *default*. The default is not validated and so does not have to meet any criteria.

-h The help message is *help*.

`-e` The error message is *error*.

`-p` The prompt message is *prompt*.

`-k` Send process ID *pid* a signal if the user chooses to abort.

`-s` When quit is chosen, send *signal* to the process whose *pid* is specified by the `-k` option. If no signal is specified, use `SIGTERM`.

input Input to be verified against format length and/or regular expression criteria.

EXIT CODES

`0` = Successful execution
`1` = EOF on input
`2` = Usage error
`3` = User termination (quit)

NOTES

The default prompt for `ckstr` is:

```
Enter an appropriate value [?,q]
```

The default error message is dependent upon the type of validation involved. The user is told either that the length or the pattern matching failed.

The default help message is also dependent upon the type of validation involved. If a regular expression has been defined, the message is:

```
Please enter a string which matches the following pattern:
regexp
```

Other messages define the length requirement and the definition of a string.

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valstr` module does not produce any output. It returns zero for success and non-zero for failure.

Unless a "`q`" for "quit" is disabled by the `-Q` option, a single "`q`" to the following

```
ckstr -rq
```

is treated as a "quit" and not as a pattern match.

NAME

cktime - display a prompt; verify and return a time of day

SYNOPSIS

cktime [-Q] [-W *width*] [-f *format*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

errtime [-W *width*] [-e *error*] [-f *format*]

helptime [-W *width*] [-h *help*] [-f *format*]

valtime [-f *format*] *input*

DESCRIPTION

cktime prompts a user and validates the response. It defines, among other things, a prompt message whose response should be a time, text for help and error messages, and a default value (which is returned if the user responds with a RETURN). The user response must match the defined format for the time of day.

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Three visual tool modules are linked to the cktime command. They are errtime (which formats and displays an error message), helptime (which formats and displays a help message), and valtime (which validates a response). These modules should be used in conjunction with FMLI objects. In this instance, the FMLI object defines the prompt. When *format* is defined in the errtime and helptime modules, the messages describe the expected format.

The options and arguments for this command are:

-Q Do not allow quit as a valid response.

-W Use *width* as the line length for prompt, help, and error messages.

-f Verify the input against *format*. Possible formats and their definitions are:

%H = hour (00 - 23)
%I = hour (00 - 12)
%M = minute (00 - 59)
%p = ante meridian or post meridian
%r = time as %I:%M:%S %p
%R = time as %H:%M (the default format)
%S = seconds (00 - 59)
%T = time as %H:%M:%S

-d The default value is *default*. The default is not validated and so does not have to meet any criteria.

-h The help message is *help*.

`-e` The error message is *error*.

`-p` The prompt message is *prompt*.

`-k` *pid* Send process ID *pid* a signal if the user chooses to abort.

`-s` *signal*
When quit is chosen, send *signal* to the process whose *pid* is specified by the `-k` option. If no signal is specified, use `SIGTERM`.

input Input to be verified against format criteria.

EXIT CODES

`0` = Successful execution
`1` = EOF on input
`2` = Usage error
`3` = User termination (quit)
`4` = Garbled format argument

NOTES

The default prompt for `cktime` is:

```
Enter a time of day [?,q]
```

The default error message is:

```
ERROR - Please enter the time of day. Format is format.
```

The default help message is:

```
Please enter the time of day. Format is format.
```

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valtime` module does not produce any output. It returns zero for success and non-zero for failure.

NAME

ckuid - prompt for and validate a user ID

SYNOPSIS

ckuid [-Q] [-W *width*] [-m] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*] [-k *pid* [-s *signal*]]

erruid [-W *width*] [-e *error*]

helpuid [-W *width*] [-m] [-h *help*]

valuid *input*

DESCRIPTION

ckuid prompts a user and validates the response. It defines, among other things, a prompt message whose response should be an existing user ID, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. When a tilde is placed at the beginning or end of a message definition, the default text is inserted at that point, allowing both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Three visual tool modules are linked to the ckuid command. They are erruid (which formats and displays an error message), helpuid (which formats and displays a help message), and valuid (which validates a response). These modules should be used in conjunction with FML objects. In this instance, the FML object defines the prompt.

The options and arguments for this command are:

-Q Do not allow quit as a valid response.

-W Use *width* as the line length for prompt, help, and error messages.

-m Display a list of all logins when help is requested or when the user makes an error.

-d The default value is *default*. The default is not validated and so does not have to meet any criteria.

-h The help message is *help*.

-e The error message is *error*.

-p The prompt message is *prompt*.

-k Send process ID *pid* a signal if the user chooses to abort.

-s When quit is chosen, send *signal* to the process whose *pid* is specified by the -k option. If no signal is specified, use SIGTERM.

input Input to be verified against /etc/passwd.

EXIT CODES

0 = Successful execution
1 = EOF on input
2 = Usage error
3 = User termination (quit)

NOTES

The default prompt for `ckuid` is:

```
Enter the login name of an existing user [?,q]
```

The default error message is:

```
ERROR - Please enter the login name of an existing user.
```

(If the `-m` *option of* `ckuid` *is used, a list of valid users is also displayed.)*

The default help message is:

```
Please enter the login name of an existing user.
```

(If the `-m` *option of* `ckuid` *is used, a list of valid users is also displayed.)*

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valuid` module does not produce any output. It returns zero for success and non-zero for failure.

NAME

ckyorn - prompt for and validate yes/no

SYNOPSIS

ckyorn [-Q] [-W *width*] [-d *default*] [-h *help*] [-e *error*] [-p *prompt*]
[-k *pid* [-s *signal*]]

erryorn [-W *width*] [-e *error*]

helpyorn [-W *width*] [-h *help*]

valyorn *input*

DESCRIPTION

ckyorn prompts a user and validates the response. It defines, among other things, a prompt message for a yes or no answer, text for help and error messages, and a default value (which is returned if the user responds with a RETURN).

All messages are limited in length to 70 characters and are formatted automatically. Any white space used in the definition (including newline) is stripped. The -W option cancels the automatic formatting. For the -h and -e options, placing a tilde at the beginning or end of a message definition causes the default text to be inserted at that point. This allows both custom text and the default text to be displayed.

If the prompt, help or error message is not defined, the default message (as defined under NOTES) is displayed.

Three visual tool modules are linked to the ckyorn command. They are erryorn (which formats and displays an error message), helpyorn (which formats and displays a help message), and valyorn (which validates a response). These modules should be used in conjunction with FACE objects. In this instance, the FACE object defines the prompt.

The options and arguments for this command are:

-Q Do not allow quit as a valid response.

-W Use *width* as the line length for prompt, help, and error messages.

-d The default value is *default*. The default is not validated and so does not have to meet any criteria.

-h The help message is *help*.

-e The error message is *error*.

-p The prompt message is *prompt*.

-k Send process ID *pid* a signal if the user chooses to abort.

-s When quit is chosen, send *signal* to the process whose *pid* is specified by the -k option. If no signal is specified, use SIGTERM.

input Input to be verified as y, yes, Y, Yes, YES or n, no, N, No, NO.

EXIT CODES

0 = Successful execution
1 = EOF on input
2 = Usage error
3 = User termination (quit)

NOTES

The default prompt for `ckyorn` is:

```
Yes or No [y,n,?,q]
```

The default error message is:

```
ERROR - Please enter yes or no.
```

The default help message is:

```
Enter y or yes if your answer is yes;
or no if your answer is no.
```

When the quit option is chosen (and allowed), `q` is returned along with the return code `3`. The `valyorn` module does not produce any output. It returns zero for success and non-zero for failure.

NAME

clear - clear the terminal screen

SYNOPSIS

clear

DESCRIPTION

clear clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen.

SEE ALSO

tput(1), terminfo(4)

NAME

cmp - compare two files

SYNOPSIS

cmp [-l] [-s] *file1 file2* [*skip1*] [*skip2*]

DESCRIPTION

The two files are compared. (If *file1* 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. *skip1* and *skip2* are initial byte offsets into *file1* and *file2* respectively, and may be either octal or decimal; a leading 0 denotes octal.

Options:

- -l Print the byte number (decimal) and the differing bytes (octal) for each difference.
- -s Print nothing for differing files; return codes only.

FILES

/usr/lib/locale/*locale*/LC_MESSAGES/uxcore.abi
language-specific message file [See LANG on environ(5).]

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.

NAME

`cof2elf` - COFF to ELF object file translation

SYNOPSIS

`cof2elf [-iqV] [-Q{yn}] [-s` *directory*`]` *files*

DESCRIPTION

`cof2elf` converts one or more COFF object *files* to ELF. This translation occurs in place, meaning the original file contents are modified. If an input file is an archive, each member will be translated as necessary, and the archive will be rebuilt with its members in the original order. `cof2elf` does not change input files that are not COFF.

Options have the following meanings.

`-i` Normally, the files are modified only when full translation occurs. Unrecognized data, such as unknown relocation types, are treated as errors and prevent translation. Giving the `-i` flag ignores these partial translation conditions and modifies the file anyway.

`-q` Normally, `cof2elf` prints a message for each file it examines, telling whether the file was translated, ignored, etc. The `-q` flag (for quiet) suppresses these messages.

`-Q`*arg* If *arg* is `y`, identification information about `cof2elf` will be added to the output files. This can be useful for software administration. Giving `n` for *arg* explicitly asks for no such information, which is the default behavior.

`-s`*directory* As mentioned above, `cof2elf` modifies the input files. This option saves a copy of the original files in the specified *directory*, which must exist. `cof2elf` does not save files it does not modify.

`-V` This flag tells `cof2elf` to print a version message on standard error.

SEE ALSO

`ld`(1), `elf`(3E), `a.out`(4), `ar`(4)

NOTES

Some debugging information is discarded. Although this does not affect the behavior of a running program, it may affect the information available for symbolic debugging.

`cof2elf` translates only COFF relocatable files. It does not translate executable or static shared library files for two main reasons. First, the operating system supports executable files and static shared libraries, making translation unnecessary. Second, those files have specific address and alignment constraints determined by the file format. Matching the constraints with a different object file format is problematic.

When possible, programmers should recompile their source code to build new object files. `cof2elf` is provided for those times when source code is unavailable.

NAME

col - filter reverse line-feeds

SYNOPSIS

col [-b] [-f] [-x] [-p]

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 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.

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 (\017) and SI (\016) 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 escape sequences unknown to it that are 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.

SEE ALSO

nroff(1), tbl(1), ascii(5).

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.

col cannot back up more than 128 lines or handle more than 800 characters per line.

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.

NAME

colltbl - create collation database

SYNOPSIS

colltbl [*file* | -]

DESCRIPTION

The colltbl command takes as input a specification file, *file*, that describes the collating sequence for a particular language and creates a database that can be read by strxfrm(3C) and strcoll(3C). strxfrm(3C) transforms its first argument and places the result in its second argument. The transformed string is such that it can be correctly ordered with other transformed strings by using strcmp(3C), strncmp(3C) or memcmp(3C). strcoll(3C) transforms its arguments and does a comparison.

If no input file is supplied, *stdin* is read.

The output file produced contains the database with collating sequence information in a form usable by system commands and routines. The name of this output file is the value you assign to the keyword codeset read in from *file*. Before this file can be used, it must be installed in the /usr/lib/locale/*locale* directory with the name LC_COLLATE by someone who is super-user or a member of group bin. *locale* corresponds to the language area whose collation sequence is described in *file*. This file must be readable by user, group, and other; no other permissions should be set. To use the collating sequence information in this file, set the LC_COLLATE environment variable appropriately (see environ(5) or setlocale(3C)).

The colltbl command can support languages whose collating sequence can be completely described by the following cases:

Ordering of single characters within the codeset. For example, in Swedish, V is sorted after U, before X and with W (V and W are considered identical as far as sorting is concerned).

Ordering of "double characters" in the collation sequence. For example, in Spanish, ch and ll are collated after c and l, respectively.

Ordering of a single character as if it consists of two characters. For example, in German, the "sharp s", β, is sorted as ss. This is a special instance of the next case below.

Substitution of one character string with another character string. In the example above, the string β is replaced with ss during sorting.

Ignoring certain characters in the codeset during collation. For example, if - were ignored during collation, then the strings re-locate and relocate would be equal.

Secondary ordering between characters. In the case where two characters are sorted together in the collation sequence, (i.e., they have the same "primary" ordering), there is sometimes a secondary ordering that is used if two strings are identical except for characters that have the same primary ordering. For example, in French, the letters e and è have the same primary ordering but e comes before è in the secondary ordering. Thus the

word `lever` would be ordered before `lèver`, but `lèver` would be sorted before `levitate`. (Note that if `e` came before `è` in the primary ordering, then `lèver` would be sorted after `levitate`.)

The specification file consists of three types of statements:

1. `codeset` *filename*

   filename is the name of the output file to be created by `colltbl`.

2. `order is` *order_list*

   order_list is a list of symbols, separated by semicolons, that defines the collating sequence. The special symbol, `...`, specifies symbols that are lexically sequential in a short-hand form. For example,

   ```
   order is    a;b;c;d;...;x;y;z
   ```

   would specify the list of lower_case letters. Of course, this could be further compressed to just `a;...;z`.

   A symbol can be up to two bytes in length and can be represented in any one of the following ways:

   the symbol itself (for example, `a` for the lower-case letter `a`),

   in octal representation (for example, `\141` or `0141` for the letter `a`), or

   in hexadecimal representation (for example, `\x61` or `0x61` for the letter `a`).

   Any combination of these may be used as well.

   The backslash character, `\`, is used for continuation. No characters are permitted after the backslash character.

   Symbols enclosed in parenthesis are assigned the same primary ordering but different secondary ordering. Symbols enclosed in curly brackets are assigned only the same primary ordering. For example,

   ```
   order is    a;b;c;ch;d;(e;è);f;...;z;\
               {1;...;9};A;...;Z
   ```

   In the above example, `e` and `è` are assigned the same primary ordering and different secondary ordering, digits 1 through 9 are assigned the same primary ordering and no secondary ordering. Only primary ordering is assigned to the remaining symbols. Notice how double letters can be specified in the collating sequence (letter `ch` comes between `c` and `d`).

   If a character is not included in the `order is` statement it is excluded from the ordering and will be ignored during sorting.

3. `substitute` *string* `with` *repl*

   The `substitute` statement substitutes the string *string* with the string *repl*. This can be used, for example, to provide rules to sort the abbreviated month names numerically:

```
substitute "Jan" with "01"
substitute "Feb" with "02"
      .
      .
      .
substitute "Dec" with "12"
```

A simpler use of the `substitute` statement that was mentioned above was to substitute a single character with two characters, as with the substitution of β with `ss` in German.

The `substitute` statement is optional. The `order is` and `codeset` statements must appear in the specification file.

Any lines in the specification file with a `#` in the first column are treated as comments and are ignored. Empty lines are also ignored.

EXAMPLE

The following example shows the collation specification required to support a hypothetical telephone book sorting sequence.

The sorting sequence is defined by the following rules:

a. Upper and lower case letters must be sorted together, but upper case letters have precedence over lower case letters.

b. All special characters and punctuation should be ignored.

c. Digits must be sorted as their alphabetic counterparts (for example, `0` as `zero`, `1` as `one`).

d. The `Ch`, `ch`, `CH` combinations must be collated between `C` and `D`.

e. `V` and `W`, `v` and `w` must be collated together.

The input specification file to `colltbl` will contain:

```
codeset    telephone

order is   A;a;B;b;C;c;CH;Ch;ch;D;d;E;e;F;f;\
           G;g;H;h:I;i;J;j;K;k;L;l;M;m;N;n;O;o;P;p;\
           Q;q;R;r;S;s;T;t;U;u;{V;W};{v;w};X;x;Y;y;Z;z

substitute "0" with "zero"
substitute "1" with "one"
substitute "2" with "two"
substitute "3" with "three"
substitute "4" with "four"
substitute "5" with "five"
substitute "6" with "six"
substitute "7" with "seven"
substitute "8" with "eight"
substitute "9" with "nine"
```

FILES

`/lib/locale/`*locale*`/LC_COLLATE`
`LC_COLLATE` database for *locale*

`/usr/lib/locale/C/colltbl_C`
input file used to construct `LC_COLLATE` in the default locale.

SEE ALSO

`memory`(3C), `setlocale`(3C), `strcoll`(3C), `string`(3C), `strxfrm`(3C), `environ`(5).

NAME

comb - combine SCCS deltas

SYNOPSIS

comb [-o] [-s] [-p*SID*] [-c*list*] *files*

DESCRIPTION

comb generates a shell procedure [see sh(1)] that, when run, reconstructs the given SCCS files. The reconstructed files are typically 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 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 argument is explained as if only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

-o For each get -e, 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 that, when run, produces a report that gives 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.

-p*SID* The SCCS identification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

-c*list* A *list* of deltas to be preserved. All other deltas are discarded. See get(1) for the syntax of a *list*.

If no keyletter arguments are specified, comb preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB	the reconstructed SCCS file
comb?????	temporary file

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sh(1), sccsfile(4).

DIAGNOSTICS

Use help(1) for explanations.

NOTES

comb may rearrange the shape of the tree of deltas.

comb may not save any space; in fact, it is possible for the reconstructed file to be larger than the original.

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 ASCII collating sequence [see sort(1)], 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 prints nothing.

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1)

NAME

compress, uncompress, zcat - compress, expand or display expanded files

SYNOPSIS

compress [-cfv] [-b *bits*] [*filename...*]
uncompress [-cv] [*filename...*]
zcat [*filename...*]

DESCRIPTION

compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with a .Z, extension. The ownership modes, access time and modification time will stay the same. If no files are specified, the standard input is compressed to the standard output.

The amount of compression obtained depends on the size of the input, the number of *bits* 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(1)], and takes less time to compute. The *bits* parameter specified during compression is encoded within the compressed file, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is subsequently allowed.

Compressed files can be restored to their original form using uncompress.

zcat produces uncompressed output on the standard output, but leaves the compressed .Z file intact.

OPTIONS

-c Write to the standard output; no files are changed. The nondestructive behavior of zcat is identical to that of 'uncompress -c'.

-f Force compression, even if the file does not actually shrink, or the corresponding .Z file already exists. Except when running in the background (under /usr/bin/sh), if -f is not given, prompt to verify whether an existing .Z file should be overwritten.

-v Verbose. Display the percentage reduction for each file compressed.

-b *bits* Set the upper limit (in bits) for common substring codes. *bits* must be between 9 and 16 (16 is the default). Lowering the number of bits will result in larger, less compressed files.

FILES

/usr/bin/sh

SEE ALSO

pack(1)

A Technique for High Performance Data Compression, Terry A. Welch, *IEEE Computer*, vol. 17, no. 6 (June 1984), pp. 8-19.

DIAGNOSTICS

Exit status is normally 0. If the last file was not compressed because it became larger, the status is 2. If an error occurs, exit status is 1.

Usage: compress [-fvc] [-b maxbits] [*filename...*]
Invalid options were specified on the command line.

`Missing maxbits`
: Maxbits must follow `-b`.

filename`: not in compressed format`
: The file specified to `uncompress` has not been compressed.

filename`: compressed with` *xx*`bits, can only handle` *yy*`bits`
: *filename* was compressed by a program that could deal with more *bits* than the compress code on this machine. Recompress the file with smaller *bits*.

filename`: already has .Z suffix -- no change`
: The file is assumed to be already compressed. Rename the file and try again.

filename`: 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 is 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, (such as a directory), it is left unaltered.

`-- has` *xx* `other links: unchanged`
: The input file has links; it is left unchanged. See `ln`(1) for more information.

`-- file unchanged`
: No savings are achieved by compression. The input remains uncompressed.

NOTES

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).

`compress` should be more flexible about the existence of the `.Z` suffix.

NAME

comsat, in.comsat - biff server

SYNOPSIS

in.comsat

DESCRIPTION

comsat is the server process which listens for reports of incoming mail and notifies users who have requested to be told when mail arrives. It is invoked as needed by inetd(1M), and times out if inactive for a few minutes.

comsat listens on a datagram port associated with the biff service specification [see services(4)] for one line messages of the form

user@mailbox-offset

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a biff y), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the From, To, Date, or Subject lines are not printed when displaying the message.

FILES

/var/adm/utmp who's logged on and on what terminals

SEE ALSO

services(4), inetd(1M).

NOTES

The message header filtering is prone to error.

NAME

cocreate, cosend, cocheck, coreceive, codestroy - communicate with a process

SYNOPSIS

cocreate [-r *rpath*] [-w *wpath*] [-i *id*] [-R *refname*] [-s *send_string*]
[-e *expect_string*] *command*

cosend [-n] *proc_id string*

cocheck *proc_id*

coreceive *proc_id*

codestroy [-R *refname*] *proc_id* [*string*]

DESCRIPTION

These co-processing functions provide a flexible means of interaction between FMLI and an independent process; especially, they enable FMLI to be responsive to asynchronous activity.

The cocreate function starts *command* as a co-process and initializes communications by setting up pipes between FMLI and the standard input and standard output of *command*. The argument *command* must be an executable and its arguments (if any). This means that *command* expects strings on its input (supplied by cosend) and sends information on its output that can be handled in various ways by FMLI. The following options can be used with cocreate.

-r *rpath* — If -r is specified, *rpath* is the pathname from which FMLI reads information. This option is usually used to set up communication with processes that naturally write to a certain path. If -r is not specified, cocreate will choose a unique path in /var/tmp.

-w *wpath* — If -w is specified, *wpath* is the pathname to which cosend writes information. This option is usually used so that one process can talk to many different FMLI processes through the same pipe. If -w is not specified, cocreate will choose a unique path in /var/tmp.

-i *id* — If -i is specified, *id* is an alternative name for the co-process initialized by this cocreate. If -i is not specified, *id* defaults to *command*. The argument *id* can later be used with the other co-processing functions rather than *command*. This option is typically used, since it facilitates the creation of two or more co-processes generated from the same *command*. (For example, cocreate -i ID1 program args and cocreate -i ID2 program different_args.)

-R *refname* — If -R is specified, *refname* is a local name for the co-process. Since the cocreate function can be issued more than once, a *refname* is useful when the same co-process is referenced a second or subsequent time. With the -R option, if the co-process already exists a new one will not be created: the same pipes will be shared. Then, *refname* can be used as an argument to the -R option to codestroy when you want to end a particular connection to a co-process and leave other connections undisturbed. (The co-process is only killed after codestroy -R has been called as many times as cocreate -R was called.)

`-s` *send_string* The `-s` option specifies *send_string* as a string that will be appended to all output sent to the co-process using `cosend`. This option allows a co-process to know when input from FMLI has completed. The default *send_string* is a newline if `-s` is not specified.

`-e` *expect_string* The `-e` option specifies *expect_string* as a string that identifies the end of all output returned by the co-process. (Note: *expect_string* need only be the initial part of a line, and there must be a newline at the end of the co-process output). This option allows FMLI to know when output from the co-process has completed. The default *expect_string* is a newline if `-e` is not specified.

The `cosend` function sends *string* to the co-process identified by *proc_id* via the pipe set up by `cocreate` (optionally *wpath*), where *proc_id* can be either the *command* or *id* specified in `cocreate`. By default, `cosend` blocks, waiting for a response from the co-process. Also by default, FMLI does not send a *send_string* and does not expect an *expect_string* (except a newline). That is, it reads only one line of output from the co-process. If `-e` *expect_string* was not defined when the pipe was created, then the output of the co-process is any single string followed by a newline: any other lines of output remain on the pipe. If the `-e` option was specified when the pipe was created, `cosend` reads lines from the pipe until it reads a line starting with *expect_string*. All lines except the line starting with *expect_string* become the output of `cosend`. The following option can be used with `cosend`:

`-n` If the `-n` option is specified, `cosend` will not wait for a response from the co-process. It simply returns, providing no output. If the `-n` option is not used, a co-process that does not answer will cause FMLI to permanently hang, waiting for input from the co-process.

The `cocheck` function determines if input is available from the process identified by *proc_id*, where *proc_id* can be either the *command* or *id* specified in `cocreate`. It returns a Boolean value, which makes `cocheck` useful in `if` statements and in other backquoted expressions in Boolean descriptors. `cocheck` receives no input from the co-process; it simply indicates if input is available from the co-process. You must use `coreceive` to actually accept the input. The `cocheck` function can be called from a `reread` descriptor to force a frame to update when new data is available. This is useful when the default value of a field in a form includes `coreceive`.

The `coreceive` function is used to read input from the co-process identified by *proc_id*, where *proc_id* can be either the *command* or *id* specified in `cocreate`. It should only be used when it has been determined, using `cocheck`, that input is actually available. If the `-e` option was used when the co-process was created, `coreceive` will continue to return lines of input until *expect_string* is read. At this point, `coreceive` will terminate. The output of `coreceive` is all the lines that were read excluding the line starting with *expect_string*. If the `-e` option was not used in the `cocreate`, each invocation of `coreceive` will return exactly one line from the co-process. If no input is available when `coreceive` is invoked, it will simply terminate without producing output.

The `codestroy` function terminates the read/write pipes to *proc-id*, where *proc_id* can be either the *command* or *id* specified in `cocreate`. It generates a SIGPIPE signal to the (child) co-process. This kills the co-process, unless the co-process ignores the SIGPIPE signal. If the co-process ignores the SIGPIPE, it will not die, even after

the FMLI process terminates (the parent process id of the co-process will be `1`).

The optional argument *string* is sent to the co-process before the co-process dies. If *string* is not supplied, a NULL string is passed, followed by the normal *send_string* (newline by default). That is, `codestroy` will call `cosend` *proc_id string*: this implies that `codestroy` will write any output generated by the co-process to *stdout*. For example, if an interactive co-process is written to expect a "quit" string when the communication is over, the `close` descriptor could be defined;

```
close=`codestroy ID 'quit' | message`
```

and any output generated by the co-process when the string `quit` is sent to it via `codestroy` (using `cosend`) would be redirected to the message line.

The `codestroy` function should usually be given the `-R` option, since you may have more than one process with the same name, and you do not want to kill the wrong one. `codestroy` keeps track of the number of *refnames* you have assigned to a process with `cocreate`, and when the last instance is killed, it kills the process (*id*) for you. `codestroy` is typically called as part of a `close` descriptor because `close` is evaluated when a frame is closed. This is important because the co-process will continue to run if `codestroy` is not issued.

When writing programs to use as co-processes, the following tips may be useful. If the co-process program is written in C language, be sure to flush output after writing to the pipe. (Currently, `awk`(1) and `sed`(1) cannot be used in a co-process program because they do not flush after lines of output.) Shell scripts are well-mannered, but slow. C language is recommended. If possible, use the default *send_string*, *rpath* and *wpath*. In most cases, *expect_string* will have to be specified. This, of course, depends on the co-process.

In the case where asynchronous communication from a co-process is desired, a co-process program should use `vsig` to force strings into the pipe and then signal FMLI that output from the co-process is available. This causes the `reread` descriptor of all frames to be evaluated immediately.

EXAMPLE

```
  .
  .
  .
init=`cocreate -i BIGPROCESS initialize`
close=`codestroy BIGPROCESS`
  .
  .
  .
reread=`cocheck BIGPROCESS`
name=`cosend -n BIGPROCESS field1`
  .
  .
  .
name="Receive field"
inactive=TRUE
value=`coreceive BIGPROCESS`
```

NOTES

Co-processes for trusted FMLI applications should use named pipes created by the application with the appropriate permissions; the default pipes created by FMLI are readable and writable by everyone. Handshaking can also be used to enhance security.

If `cosend` is used without the `-n` option, a co-process that does not answer will cause FMLI to permanently hang.

The use of non-alphabetic characters in input and output strings to a co-process should be avoided because they may not get transferred correctly.

SEE ALSO

`vsig`(1F)
`awk`(1), `cat`(1), `sed`(1).

NAME

copy - copy groups of files

SYNOPSIS

copy [*option*] . . . *source* . . . *dest*

DESCRIPTION

The copy command copies the contents of directories to another directory. It is possible to copy whole file systems since directories are made when needed.

If files, directories, or special files do not exist at the destination, then they are created with the same modes and flags as the source. In addition, the super-user may set the user and group ID. The owner and mode are not changed if the destination file exists. Note that there may be more than one source directory. If so, the effect is the same as if the copy command had been issued for each source directory with the same destination directory for each copy.

All of the options must be given as separate arguments, and they may appear in any order even after the other arguments. The arguments are:

-a Asks the user before attempting a copy. If the response does not begin with a "y", then a copy is not done. This option also sets the ad option.

-l Uses links instead whenever they can be used. Otherwise a copy is done. Note that links are never done for special files or directories.

-n Requires the destination file to be new. If not, then the copy command does not change the destination file. The -n flag is meaningless for directories. For special files an -n flag is assumed (that is, the destination of a special file must not exist).

-o If set then every file copied has its owner and group set to those of source. If not set, then the file's owner is the user who invoked the program.

-m If set, then every file copied has its modification time and access time set to that of the source. If not set, then the modification time is set to the time of the copy.

-r If set, then every directory is recursively examined as it is encountered. If not set, then any directories that are found are ignored.

-ad Asks the user whether an -r flag applies when a directory is discovered. If the answer does not begin with a "y," then the directory is ignored.

-v If the verbose option is set, messages are printed that reveal what the program is doing.

source This may be a file, directory or special file. It must exist. If it is not a directory, then the results of the command are the same as for the cp command.

dest The destination must be either a file or directory that is different from the source. If *source* and *destination* are anything but directories, then copy acts just like a cp command. If both are directories, then copy copies each file into the destination directory according to the flags that have been set.

NOTES

Special device files can be copied. When they are copied, any data associated with the specified device is not copied.

NAME

cp - copy files

SYNOPSIS

cp [-i] [-p] [-r] *file1* [*file2* . . .] *target*

DESCRIPTION

The cp command copies *filen* to *target*. *filen* and *target* may not have the same name. (Care must be taken when using sh(1) metacharacters.) If *target* is not a directory, only one file may be specified before it; if it is a directory, more than one file may be specified. If *target* does not exist, cp creates a file named *target*. If *target* exists and is not a directory, its contents are overwritten. If *target* is a directory, the file(s) are copied to that directory.

The following options are recognized:

-i cp will prompt for confirmation whenever the copy would overwrite an existing *target*. A y answer means that the copy should proceed. Any other answer prevents cp from overwriting *target*.

-p cp will duplicate not only the contents of *filen*, but also preserves the modification time and permission modes.

-r If *filen* is a directory, cp will copy the directory and all its files, including any subdirectories and their files; *target* must be a directory.

If *filen* is a directory, *target* must be a directory in the same physical file system. *target* and *filen* do not have to share the same parent directory.

If *filen* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

If *target* does not exist, cp creates a new file named *target* which has the same mode as *filen* except that the sticky bit is not set unless the user is a privileged user; the owner and group of *target* are those of the user.

If *target* is a file, its contents are overwritten, but the mode, owner, and group associated with it are not changed. The last modification time of *target* and the last access time of *filen* are set to the time the copy was made.

If *target* is a directory, then for each file named, a new file with the same mode is created in the target directory; the owner and the group are those of the user making the copy.

NOTES

A -- permits the user to mark the end of any command line options explicitly, thus allowing cp to recognize filename arguments that begin with a -. If a -- and a - both appear on the same command line, the second will be interpreted as a filename.

SEE ALSO

chmod(1), cpio(1), ln(1), mv(1), rm(1)

NAME

cpio - copy file archives in and out

SYNOPSIS

cpio -i[bBcdfkmrsStuvV6][-C *size*][-E *file*][-H *hdr*][-I *file* [-M *message*]][-R *ID*]][*pattern ...*]

cpio -o[aABcLvV][-C *size*][-H *hdr*][-O *file* [-M *message*]]

cpio -p[adlLmuvV][-R *ID*]] *directory*

DESCRIPTION

The -i, -o, and -p options select the action to be performed. The following list describes each of the actions (which are mutually exclusive).

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* are selected. *patterns* are regular expressions given in the filename-generating notation of sh(1). In *patterns*, meta-characters ?, *, and [...] match the slash (/) character, and backslash (\) is an escape character. A ! meta-character means *not*. (For example, the !abc* pattern would exclude all files that begin with abc.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). Each pattern must be enclosed in double quotes; otherwise, the name of a file in the current directory might be used. Extracted files are conditionally created and copied into the current directory tree based on the options described below. The permissions of the files will be those of the previous cpio -o. Owner and group permissions will be the same as the current user unless the current user is super-user. If this is true, owner and group permissions will be the same as those resulting from the previous cpio -o. NOTE: If cpio -i tries to create a file that already exists and the existing file is the same age or younger (**newer**), cpio will output a warning message and not replace the file. (The -u option can be used to overwrite, unconditionally, the existing file.)

cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary by default or to the user specified block size (with the -B or -C options) or to some device-dependent block size where necessary (as with the CTC tape).

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 on the options described below.

The meanings of the available options are

-a Reset access times of input files after they have been copied. Access times are not reset for linked files when cpio -pla is specified (mutually exclusive with -m).

-A Append files to an archive. The -A option requires the -O option. Valid only with archives that are files, or that are on floppy diskettes or hard disk slices.

-b Reverse the order of the bytes within each word. (Use only with the -i option.)

-B Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this and the -C options are not used. (-B does not apply to the *pass* option; -B is meaningful only with data directed to or from a character special device, e.g. /dev/rmt/ctape1.)

-c Read or write header information in ASCII character form. The -c option implies expanded device numbers. This option is mutually exclusive with -H and -6.

-C *bufsize*
Input/output is to be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this and -B options are not used. (-C does not apply to the *pass* option; -C is meaningful only with data directed to or from a character special device, e.g. /dev/rmt/ctape1.)

-d Directories are to be created as needed.

-E *file* Specify an input file (*file*) that contains a list of filenames to be extracted from the archive (one filename per line).

-f Copy in all files except those in *patterns.* (See the paragraph on cpio -i for a description of *patterns*.)

-H *hdr* Read or write header information in *hdr* format. This option should be used when the origin and destination machines are different types. (mutually exclusive with -c and -6). Valid values for *hdr* are:

crc or CRC - ASCII header with expanded device numbers and an additional per-file checksum

ustar or USTAR - IEEE/P1003.1 Data Interchange Standard tar header and format

tar or TAR - tar header and format

odc - ASCII header with small device numbers, IEEE/P1003.1 Data Interchange Standard cpio header and format.

See the NOTES section for additional information.

-I *file* Read the contents of *file* as an input archive. If *file* is a character special device, and the current medium has been completely read, replace the medium and press RETURN to continue to the next medium. This option is used only with the -i option.

-k Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For cpio archives that contain other cpio archives, if an error is encountered cpio may terminate prematurely. cpio will find the next good header, which may be one for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the -i option.

-l Whenever possible, link files rather than copying them. (Usable only with the -p option.)

-L Follow symbolic links. The default is not to follow symbolic links.

-m Retain previous file modification time. This option is ineffective on directories that are being copied (mutually exclusive with -a).

-M *message*
Define a *message* to use when switching media. When you use the -O or -I options and specify a character special device, you can use this option to define the message that is printed when you reach the end of the medium. One %d can be placed in *message* to print the sequence number of the next medium needed to continue.

-O *file* Direct the output of cpio to *file*. If *file* is a character special device and the current medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the -o option.

-r Interactively rename files. If the user types a carriage return alone, the file is skipped. If the user types a "." the original pathname will be retained. (Not available with cpio -p.)

-R *ID* Reassign ownership and group information for each file to *user ID* (*ID* must be a valid login ID from /etc/passwd). This option is valid only for the super-user.

-s Swap bytes within each half word.

-S Swap halfwords within each word.

-t Print a table of contents of the input. No files are created (mutually exclusive with -V).

-u Copy unconditionally (normally, an older file will not replace a newer file with the same name).

-v Verbose: causes 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)].

-V Special Verbose: print a dot for each file read or written. Useful to assure the user that cpio is working without printing out all file names.

-6 Process a UNIX System Sixth Edition archive format file. Use only with the -i option (mutually exclusive with -c and -H)).

NOTE: cpio assumes four-byte words.

If, when writing to a character device (-o) or reading from a character device (-i), cpio reaches the end of a medium (such as the end of a diskette), and the -O and -I options aren't used, cpio will print the following message:

```
If you want to go on, type device/file name when ready.
```

To continue, you must replace the medium and type the character special device name (/dev/rmt/ctape1 for example) and press RETURN. You may want to continue by directing cpio to use a different device. For example, if you have two floppy drives you may want to switch between them so cpio can proceed while you are changing the floppies. (Simply pressing RETURN causes the cpio process

to exit.)

EXAMPLES

The following examples show three uses of cpio.

When standard input is directed through a pipe to cpio -o, it groups the files so they can be directed (>) to a single file (../newfile). The -c option insures that the file will be portable to other machines (as would the -H option). Instead of ls(1), you could use find(1), echo(1), cat(1), and so on, to pipe a list of names to cpio. You could direct the output to a device instead of a file.

```
ls | cpio -oc > ../newfile
```

cpio -i uses the output file of cpio -o (directed through a pipe with cat in the example below), extracts those files that match the patterns (memo/a1, memo/b*), creates directories below the current directory as needed (-d option), and places the files in the appropriate directories. The -c option is used if the input file was created with a portable header. If no patterns were given, all files from newfile would be placed in the directory.

```
cat newfile | cpio -icd "memo/a1" "memo/b*"
```

cpio -p takes the file names piped to it and copies or links (-l option) those files to another directory (newdir in the example below). The -d option says to create directories as needed. The -m option says retain the modification time. (It is important to use the -depth option of find(1) to generate path names for cpio. This eliminates problems cpio could have trying to create files under read-only directories.) The destination directory, newdir, must exist.

```
find . -depth -print | cpio -pdlmv newdir
```

Note that when you use cpio in conjunction with find, if you use the -L option with cpio then you must use the -follow option with find and vice versa. Otherwise there will be undesirable results.

SEE ALSO

ar(1), cat(1), echo(1), find(1), ls(1), tar(1), archives(4).

NOTES

An archive created with the -c option on a Release 4 system cannot be read on System V Release 3.2 systems, or earlier. The -H odc header in Release 4 is equivalent to the -c header in earlier System V Releases.

The following table illustrates important capabilities of the supported archive formats. In the table, support for expanded device types indicates that the format can accommodate minor numbers greater than 255 and major numbers greater than 127.

Option	Pathname length (in bytes)	Supports Expanded device types	Readable on SVR3.2
(default)	256	No	Yes
-c	1024	Yes	No
-Hodc	256	No	Yes
-Hcrc	1024	Yes	No
-Htar	256	Yes	tar-Yes, cpio-No
-Hustar	256	Yes	tar-Yes, cpio-No

Only the super-user can copy special files.

Blocks are reported in 512-byte quantities.

If a file has 000 permissions, contains more than 0 characters of data, and the user is not root, the file will not be saved or restored.

INTERNATIONAL FUNCTIONS

cpio can process files containing characters from supplementary code sets. In pattern processing using metacharacters, matching is performed on characters, not bytes.

message with the -M option can include characters from supplementary code sets.

NAME

crash - examine system images

SYNOPSIS

/usr/sbin/crash [-d *dumpfile*] [-n *namelist*] [-w]

DESCRIPTION

The crash command is used to examine the system memory image of a running or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to crash are *dumpfile, namelist*, and *outputfile.*

dumpfile is the file containing the system memory image. The default *dumpfile* is /dev/mem.

The text file *namelist* contains the symbol table information needed for symbolic access to the system memory image to be examined. The default *namelist* is /stand/unix. If a system image from another machine is to be examined, the corresponding text file must be copied from that machine.

When the crash command is invoked, a session is initiated. The output from a crash session is directed to *outputfile.* The default *outputfile* is the standard output.

Input during a crash session is of the form:

function [*argument*...]

where *function* is one of the crash functions described in the "FUNCTIONS" subsection of this manual page, and *arguments* are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system or the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to crash functions wherever they are semantically valid.

-e	Display every entry in a table.
-f	Display the full structure.
-p	Interpret all address arguments in the command line as physical addresses. If they are not physical addresses, results are inconsistent.
-s *process*	Specify a process slot other than the default.
-w *file*	Redirect the output of a function to *file.*

The functions mode, defproc, and redirect correspond to the function options -p, -s, and -w. The mode function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; defproc sets the value of the process slot argument for subsequent functions; and redirect redirects all subsequent output.

Output from crash functions may be piped to another program in the following way:

function [*argument*...] ! *shell_command*

For example,

```
mount ! grep rw
```

writes all mount table entries with an `rw` flag to the standard output. The redirection option (`-w`) cannot be used with this feature.

Depending on the context of the function, numeric arguments are assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table address arguments larger than the size of the function table are interpreted as hexadecimal addresses; those smaller are assumed to be decimal slots in the table. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal is interpreted as hexadecimal if it is preceded by `0x` and as octal if it is preceded by `0`. Decimal override is designated by `0d`, and binary by `0b`.

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as `p` for `proc`, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information accept a table entry number, a physical address, a virtual address, a symbol, a range, or an expression. A range of slot numbers may be specified in the form *a-b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, -, *, /, &, or | . An operand that is a number should be preceded by a radix prefix if it is not a decimal number (`0` for octal, `0x` for hexadecimal, `0b` for binary). The expression must be enclosed in parentheses. Other functions accept any of these argument forms that are meaningful.

Two abbreviated arguments to `crash` functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = table entry | address | symbol | range | expression

start_addr = address | symbol | expression

FUNCTIONS

`?` [`-w`*file*]
List available functions.

`!`*command*
Escape to the shell and execute *command*.

`as` [`-e`] [`-f`] [`-w`*file*] [*proc*...]
Print information on process segments.

`base` [`-w`*file*] *number*...
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other than decimal should be preceded by a prefix that indicates its radix as follows: `0x`, hexadecimal; `0`, octal; and `0b`, binary.

`buffer` [`-w`*file*] [*-format*] *bufferslot*

`buffer` [`-w`*file*] [-format] [`-p`] *start_addr*
Alias: `b`.
Print the contents of a buffer in the designated format. The following format designations are recognized: `-b`, byte: `-c`, character; `-d`, decimal; `-x`,

hexadecimal; `-o`, octal; and, `-i`, inode. If no format is given, the previous format is used. The default format at the beginning of a `crash` session is hexadecimal.

`bufhdr` [`-f`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Alias: `buf`.
Print system buffer headers. The `-f` option produces different output depending on whether the buffer is local or remote (contains RFS data).

`callout` [`-w` *file*]
Alias: `c`.
Print the callout table.

`class` [`-w` *file*] [*table_entry* . . .]
Print information about process scheduler classes.

`dbfree` [`-w` *file*] [*class* . . .]
Print free streams data block headers. If a class is entered, only data block headers for the class specified is printed.

`dblock` [`-e`] [`-w` *file*] [`-c` *class* . . .]

`dblock` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Print allocated streams data block headers. If the class option (`-c`) is used, only data block headers for the class specified is printed.

`defproc` [`-w` *file*] [`-c`]

`defproc` [`-w` *file*] [*slot*]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (`-c`) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a `crash` session, the process slot is set to the current process.

`dis` [`-w` *file*] [`-a`] *start_addr* [*count*]

`dis` [`-w` *file*] [`-a`] `-c` [*count*]
Disassemble *count* instructions starting at *start_addr*. The default count is 1. The absolute option (`-a`) specifies a non-symbolic disassembly. The `-c` option can be used in place of *start_addr* to continue disassembly at the address at which a previous disassembly ended.

`dispq` [`-w` *file*] [*table_entry* . . .]
Print the dispatcher (scheduler) queues.

`ds` [`-w` *file*] *virtual_address* . . .
Print the data symbol whose address is closest to, but not greater than, the address entered.

`file` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Alias: `f`.
Print the file table.

`findaddr` [`-w` *file*] *table slot*
Print the address of *slot* in *table*. Only tables available to the `size` function are available to `findaddr`.

`findslot` [`-w` *file*] *virtual_address* . . .
Print the table, entry slot number, and offset for the address entered. Only tables available to the `size` function are available to `findslot`.

`fs` [`-w` *file*] [[`-p`] *table_entry* . . .]
Print the file system information table.

`gdp` [`-e`] [`-f`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Print the gift descriptor protocol table.

`help` [`-w` *file*] *function* . . .
Print a description of the named function, including syntax and aliases.

`inode` [`-e`] [`-f`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Alias: `i`.
Print the inode table, including file system switch information.

`kfp` [`-w` *file*] [`-s` *process*] [`-r`]

`kfp` [`-w` *file*] [`-s` *process*] [*value*]
Print the kernel frame pointer (kfp) for the start of a kernel stack trace. The kfp value can be set using the value argument or the reset option (`-r`), which sets the kfp from the saved kfp in the `dumpfile`. If no argument is entered, the current value of the kfp is printed.

`kmastat` [`-w` *file*]
Print kernel memory allocator statistics.

`lck` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Alias: `l`.
Print record locking information. If the `-e` option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

`linkblk` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Print the linkblk table.

`major` [`-w` *file*] [*entry* . . .]
Print the MAJOR table.

`map` [`-w` *file*] *mapname* . . .
Print the map structure of the given mapname.

`mbfree` [`-w` *file*]
Print free streams message block headers.

`mblock` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Print allocated streams message block headers.

`mode` [`-w` *file*] [*mode*]
Set address translation of arguments to virtual (`v`) or physical (`p`) mode. If no mode argument is given, the current mode is printed. At the start of a `crash` session, the mode is virtual.

`mount` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Alias: `m`, `vfs`.
Print information about mounted file systems.

`nm` [`-w` *file*] *symbol* . . .
Print value and type for the given symbol.

`nvram` [`-w` *file*] `user` | `0` | `net` | `1` | `os` | `3` | `bug` | `5` | `config` | `6` | `all` | `7`
Print out the contents of non-volatile RAM. Using a numerical section identifier causes the values to be printed in hexadecimal byte and ascii formats. Using the section name causes the values to be printed in formatted form if a specific format is defined for that section of non-volatile RAM and in hexadecimal byte and ascii formats if no specific format has been defined. Currently, only the `config` section has a specific format. The `nvram` command may not be available on all systems since some systems may not have any non-volatile RAM.

`od` [`-p`] [`-w` *file*] [*-format*] [*-mode*] [`-s` *process*] *start_addr* [*count*]
Alias: `rd`.
Print *count* values starting at *start_addr* in one of the following formats: character (`-c`), decimal (`-d`), hexadecimal (`-x`), octal (`-o`), ASCII (`-a`), or hexadecimal/character (`-h`), and one of the following modes: long (`-l`), short (`-t`), or byte (`-b`). The default mode for character and ASCII formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format `-h` prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a `crash` session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

`page` [`-e`] [`-w`*file*] [[`-p`] *table_entry* . . .]
Print information about pages.

`pcb` [`-w` *file*] [*process*]
Print the process control block. If no arguments are given, the active `pcb` for the current process is printed. This applies to M88000 family of processors.

`prnode` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Print information about the private data of processes being traced.

`proc` [`-e`] [`-f`] [`-w` *file*] [[`-p`] *table_entry* . . . `#`*procid* . . .]

`proc` [`-f`] [`-w` *file*] [`-r`]
Alias: `p`.
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process IDs may be entered. Each process ID must be preceded by a `#`. Alternatively, process table information for runnable processes may be specified with the runnable option (`-r`).

`ptbl` [`-w` *file*] [`-s`*process*] *ste* [*count*]

`ptbl` [`-w` *file*] [`-s`*process*] [`-p`] *addr* [*count*]
Print information on page descriptor tables, where *ste* is the segment table entry.

`pty` [`-f`] [`-e`] [`-w` *file*] [`-s`] [`-h`] [`-l`]
Print the pseudo ttys presently configured. The `-l`, `-s` and `-h` options give information about the STREAMS modules `ldterm`, `ptem` and `pckt`, respectively.

`qrun` [`-w`*file*]
Print the list of scheduled streams queues.

`queue` [`-e`] [`-w`*file*] [[`-p`] *table_entry* . . .]
Print streams queues.

`quit` Alias: `q`.
Terminate the `crash` session.

`rcvd` [`-e`] [`-f`] [`-w`*file*] [[`-p`] *table_entry* . . .]
Print the receive descriptor table.

`rduser` [`-e`] [`-f`] [`-w`*file*] [[`-p`] *table_entry* . . .]
Print the receive descriptor user table.

`regs` [`-w`*file*] [`-g`] [`-s`] [`-m`] [`-f`] [`-a`]
Print Motorola M68000 or M88000 family of processors and co-processor registers including the general, supervisor, mmu, and floating point registers. The (`-a`) option prints all of the register groups.

`redirect` [`-w`*file*] [`-c`]

`redirect` [`-w`*file*] [*newfile*]
Used with a file name, redirects output of a `crash` session to *newfile*. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (`-c`) closes the previously set file and redirects output to the standard output.

`resource` [`-e`] [`-w`*file*] [[`-p`] *table_entry* . . .]
Print the advertise table.

`rtdptbl` [`-w`*file*] [*table_entry* . . .]
Print the real-time scheduler parameter table. See `rt_dptbl`(4).

`rtproc` [`-w`*file*]
Print information about processes in the real-time scheduler class.

`sdt` [`-e`] [`-w`*file*] [`-s` *process*] [`-p` *start_addr*] [*count*]
The segment descriptor table for the named memory section is printed. Alternatively, the segment descriptor table starting at *start_addr* for *count* entries is printed. If no count is given, a count of 1 is assumed.

`search` [`-p`] [`-w`*file*] [`-m` *mask*] [`-s` *process*] *pattern start_addr length*
Print the words in memory that match *pattern*, beginning at the *start_addr* for *length* words. The mask is ANDed (&) with each memory word and the result compared against the pattern. The mask defaults to `0xffffffff`.

`size` [`-w`*file*] [`-x`] [*structure_name* . . .]
Print the size of the designated structure. The (`-x`) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

`sndd` [`-e`] [`-f`] [`-w`*file*] [[`-p`] *table_entry* . . .]
Print the send descriptor table.

`snode` [`-e`] [`-f`] [`-w`*file*] [[`-p`] *table_entry* . . .]
Print information about open special files.

`srmount` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Print the server mount table.

`stat` [`-w` *file*]
Print out the system status. This consists of general information about the system itself, information about when the system crashed (or the current time if used on a running system) and how long the system was running before a crash (or has been running).

`stack` [`-w` *file*] [`-u`] [*process*]

`stack` [`-w` *file*] [`-k`] [*process*]

`stack` [`-w` *file*] [[`-p`] `-i` *start_addr*]
Alias: `s`.
Dump the stack. The (`-u`) option prints the user stack. The (`-k`) option prints the kernel stack. The (`-i`) option prints the interrupt stack. If no arguments are entered, the kernel stack for the current process is printed. The interrupt stack and the stack for the current process are not available on a running system.

`stream` [`-e`] [`-f`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Print the streams table.

`strstat` [`-w` *file*]
Print streams statistics.

`trace` [`-w` *file*] [`-r`] [*process*]

`trace` [`-w` *file*] [[`-p`] `-i` *start_addr*]
Alias: `t`.
Print stack trace. The kfp value is used with the `-r` option; the `kfp` function prints or sets the kfp (kernel frame pointer) value. The interrupt option prints a trace of the interrupt stack beginning at *start_addr*. The interrupt stack trace and the stack trace for the current process are not available on a running system.

`ts` [`-w` *file*] *virtual_address* . . .
Print text symbol closest to the designated address.

`tsdptbl` [`-w` *file*] [*table_entry* . . .]
Print the time-sharing scheduler parameter table. See `ts_dptbl`(4).

`tsproc` [`-w` *file*]
Print information about processes in the time-sharing scheduler class.

`tty` [`-e`] [`-f`] [`-l`] [`-w` *file*] [`-t` *type* [[`-p`] *table_entry* . . .]]

`tty` [`-e`] [`-f`] [`-l`] [`-w` *file*] [[`-p`] *start_addr*]
Valid types: `iu`.
Print the tty table. If no arguments are given, the tty table for the console tty type is printed. If the `-t` option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified using *start_addr*. The `-l` option prints the line discipline information.

`uinode` [`-e`] [`-f`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Alias: `ui`.
Print the ufs inode table.

`user` [`-f`] [`-w` *file*] [*process*]
Alias: `u`.
Print the ublock for the designated process.

`var` [`-w` *file*]
Alias: `v`.
Print the tunable system parameters.

`vfs` [`-e`] [`-w` *file*] [[`-p`] *table_entry* . . .]
Alias: `mount`, `m`.
Print information about mounted file systems.

`vfssw` [`-w` *file*] [[`-p`] *table_entry* . . .]
Print information about configured file system types.

`vnode` [`-w` *file*] [[`-p`] *vnode_addr* . . .]
Print information about vnodes.

`vtop` [`-w` *file*] [`-s` process] *start_addr* . . .
Print the physical address translation of the virtual address *start_addr*.

FILES

`/dev/mem`	system image of currently running system
`/dev/rmt/ctape1`	used to access system image on cartridge tape
`/dev/swap`	used to access system image in swap slice

SEE ALSO

`ldsysdump`(1M)

NAME

crashconf - enable/disable crash dumps

SYNOPSIS

/usr/sbin/crashconf [-a] *special_device*

DESCRIPTION

The crashconf command with arguments configures the crash dump system to take a crash dump to the *special_device* if the operating system should panic. The *special_device* must follow the conventions for device specific files as given in intro(7). If a disk slice, it must also be tagged with V_SWAP.

If the -a option is given, the crash dump system will be in automatic mode and will not prompt the user before beginning the crash dump when the system panics; otherwise, the crash dump system will be in manual mode and prompts the user to prepare the crash dump device prior to beginning the crash dump. Manual mode also allows the user to attempt to recover if a failure occurs if the device supports recovery.

The crashconf command with no arguments disables the crash dump system and no crash dump will be taken.

The buildsys(1M) manpage contains information regarding preservation of the bootable unix corresponding to a crash dump.

EXAMPLES

This example shows a disk slice being configured as the crash dump device with no user intervention requested.

```
$crashconf -a /dev/rdsk/m328_c0d0s1
$
```

FILES

/etc/init.d/CRASHDUMP	script to configure crash dumps during boot
/dev/rdsk/prefix_c*n*d*n*s*n*	device used for disk slice
/dev/rmt/prefix_c*n*d*n*	device used for tapes

SEE ALSO

buildsys(1M), crash(1M), fmthard(1M), ldsysdump(1M), sysm68k(2), sysm88k(2).

DIAGNOSTICS

If the crash dump system can not be configured to the device *special_device* for some reason, crashconf will print explanatory messages and the exit value will be 1. The exit value is 0 upon success.

NOTES

The *special_device* must be large enough to accommodate the entire physical memory of the system or data may be lost.

If the *special_device* also serves as a swap slice, it may need to be significantly larger than the amount of physical memory to keep from corrupting the crash dump image after reboot.

NAME

cron - clock daemon

SYNOPSIS

/usr/sbin/cron

DESCRIPTION

The cron command starts a process that executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in crontab files in the directory /var/spool/cron/crontabs. Users can submit their own crontab file via the crontab command. Commands which are to be executed only once may be submitted via the at command.

cron only examines crontab files and at command files during process initialization and when a file changes via the crontab or at commands. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since cron never exits, it should be executed only once. This is done routinely through /sbin/rc2.d/S75cron at system boot time. /etc/cron.d/FIFO is used as a lock file to prevent the execution of more than one cron.

To keep a log of all actions taken by cron, CRONLOG=YES (by default) must be specified in the /etc/default/cron file. If CRONLOG=NO is specified, no logging is done. Keeping the log is a user configurable option since cron usually creates huge log files.

FILES

/usr/sbin/cron.d	main cron directory
/etc/default/cron	used to maintain a log
/etc/cron.d/FIFO	used as a lock file
/var/cron/log	accounting information
/var/spool/cron	spool area

SEE ALSO

at(1), crontab(1), sh(1).

DIAGNOSTICS

A history of all actions taken by cron are recorded in /var/cron/log.

NAME

`crontab` - user crontab file

SYNOPSIS

`crontab` [*file*]
`crontab -e` [username]
`crontab -r` [username]
`crontab -l` [username]

DESCRIPTION

`crontab` copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The `-e` option edits a copy of the current user's `crontab` file, or creates an empty file to edit if `crontab` does not exist. When editing is complete, the file is installed as the user's `crontab` file. If a *username* is given, the specified user's `crontab` file is edited, rather than the current user's `crontab` file; this may only be done by a privileged user. When invoked with the `-e` option, the editor used is determined by first checking the environment variable `VISUAL`, then `EDITOR`. If neither is set, it defaults to `ed`(1). The `-r` option removes a user's crontab from the crontab directory. `crontab -l` will list the crontab file for the invoking user. Only a privileged user can specify a *username* following the `-r` or `-l` options to remove or list the `crontab` file of the specified user.

Users are permitted to use `crontab` if their names appear in the file `/etc/cron.d/cron.allow`. If that file does not exist, the file `/etc/cron.d/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 `cron.allow` does not exist and `cron.deny` exists but 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.

Any line beginning with a # is a comment and will be ignored.

The shell is invoked from your $HOME directory with an arg0 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:/usr/lbin).

If you do not redirect the standard output and standard error of your commands, any generated output or errors will be mailed to you.

FILES

/usr/sbin/cron.d	main cron directory
/var/spool/cron/crontabs	spool area
/usr/sbin/cron.d/log	accounting information
/etc/cron.d/cron.allow	list of allowed users
/etc/cron.d/cron.deny	list of denied users

SEE ALSO

atq(1), atrm(1), sh(1), su(1), vi(1) cron(1M)

NOTES

If you inadvertently enter the crontab command with no argument(s), do not attempt to get out with a CTRL-D. This will cause all entries in your crontab file to be removed. Instead, exit with a DEL.

If a privileged user modifies another user's crontab file, resulting behavior may be unpredictable. Instead, the privileged user should first su(1M) to the other user's login before making any changes to the crontab file.

NAME

crypt - encode/decode

SYNOPSIS

crypt [*password*]
crypt [-k]

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 argument is given, crypt demands a key from the terminal and turns off printing while the key is being typed in. If the -k option is used, crypt will use the key assigned to the environment variable CRYPTKEY. crypt encrypts and decrypts with the same key:

```
crypt key <clear >cypher
crypt key <cypher | pr
```

Files encrypted by crypt are compatible with those treated by the editors ed(1), edit(1), ex(1), and vi(1) 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, that is, to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

If 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.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), edit(1), ex(1), makekey(1), nroff(1), pg(1), ps(1), stty(1), vi(1)

NOTES

This command is provided with the Encryption Utilities, which is only available in the United States. 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.

If output is piped to nroff and the encryption key is not given on the command line then do not pipe crypt through pg(1) or any other program that changes the tty settings. Doing so may cause crypt to leave terminal modes in a strange state [see stty(1)].

NAME

cscope - interactively examine a C program

SYNOPSIS

cscope [*options*] *files* . . .

DESCRIPTION

cscope is an interactive screen-oriented tool that allows the user to browse through C source files for specified elements of code.

By default, cscope examines the C (.c and .h), lex (.l), and yacc (.y) source files in the current directory. cscope may also be invoked for source files named on the command line. In either case, cscope searches the standard directories for #include files that it does not find in the current directory. cscope uses a symbol cross-reference, cscope.out by default, to locate functions, function calls, macros, variables, and preprocessor symbols in the files.

cscope builds the symbol cross-reference the first time it is used on the source files for the program being browsed. On a subsequent invocation, cscope rebuilds the cross-reference only if a source file has changed or the list of source files is different. When the cross-reference is rebuilt, the data for the unchanged files are copied from the old cross-reference, which makes rebuilding faster than the initial build.

The following options can appear in any combination:

-b	Build the cross-reference only.
-C	Ignore letter case when searching.
-c	Use only ASCII characters in the cross-reference file, that is, do not compress the data.
-d	Do not update the cross-reference.
-e	Suppress the ^e command prompt between files.
-f *reffile*	Use *reffile* as the cross-reference file name instead of the default cscope.out.
-I *incdir*	Look in *incdir* (before looking in *INCDIR*, the standard place for header files, normally /usr/include) for any #include files whose names do not begin with / and that are not specified on the command line or in *namefile* below. (The #include files may be specified with either double quotes or angle brackets.) The *incdir* directory is searched in addition to the current directory (which is searched first) and the standard list (which is searched last). If more than one occurrence of -I appears, the directories are searched in the order they appear on the command line.
-i *namefile*	Browse through all source files whose names are listed in *namefile* (file names separated by spaces, tabs, or new-lines) instead of the default (cscope.files). If this option is specified, cscope ignores any files appearing on the command line.
-L	Do a single search with line-oriented output when used with the *-num pattern* option.

`-l`	Line-oriented interface (see "Line-Oriented Interface" below).
-num pattern	Go to input field *num* (counting from 0) and find *pattern*.
`-P` *path*	Prepend *path* to relative file names in a pre-built cross-reference file so you do not have to change to the directory where the cross-reference file was built. This option is only valid with the `-d` option.
`-p` *n*	Display the last *n* file path components instead of the default (1). Use 0 to not display the file name at all.
`-s` *dir*	Look in *dir* for additional source files. This option is ignored if source files are given on the command line.
`-T`	Use only the first eight characters to match against C symbols. A regular expression containing special characters other than a period (`.`) will not match any symbol if its minimum length is greater than eight characters.
`-U`	Do not check file time stamps (assume that no files have changed).
`-u`	Unconditionally build the cross-reference file (assume that all files have changed).
`-V`	Print on the first line of screen the version number of `cscope`.

The `-I`, `-p`, and `-T` options can also be in the `cscope.files` file.

Requesting the Initial Search

After the cross-reference is ready, `cscope` will display this menu:

```
Find this C symbol:
Find this function definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
```

Press the `TAB` key repeatedly to move to the desired input field, type the text to search for, and then press the `RETURN` key.

Issuing Subsequent Requests

If the search is successful, any of these single-character commands can be used:

`1-9`	Edit the file referenced by the given line number.
`SPACE`	Display next set of matching lines.
`+`	Display next set of matching lines.
`–`	Display previous set of matching lines.
`^e`	Edit displayed files in order.
`>`	Append the displayed list of lines to a file.
`|`	Pipe all lines to a shell command.

At any time these single-character commands can also be used:

TAB	Move to next input field.
RETURN	Move to next input field.
^n	Move to next input field.
^p	Move to previous input field.
^y	Search with the last text typed.
^b	Move to previous input field and search pattern.
^f	Move to next input field and search pattern.
^c	Toggle ignore/use letter case when searching. (When ignoring letter case, search for FILE will match File and file.)
^r	Rebuild the cross-reference.
!	Start an interactive shell (type ^d to return to cscope).
^l	Redraw the screen.
?	Give help information about cscope commands.
^d	Exit cscope.

Note: If the first character of the text to be searched for matches one of the above commands, escape it by typing a \ (backslash) first.

Substituting New Text for Old Text

After the text to be changed has been typed, cscope will prompt for the new text, and then it will display the lines containing the old text. Select the lines to be changed with these single-character commands:

1-9	Mark or unmark the line to be changed.
*	Mark or unmark all displayed lines to be changed.
SPACE	Display next set of lines.
+	Display next set of lines.
–	Display previous set of lines.
a	Mark all lines to be changed.
^d	Change the marked lines and exit.
ESCAPE	Exit without changing the marked lines.
!	Start an interactive shell (type ^d to return to cscope).
^l	Redraw the screen.
?	Give help information about cscope commands.

Special Keys

If your terminal has arrow keys that work in vi(1), you can use them to move around the input fields. The up-arrow key is useful to move to the previous input field instead of using the TAB key repeatedly. If you have CLEAR, NEXT, or PREV keys they will act as the ^l, +, and - commands, respectively.

Line-Oriented Interface

The -l option lets you use cscope where a screen-oriented interface would not be useful, for example, from another screen-oriented program.

cscope will prompt with >> when it is ready for an input line starting with the field number (counting from 0) immediately followed by the search pattern, for example, lmain finds the definition of the main function.

If you just want a single search, instead of the -l option use the -L and *-num pattern* options, and you won't get the >> prompt.

For -l, cscope outputs the number of reference lines

```
cscope: 2 lines
```

For each reference found, cscope outputs a line consisting of the file name, function name, line number, and line text, separated by spaces, for example,

```
main.c main 161 main(argc, argv)
```

Note that the editor is not called to display a single reference, unlike the screen-oriented interface.

You can use the r command to rebuild the database.

cscope will quit when it detects end-of-file, or when the first character of an input line is ^d or q.

ENVIRONMENT VARIABLES

EDITOR	Preferred editor, which defaults to vi(1).
INCLUDEDIRS	Colon-separated list of directories to search for #include files.
HOME	Home directory, which is automatically set at login.
SHELL	Preferred shell, which defaults to sh(1).
SOURCEDIRS	Colon-separated list of directories to search for additional source files.
TERM	Terminal type, which must be a screen terminal.
TERMINFO	Terminal information directory full path name. If your terminal is not in the standard terminfo directory, see curses(3X) and terminfo(4) for how to make your own terminal description.
TMPDIR	Temporary file directory, which defaults to /var/tmp.
VIEWER	Preferred file display program [such as pg], which overrides EDITOR (see above).
VPATH	A colon-separated list of directories, each of which has the same directory structure below it. If VPATH is set, cscope searches for source files in the directories specified; if it is not set, cscope searches only in the current directory.

FILES

cscope.files	Default files containing -I, -p, and -T options and the list of source files (overridden by the -i option).
cscope.out	Symbol cross-reference file, which is put in the home directory if it cannot be created in the current directory.
ncscope.out	Temporary file containing new cross-reference before it replaces the old cross-reference.
INCDIR	Standard directory for #include files (usually /usr/include).

SEE ALSO

curses, terminfo.

NOTES

`cscope` recognizes function definitions of the form:

fname blank (*args*) *white arg_decs white* {

where:

fname	is the function name
blank	is zero or more spaces or tabs, not including newlines
args	is any string that does not contain a " or a newline
white	is zero or more spaces, tabs, or newlines
arg_decs	are zero or more argument declarations (*arg_decs* may include comments and white space)

It is not necessary for a function declaration to start at the beginning of a line. The return type may precede the function name; `cscope` will still recognize the declaration. Function definitions that deviate from this form will not be recognized by `cscope`.

The `Function` column of the search output for the menu option `Find functions called by this function:` input field will only display the first function called in the line, that is, for this function

```
e()
{
      return (f() + g());
}
```

the display would be

```
Functions called by this function: e

File Function Line
a.c  f     3 return(f() + g());
```

Occasionally, a function definition or call may not be recognized because of braces inside `#if` statements. Similarly, the use of a variable may be incorrectly recognized as a definition.

A `typedef` name preceding a preprocessor statement will be incorrectly recognized as a global definition, for example,

```
LDFILE *
#if AR16WR
```

Preprocessor statements can also prevent the recognition of a global definition, for example,

```
char flag
#ifdef ALLOCATE_STORAGE
      = -1
#endif
;
```

A function declaration inside a function is incorrectly recognized as a function call, for example,

```
f()
{
```

```
        void g();
}
```

is incorrectly recognized as a call to `g()`.

`cscope` recognizes C++ classes by looking for the class keyword, but doesn't recognize that a `struct` is also a class, so it doesn't recognize inline member function definitions in a structure. It also doesn't expect the class keyword in a `typedef`, so it incorrectly recognizes `X` as a definition in

```
typedef class X * Y;
```

It also doesn't recognize operator function definitions

```
Bool Feature::operator==(const Feature & other)
{
        ...
}
```

NAME

csh - shell command interpreter with a C-like syntax

SYNOPSIS

csh [-bcefinstvVxX] [*argument* . . .]

DESCRIPTION

csh, the C shell, is a command interpreter with a syntax reminiscent of the C language. It provides a number of convenient features for interactive use that are not available with the standard (Bourne) shell, including filename completion, command aliasing, history substitution, job control, and a number of built-in commands. As with the standard shell, the C shell provides variable, command and filename substitution.

Initialization and Termination

When first started, the C shell normally performs commands from the .cshrc file in your home directory, provided that it is readable and you either own it or your real group ID matches its group ID. If the shell is invoked with a name that starts with '-', as when started by login(1), the shell runs as a login shell. In this case, after executing commands from the .cshrc file, the shell executes commands from the .login file in your home directory; the same permission checks as those for .cshrc are applied to this file. Typically, the .login file contains commands to specify the terminal type and environment.

As a login shell terminates, it performs commands from the .logout file in your home directory; the same permission checks as those for .cshrc are applied to this file.

Interactive Operation

After startup processing is complete, an interactive C shell begins reading commands from the terminal, prompting with *hostname*% (or *hostname*# for the privileged user). The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the history list and then parsed, as described under USAGE, below. Finally, the shell executes each command in the current line.

Noninteractive Operation

When running noninteractively, the shell does not prompt for input from the terminal. A noninteractive C shell can execute a command supplied as an *argument* on its command line, or interpret commands from a script.

The following options are available:

-b Force a break from option processing. Subsequent command-line arguments are not interpreted as C shell options. This allows the passing of options to a script without confusion. The shell does not run a set-user-ID script unless this option is present.

-c Read commands from the first filename *argument* (which must be present). Remaining arguments are placed in argv, the argument-list variable.

-e Exit if a command terminates abnormally or yields a nonzero exit status.

-f Fast start. Read neither the .cshrc file, nor the .login file (if a login shell) upon startup.

`-i` Forced interactive. Prompt for command-line input, even if the standard input does not appear to be a terminal (character-special device).

`-n` Parse (interpret), but do not execute commands. This option can be used to check C shell scripts for syntax errors.

`-s` Take commands from the standard input.

`-t` Read and execute a single command line. A '\' (backslash) can be used to escape each newline for continuation of the command line onto subsequent input lines.

`-v` Verbose. Set the `verbose` predefined variable; command input is echoed after history substitution (but before other substitutions) and before execution.

`-V` Set `verbose` before reading `.cshrc`.

`-x` Echo. Set the `echo` variable; echo commands after all substitutions and just before execution.

`-X` Set `echo` before reading `.cshrc`.

Except with the options `-c`, `-i`, `-s` or `-t`, the first nonoption *argument* is taken to be the name of a command or script. It is passed as argument zero, and subsequent arguments are added to the argument list for that command or script.

USAGE

Filename Completion

When enabled by setting the variable `filec`, an interactive C shell can complete a partially typed filename or user name. When an unambiguous partial filename is followed by an ESC character on the terminal input line, the shell fills in the remaining characters of a matching filename from the working directory.

If a partial filename is followed by the EOF character (usually typed as CTRL-d), the shell lists all filenames that match. It then prompts once again, supplying the incomplete command line typed in so far.

When the last (partial) word begins with a tilde (`~`), the shell attempts completion with a user name, rather than a file in the working directory.

The terminal bell signals errors or multiple matches; this can be inhibited by setting the variable `nobeep`. You can exclude files with certain suffixes by listing those suffixes in the variable `fignore`. If, however, the only possible completion includes a suffix in the list, it is not ignored. `fignore` does not affect the listing of filenames by the EOF character.

Lexical Structure

The shell splits input lines into words at space and tab characters, except as noted below. The characters `&`, `|`, `;`, `<`, `>`, `(`, and `)` form separate words; if paired, the pairs form single words. These shell metacharacters can be made part of other words, and their special meaning can be suppressed by preceding them with a '\' (backslash). A newline preceded by a \ is equivalent to a space character.

In addition, a string enclosed in matched pairs of single-quotes ('), double-quotes ("), or backquotes (` ), forms a partial word; metacharacters in such a string, including any space or tab characters, do not form separate words. Within pairs of backquote ( `) or double-quote (") characters, a newline preceded by a '\' (backslash) gives a true newline character. Additional functions of each type of

quote are described, below, under `Variable Substitution`, `Command Substitution`, and `Filename Substitution`.

When the shell's input is not a terminal, the character # introduces a comment that continues to the end of the input line. Its special meaning is suppressed when preceded by a \ or enclosed in matching quotes.

Command Line Parsing

A *simple command* is composed of a sequence of words. The first word (that is not part of an I/O redirection) specifies the command to be executed. A simple command, or a set of simple commands separated by | or |& characters, forms a *pipeline*. With |, the standard output of the preceding command is redirected to the standard input of the command that follows. With |&, both the standard error and the standard output are redirected through the pipeline.

Pipelines can be separated by semicolons (;), in which case they are executed sequentially. Pipelines that are separated by && or || form conditional sequences in which the execution of pipelines on the right depends upon the success or failure, respectively, of the pipeline on the left.

A pipeline or sequence can be enclosed within parentheses '( )' to form a simple command that can be a component in a pipeline or sequence.

A sequence of pipelines can be executed asynchronously, or in the background by appending an '&'; rather than waiting for the sequence to finish before issuing a prompt, the shell displays the job number (see `Job Control`, below) and associated process IDs, and prompts immediately.

History Substitution

History substitution allows you to use words from previous command lines in the command line you are typing. This simplifies spelling corrections and the repetition of complicated commands or arguments. Command lines are saved in the history list, the size of which is controlled by the `history` variable. The most recent command is retained in any case. A history substitution begins with a ! (although you can change this with the `histchars` variable) and may occur anywhere on the command line; history substitutions do not nest. The ! can be escaped with \ to suppress its special meaning.

Input lines containing history substitutions are echoed on the terminal after being expanded, but before any other substitutions take place or the command gets executed.

Event Designators

An event designator is a reference to a command-line entry in the history list.

!	Start a history substitution, except when followed by a space character, tab, newline, = or (.
!!	Refer to the previous command. By itself, this substitution repeats the previous command.
!*n*	Refer to command-line *n* .
!-*n*	Refer to the current command-line minus *n*.
!`str`	Refer to the most recent command starting with `str`.
!?*str*[?]	Refer to the most recent command containing `str`.

`!{...}` Insulate a history reference from adjacent characters (if necessary).

Word Designators

A ':' (colon) separates the event specification from the word designator. It can be omitted if the word designator begins with a `^`, `$`, `*`, `-` or `%`. If the word is to be selected from the previous command, the second `!` character can be omitted from the event specification. For instance, `!!:1` and `!:1` both refer to the first word of the previous command, while `!!$` and `!$` both refer to the last word in the previous command. Word designators include:

`#`	The entire command line typed so far.
`0`	The first input word (command).
n	The *n*'th argument.
`^`	The first argument, that is, `1`.
`$`	The last argument.
`%`	The word matched by (the most recent) `?`*s* search.
x-*y*	A range of words; -*y* abbreviates `0`-*y*.
`*`	All the arguments, or a null value if there is just one word in the event.
x`*`	Abbreviates *x*-`$`.
x-	Like *x*`*` but omitting word `$`.

Modifiers

After the optional word designator, you can add a sequence of one or more of the following modifiers, each preceded by a `:`.

`h`	Remove a trailing pathname component, leaving the head.
`r`	Remove a trailing suffix of the form '.*xxx*', leaving the basename.
`e`	Remove all but the suffix.
`s/`*l*`/`*r*`[/]`	Substitute *r* for *l*.
`t`	Remove all leading pathname components, leaving the tail.
`&`	Repeat the previous substitution.
`g`	Apply the change to the first occurrence of a match in each word, by prefixing the above (for example, `g&`).
`p`	Print the new command but do not execute it.
`q`	Quote the substituted words, escaping further substitutions.
`x`	Like `q`, but break into words at each space character, tab or newline.

Unless preceded by a `g`, the modification is applied only to the first string that matches *l*; an error results if no string matches.

The left-hand side of substitutions are not regular expressions, but character strings. Any character can be used as the delimiter in place of `/`. A backslash quotes the delimiter character. The character `&`, in the right hand side, is replaced by the text from the left-hand-side. The `&` can be quoted with a backslash. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* from `!?`*s*. You can omit the rightmost delimiter if a newline immediately follows *r*; the rightmost `?` in a context scan can similarly be omitted.

Without an event specification, a history reference refers either to the previous command, or to a previous history reference on the command line (if any).

Quick Substitution

^*l*^*r*[^] This is equivalent to the history substitution: `!:s^`*l*`^`*r*`[^]`.

Aliases

The C shell maintains a list of aliases that you can create, display, and modify using the `alias` and `unalias` commands. The shell checks the first word in each command to see if it matches the name of an existing alias. If it does, the command is reprocessed with the alias definition replacing its name; the history substitution mechanism is made available as though that command were the previous input line. This allows history substitutions, escaped with a backslash in the definition, to be replaced with actual command-line arguments when the alias is used. If no history substitution is called for, the arguments remain unchanged.

Aliases can be nested. That is, an alias definition can contain the name of another alias. Nested aliases are expanded before any history substitutions is applied. This is useful in pipelines such as

```
alias lm 'ls -l \!* | more'
```

which when called, pipes the output of `ls`(1V) through `more`(1).

Except for the first word, the name of the alias may not appear in its definition, nor in any alias referred to by its definition. Such loops are detected, and cause an error message.

I/O Redirection

The following metacharacters indicate that the subsequent word is the name of a file to which the command's standard input, standard output, or standard error is redirected; this word is variable, command, and filename expanded separately from the rest of the command.

<	Redirect the standard input.
<<*word*	Read the standard input, up to a line that is identical with *word*, and place the resulting lines in a temporary file. Unless *word* is escaped or quoted, variable and command substitutions are performed on these lines. Then, invoke the pipeline with the temporary file as its standard input. *word* is not subjected to variable, filename, or command substitution, and each line is compared to it before any substitutions are performed by the shell.

`> >! >& >&!` Redirect the standard output to a file. If the file does not exist, it is created. If it does exist, it is overwritten; its previous contents are lost.

When set, the variable `noclobber` prevents destruction of existing files. It also prevents redirection to terminals and `/dev/null`, unless one of the `!` forms is used. The `&` forms redirect both standard output and the the standard error (diagnostic output) to the file.

`>> >>& >>! >>&!`
Append the standard output. Like `>`, but places output at the end of the file rather than overwriting it. If `noclobber` is set, it is an error for the file not to exist, unless one of the `!` forms is used. The `&` forms append both the standard error and standard output to the file.

Variable Substitution

The C shell maintains a set of *variables*, each of which is composed of a *name* and a *value*. A variable name consists of up to 20 letters and digits, and starts with a letter (the underscore is considered a letter). A variable's value is a space-separated list of zero or more words.

To refer to a variable's value, precede its name with a '`$`'. Certain references (described below) can be used to select specific words from the value, or to display other information about the variable. Braces can be used to insulate the reference from other characters in an input-line word.

Variable substitution takes place after the input line is analyzed, aliases are resolved, and I/O redirections are applied. Exceptions to this are variable references in I/O redirections (substituted at the time the redirection is made), and backquoted strings (see Command Substitution).

Variable substitution can be suppressed by preceding the `$` with a `\`, except within double-quotes where it always occurs. Variable substitution is suppressed inside of single-quotes. A `$` is escaped if followed by a space character, tab or newline.

Variables can be created, displayed, or destroyed using the `set` and `unset` commands. Some variables are maintained or used by the shell. For instance, the `argv` variable contains an image of the shell's argument list. Of the variables used by the shell, a number are toggles; the shell does not care what their value is, only whether they are set or not.

Numerical values can be operated on as numbers (as with the `@` built-in). With numeric operations, an empty value is considered to be zero; the second and subsequent words of multiword values are ignored. For instance, when the `verbose` variable is set to any value (including an empty value), command input is echoed on the terminal.

Command and filename substitution is subsequently applied to the words that result from the variable substitution, except when suppressed by double-quotes, when `noglob` is set (suppressing filename substitution), or when the reference is quoted with the `:q` modifier. Within double-quotes, a reference is expanded to form (a portion of) a quoted string; multiword values are expanded to a string with embedded space characters. When the `:q` modifier is applied to the reference, it is expanded to a list of space-separated words, each of which is quoted to prevent

subsequent command or filename substitutions.

Except as noted below, it is an error to refer to a variable that is not set.

`$`*var*
`${`*var*`}` These are replaced by words from the value of *var*, each separated by a space character. If *var* is an environment variable, its value is returned (but `':'` modifiers and the other forms given below are not available).

`$`*var*`[`*index*`]`
`${`*var*`[`*index*`]}` These select only the indicated words from the value of *var*. Variable substitution is applied to *index*, which may consist of (or result in) a either single number, two numbers separated by a `'-'`, or an asterisk. Words are indexed starting from 1; a `'*'` selects all words. If the first number of a range is omitted (as with `$argv[-2]`), it defaults to 1. If the last number of a range is omitted (as with `$argv[1-]`), it defaults to `$#`*var* (the word count). It is not an error for a range to be empty if the second argument is omitted (or within range).

`$#`*name*
`${#`*name*`}` These give the number of words in the variable.

`$0` This substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

`$`*n*
`${`*n*`}` Equivalent to `$argv[`*n*`]`.

`$*` Equivalent to `$argv[*]`.

The modifiers `:e`, `:h`, `:q`, `:r`, `:t` and `:x` can be applied (see `History Substitution`), as can `:gh`, `:gt` and `:gr`. If `{ }` (braces) are used, then the modifiers must appear within the braces. The current implementation allows only one such modifier per expansion.

The following references may not be modified with `:` modifiers.

`$?`*var*
`${?`*var*`}` Substitutes the string 1 if *var* is set or 0 if it is not set.

`$?0` Substitutes 1 if the current input filename is known, or 0 if it is not.

`$$` Substitute the 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 C shell script.

Command and Filename Substitutions

Command and filename substitutions are applied selectively to the arguments of built-in commands. Portions of expressions that are not evaluated are not expanded. For non-built-in commands, filename expansion of the command name is done separately from that of the argument list; expansion occurs in a subshell, after I/O redirection is performed.

Command Substitution

A command enclosed by backquotes (`...`) is performed by a subshell. Its standard output is broken into separate words at each space character, tab and newline; null words are discarded. This text replaces the backquoted string on the current command line. Within double-quotes, only newline characters force new words; space and tab characters are preserved. However, a final newline is ignored. It is therefore possible for a command substitution to yield a partial word.

Filename Substitution

Unquoted words containing any of the characters `*`, `?`, `[` or `{`, or that begin with `~`, are expanded (also known as *globbing*) to an alphabetically sorted list of filenames, as follows:

`*`	Match any (zero or more) characters.
`?`	Match any single character.
`[...]`	Match any single character in the enclosed list(s) or range(s). A list is a string of characters. A range is two characters separated by a minus-sign (-), and includes all the characters in between in the ASCII collating sequence [see `ascii`(7)].
`{` *str*, *str*, ... `}`	Expand to each string (or filename-matching pattern) in the comma-separated list. Unlike the pattern-matching expressions above, the expansion of this construct is not sorted. For instance, `{b,a}` expands to 'b' 'a', (not 'a' 'b'). As special cases, the characters `{` and `}`, along with the string `{}`, are passed undisturbed.
~[*user*]	Your home directory, as indicated by the value of the variable `home`, or that of *user*, as indicated by the password entry for *user*.

Only the patterns `*`, `?` and `[...]` imply pattern matching; an error results if no filename matches a pattern that contains them. The '.' (dot character), when it is the first character in a filename or pathname component, must be matched explicitly. The / (slash) must also be matched explicitly.

Expressions and Operators

A number of C shell built-in commands accept expressions, in which the operators are similar to those of C and have the same precedence. These expressions typically appear in the `@`, `exit`, `if`, `set` and `while` commands, and are often used to regulate the flow of control for executing commands. Components of an expression are separated by white space.

Null or missing values are considered 0. The result of all expressions are strings, which may represent decimal numbers.

The following C shell operators are grouped in order of precedence:

`(...)`	grouping
~	one's complement
!	logical negation
* / %	multiplication, division, remainder (These are right associative, which can lead to unexpected results. Group combinations explicitly with parentheses.)

`+ -`	addition, subtraction (also right associative)
`<< >>`	bitwise shift left, bitwise shift right
`< > <= >=`	less than, greater than, less than or equal to, greater than or equal to
`== != =~ !~`	equal to, not equal to, filename-substitution pattern match (described below), filename-substitution pattern mismatch
`&`	bitwise AND
`^`	bitwise XOR (exclusive or)
`|`	bitwise inclusive OR
`&&`	logical AND
`| |`	logical OR

The operators: `==`, `!=`, `=~`, and `!~` compare their arguments as strings; other operators use numbers. The operators `=~` and `!~` each check whether or not a string to the left matches a filename substitution pattern on the right. This reduces the need for `switch` statements when pattern-matching between strings is all that is required.

Also available are file inquiries:

`-r` *filename*	Return true, or 1 if the user has read access. Otherwise it returns false, or 0.
`-w` *filename*	True if the user has write access.
`-x` *filename*	True if the user has execute permission (or search permission on a directory).
`-e` *filename*	True if *file* exists.
`-o` *filename*	True if the user owns *file*.
`-z` *filename*	True if *file* is of zero length (empty).
`-f` *filename*	True if *file* is a plain file.
`-d` *filename*	True if *file* is a directory.

If *file* does not exist or is inaccessible, then all inquiries return false.

An inquiry as to the success of a command is also available:

`{` *command* `}`	If *command* runs successfully, the expression evaluates to true, 1. Otherwise it evaluates to false 0. (Note that, conversely, *command* itself typically returns 0 when it runs successfully, or some other value if it encounters a problem. If you want to get at the status directly, use the value of the `status` variable rather than this expression).

Control Flow

The shell contains a number of commands to regulate the flow of control in scripts, and within limits, from the terminal. These commands operate by forcing the shell either to reread input (to *loop*), or to skip input under certain conditions (to *branch*).

Each occurrence of a `foreach`, `switch`, `while`, `if...then` and `else` built-in must appear as the first word on its own input line.

If the shell's input is not seekable and a loop is being read, that input is buffered. The shell performs seeks within the internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward `goto` commands will succeed on nonseekable inputs.)

Command Execution

If the command is a C shell built-in, the shell executes it directly. Otherwise, the shell searches for a file by that name with execute access. If the command-name contains a /, the shell takes it as a pathname, and searches for it. If the command-name does not contain a /, the shell attempts to resolve it to a pathname, searching each directory in the `path` variable for the command. To speed the search, the shell uses its hash table (see the `rehash` built-in) to eliminate directories that have no applicable files. This hashing can be disabled with the `-c` or `-t`, options, or the `unhash` built-in.

As a special case, if there is no / in the name of the script and there is an alias for the word `shell`, the expansion of the `shell` alias is prepended (without modification), to the command line. The system attempts to execute the first word of this special (late-occurring) alias, which should be a full pathname. Remaining words of the alias's definition, along with the text of the input line, are treated as arguments.

When a pathname is found that has proper execute permissions, the shell forks a new process and passes it, along with its arguments to the kernel (using the `execve`(2) system call). The kernel then attempts to overlay the new process with the desired program. If the file is an executable binary (in `a.out`(4) format) the kernel succeeds, and begins executing the new process. If the file is a text file, and the first line begins with `#!`, the next word is taken to be the pathname of a shell (or command) to interpret that script. Subsequent words on the first line are taken as options for that shell. The kernel invokes (overlays) the indicated shell, using the name of the script as an argument.

If neither of the above conditions holds, the kernel cannot overlay the file (the `execve`(2) call fails); the C shell then attempts to execute the file by spawning a new shell, as follows:

- If the first character of the file is a #, a C shell is invoked.
- Otherwise, a standard (Bourne) shell is invoked.

Signal Handling

The shell normally ignores QUIT signals. Background jobs are immune to signals generated from the keyboard, including hangups (HUP). Other signals have the values that the C shell inherited from its environment. The shell's handling of interrupt and terminate signals within scripts can be controlled by the `onintr` built-in. Login shells catch the TERM signal; otherwise this signal is passed on to child processes. In no case are interrupts allowed when a login shell is reading the `.logout` file.

Job Control

The shell associates a numbered *job* with each command sequence, to keep track of those commands that are running in the background or have been stopped with TSTP signals (typically CTRL-z). When a command, or command sequence (semicolon separated list), is started in the background using the `&` metacharacter, the shell displays a line with the job number in brackets, and a list of associated process numbers:

```
[1] 1234
```

To see the current list of jobs, use the `jobs` built-in command. The job most recently stopped (or put into the background if none are stopped) is referred to as the *current* job, and is indicated with a '+'. The previous job is indicated with a '-'; when the current job is terminated or moved to the foreground, this job takes its place (becomes the new current job).

To manipulate jobs, refer to the `bg`, `fg`, `kill`, `stop` and `%` built-ins.

A reference to a job begins with a '%'. By itself, the percent-sign refers to the current job.

`% %+ %%`	The current job.
`%-`	The previous job.
%j	Refer to job *j* as in: '`kill -9 %j`'. *j* can be a job number, or a string that uniquely specifies the command-line by which it was started; '`fg %vi`' might bring a stopped `vi` job to the foreground, for instance.
`%?`*string*	Specify the job for which the command-line uniquely contains *string*.

A job running in the background stops when it attempts to read from the terminal. Background jobs can normally produce output, but this can be suppressed using the '`stty tostop`' command.

Status Reporting

While running interactively, the shell tracks the status of each job and reports whenever a finishes or becomes blocked. It normally displays a message to this effect as it issues a prompt, so as to avoid disturbing the appearance of your input. When set, the `notify` variable indicates that the shell is to report status changes immediately. By default, the `notify` command marks the current process; after starting a background job, type `notify` to mark it.

Built-In Commands

Built-in commands are executed within the C shell. If a built-in command occurs as any component of a pipeline except the last, it is executed in a subshell.

`:` Null command. This command is interpreted, but performs no action.

`alias` [*name* [*def*]]
Assign *def* to the alias *name*. *def* is a list of words that may contain escaped history-substitution metasyntax. *name* is not allowed to be `alias` or `unalias`. If *def* is omitted, the alias *name* is displayed along with its current definition. If both *name* and *def* are omitted, all aliases are displayed.

`bg` [*%job*] . . .
Run the current or specified jobs in the background.

`break` Resume execution after the `end` of the nearest enclosing `foreach` or `while` loop. The remaining commands on the current line are executed. This allows multilevel breaks to be written as a list of `break` commands, all on one line.

`breaksw` Break from a `switch`, resuming after the `endsw`.

`case` *label*:
A label in a `switch` statement.

`cd` [*dir*]
`chdir` [*dir*]
Change the shell's working directory to directory *dir*. If no argument is given, change to the home directory of the user. If *dir* is a relative pathname not found in the current directory, check for it in those directories listed in the `cdpath` variable. If *dir* is the name of a shell variable whose value starts with a `/`, change to the directory named by that value.

`continue` Continue execution of the nearest enclosing `while` or `foreach`.

`default:` Labels the default case in a `switch` statement. The default should come after all `case` labels. Any remaining commands on the command line are first executed.

`dirs` [`-l`]
Print the directory stack, most recent to the left; the first directory shown is the current directory. With the `-l` argument, produce an unabbreviated printout; use of the ~ notation is suppressed.

`echo` [`-n`] *list*
The words in *list* are written to the shell's standard output, separated by space characters. The output is terminated with a newline unless the `-n` option is used.

`eval` *argument* ...
Reads the arguments as input to the shell, and executes the resulting command(s). This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See `tset`(1) for an example of how to use `eval`.

`exec` *command*
Execute *command* in place of the current shell, which terminates.

`exit` [(*expr*)]
The shell exits, either with the value of the STATUS variable, or with the value of the specified by the expression `expr`.

`fg %` [*job*]
Bring the current or specified *job* into the foreground.

`foreach` *var* (*wordlist*)
...
`end`
The variable *var* is successively set to each member of *wordlist*. The sequence of commands between this command and the matching `end` is executed for each new value of *var*. (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` to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with `?` before any statements in the loop are

executed.

`glob` *wordlist*
Perform filename expansion on *wordlist*. Like `echo`, but no `\` escapes are recognized. Words are delimited by `NULL` characters in the output.

`goto` *label* The specified *label* is filename and command expanded to yield a label. The shell rewinds its input as much as possible and searches for a line of the form *label*`:` possibly preceded by space or tab characters. Execution continues after the indicated line. It is an error to jump to a label that occurs between a `while` or `for` built-in, and its corresponding `end`.

`hashstat` Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding `execs`). An `exec` is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component that does not begin with a '/'.

`history` [`-hr`] [*n*]
Display the history list; if *n* is given, display only the *n* most recent events.

`-r` Reverse the order of printout to be most recent first rather than oldest first.

`-h` Display the history list without leading numbers. This is used to produce files suitable for sourcing using the `-h` option to *source*.

`if (`*expr*`)` *command*
If the specified expression evaluates to 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, or a parenthesized command list. Note: I/O redirection occurs even if `expr` is false, when *command* is *not* executed (this is a bug).

`if (`*expr*`) then`
...
`else if (`*expr2*`) then`
...
`else`
...
`endif` If `expr`"" is true, commands up to the first `else` are executed. Otherwise, if *expr2* is true, the commands between the `else if` and the second `else` are executed. Otherwise, commands between the `else` and the `endif` are executed. Any number of `else if` pairs are allowed, but only one `else`. Only one `endif` is needed, but it is required. The words `else` and `endif` must be the first nonwhite characters on a line. The `if` must appear alone on its input line or after an `else`.)

`jobs`[`-l`]
List the active jobs under job control.

`-l` List process IDs, in addition to the normal information.

`kill` [*-sig*] [*pid*] [*%job*] ...

`kill -l` Send the TERM (terminate) signal, by default, or the signal specified, to the specified process ID, the *job* indicated, or the current *job*. Signals are either given by number or by name. There is no default. Typing `kill` does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

`-l` List the signal names that can be sent.

`limit` [`-h`] [*resource* [*max-use*]]

Limit the consumption by the current process or any process it spawns, each not to exceed *max-use* on the specified *resource*. If *max-use* is omitted, print the current limit; if *resource* is omitted, display all limits.

`-h` Use hard limits instead of the current limits. Hard limits impose a ceiling on the values of the current limits. Only the privileged user may raise the hard limits.

resource is one of:

`cputime`	Maximum CPU seconds per process.
`filesize`	Largest single file allowed.
`datasize`	Maximum data size (including stack) for the process.
`stacksize`	Maximum stack size for the process.
`coredumpsize`	Maximum size of a core dump (file).

max-use is a number, with an optional scaling factor, as follows:

n`h`	Hours (for `cputime`).
n`k`	*n* kilobytes. This is the default for all but `cputime`.
n`m`	*n* megabytes or minutes (for `cputime`).
mm`:`*ss*	Minutes and seconds (for `cputime`).

`login` [*username* | `-p`]

Terminate a login shell and invoke `login`(1). The `.logout` file is not processed. If *username* is omitted, `login` prompts for the name of a user.

`-p` Preserve the current environment (variables).

`logout` Terminate a login shell.

`nice` [*+n* | *-n*] [*command*]

Increment the process priority value for the shell or for *command* by *n*. The higher the priority value, the lower the priority of a process, and the slower it runs. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple `if` commands apply. If *command* is omitted, `nice` increments the value for the current shell. If no increment is specified, `nice` sets the process priority value to 4. The range of process priority values is from -20 to 20. Values of *n* outside this range set the value to the lower, or to the higher boundary, respectively.

+*n* Increment the process priority value by *n*.
-*n* Decrement by *n*. This argument can be used only by the privileged user.

`nohup` [*command*]
Run *command* with HUPs ignored. With no arguments, ignore HUPs throughout the remainder of a script. When given, *command* is always run in a subshell, and the restrictions placed on commands in simple `if` commands apply. All processes detached with `&` are effectively `nohup`'d.

`notify` [`%`*job*] ...
Notify the user asynchronously when the status of the current, or of specified jobs, changes.

`onintr` [- | *label*]
Control the action of the shell on interrupts. With no arguments, `onintr` restores the default action of the shell on interrupts. (The shell terminates shell scripts and returns to the terminal command input level). With the - argument, the shell ignores all interrupts. With a *label* argument, the shell executes a `goto` *label* when an interrupt is received or a child process terminates because it was interrupted.

`popd` [+*n*] Pop the directory stack, and `cd` to the new top directory. The elements of the directory stack are numbered from 0 starting at the top.

+*n* Discard the *n*'th entry in the stack.

`pushd` [+*n* | *dir*]
Push a directory onto the directory stack. With no arguments, exchange the top two elements.

+*n* Rotate the *n*'th entry to the top of the stack and `cd` to it.
dir Push the current working directory onto the stack and change to *dir*.

`rehash` Recompute the internal hash table of the contents of directories listed in the *path* variable to account for new commands added.

`repeat` *count command*
Repeat *command count* times. *command* is subject to the same restrictions as with the one-line `if` statement.

`set` [*var* [= *value*]]
`set` *var*`[`*n*`]` = *word*
With no arguments, `set` displays the values of all shell variables. Multi-word values are displayed as a parenthesized list. With the *var* argument alone, `set` assigns an empty (null) value to the variable *var*. With arguments of the form *var* = *value* `set` assigns *value* to *var*, where *value* is one of:

word A single word (or quoted string).
(*wordlist*) A space-separated list of words enclosed in parentheses.

Values are command and filename expanded before being assigned. The form `set` *var*`[`*n*`]` `=` *word* replaces the *n*'th word in a multiword value with *word*.

`setenv` [*VAR* [*word*]]
With no arguments, `setenv` displays all environment variables. With the *VAR* argument sets the environment variable *VAR* to have an empty (null) value. (By convention, environment variables are normally given upper-case names.) With both *VAR* and *word* arguments `setenv` sets the environment variable `NAME` to the value *word*, which must be either a single word or a quoted 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. In addition, the shell sets the `PWD` environment variable from the `csh` variable `cwd` whenever the latter changes.

`shift` [*variable*]
The components of `argv`, or *variable*, if supplied, are shifted to the left, discarding the first component. It is an error for the variable not to be set, or to have a null value.

`source` [`-h`] *name*
Reads commands from *name*. `source` commands may be nested, but if they are nested too deeply the shell may run out of file descriptors. An error in a sourced file at any level terminates all nested `source` commands.

`-h` Place commands from the the file *name* on the history list without executing them.

`stop` [`%`*job*] ...
Stop the current or specified background job.

`suspend` Stop the shell in its tracks, much as if it had been sent a stop signal with `^Z`. This is most often used to stop shells started by `su`.

`switch (`*string*`)`
`case` *label*`:`
...
`breaksw`
...
`default:`
...
`breaksw`
`endsw` Each *label* is successively matched, against the specified *string*, which is first command and filename expanded. The file metacharacters `*`, `?` and `[...]` may be used in the case labels, which are variable expanded. If none of the labels match before a default label is found, execution begins after the default label. Each `case` statement and the `default` statement must appear at the beginning of a line. The command `breaksw` continues execution after the `endsw`. Otherwise control falls through subsequent `case` and `default` statements as with C. If no label matches and there is no default, execution continues after the `endsw`.

`time` [*command*]
With no argument, print a summary of time used by this C shell and its children. With an optional *command*, execute *command* and print a summary of the time it uses.

`umask` [*value*]
Display the file creation mask. With *value* set the file creation mask. *value* is given in octal, and is XORed with the permissions of 666 for files and 777 for directories to arrive at the permissions for new files. Common values include 002, giving complete access to the group, and read (and directory search) access to others, or 022, giving read (and directory search) but not write permission to the group and others.

`unalias` *pattern*
Discard aliases that match (filename substitution) *pattern*. All aliases are removed by `unalias *`.

`unhash` Disable the internal hash table.

`unlimit` [`-h`] [*resource*]
Remove a limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed. See the description of the `limit` command for the list of *resource* names.

`-h` Remove corresponding hard limits. Only the privileged user may do this.

`unset` *pattern*
Remove variables whose names match (filename substitution) *pattern*. All variables are removed by 'unset *'; this has noticeably distasteful side-effects.

`unsetenv` *variable*
Remove *variable* from the environment. Pattern matching, as with `unset` is not performed.

`wait` Wait for background jobs to finish (or for an interrupt) before prompting.

`while` (*expr*)
...
`end` While `expr` is true (evaluates to non-zero), repeat commands between the `while` and the matching `end` statement. `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 shell's input is a terminal, it prompts for commands with a question-mark until the `end` command is entered and then performs the commands in the loop.

`%` [*job*] [`&`]
Bring the current or indicated *job* to the foreground. With the ampersand, continue running *job* in the background.

`@` [*var* `=expr`]

`@ [` *var* `[`*n*`] =expr ]`
With no arguments, display the values for all shell variables. With arguments, the variable *var*, or the *n*'th word in the value of *var* , to the value that `expr` evaluates to. (If `[`*n*`]` is supplied, both *var* and its *n*'th component must already exist.)

If the expression contains the characters `>`, `<`, `&` or `|`, then at least this part of `expr` must be placed within parentheses.

The operators `*=`, `+=`, etc., are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of `expr` that would otherwise be single words.

Special postfix operators, `++` and `--` increment or decrement *name*, respectively.

Environment Variables and Predefined Shell Variables

Unlike the standard shell, the C shell maintains a distinction between environment variables, which are automatically exported to processes it invokes, and shell variables, which are not. Both types of variables are treated similarly under variable substitution. The shell sets the variables `argv`, `cwd`, `home`, `path`, `prompt`, `shell`, and `status` upon initialization. The shell copies the environment variable `USER` into the shell variable `user`, `TERM` into `term`, and `HOME` into `home`, and copies each back into the respective environment variable whenever the shell variables are reset. `PATH` and `path` are similarly handled. You need only set `path` once in the `.cshrc` or `.login` file. The environment variable `PWD` is set from `cwd` whenever the latter changes. The following shell variables have predefined meanings:

`argv` Argument list. Contains the list of command line arguments supplied to the current invocation of the shell. This variable determines the value of the positional parameters `$1`, `$2`, and so on.

`cdpath` Contains a list of directories to be searched by the `cd`, `chdir`, and `popd` commands, if the directory argument each accepts is not a subdirectory of the current directory.

`cwd` The full pathname of the current directory.

`echo` Echo commands (after substitutions), just before execution.

`fignore` A list of filename suffixes to ignore when attempting filename completion. Typically the single word '`.o`'.

`filec` Enable filename completion, in which case the CTRL-d character CTRL-d) and the ESC character have special significance when typed in at the end of a terminal input line:

- EOT Print a list of all filenames that start with the preceding string.
- ESC Replace the preceding string with the longest unambiguous extension.

`hardpaths` If set, pathnames in the directory stack are resolved to contain no symbolic-link components.

`histchars` A two-character string. The first character replaces `!` as the history-substitution character. The second replaces the carat (`^`) for quick substitutions.

`history` The number of lines saved in the history list. A very large number may use up all of the C shell's memory. If not set, the C shell saves only the most recent command.

`home` The user's home directory. The filename expansion of `~` refers to the value of this variable.

`ignoreeof` If set, the shell ignores EOF from terminals. This protects against accidentally killing a C shell by typing a CTRL-d.

`mail` A list of files where the C shell checks for mail. If the first word of the value is a number, it specifies a mail checking interval in seconds (default 5 minutes).

`nobeep` Suppress the bell during command completion when asking the C shell to extend an ambiguous filename.

`noclobber` Restrict output redirection so that existing files are not destroyed by accident. `>` redirections can only be made to new files. `>>` redirections can only be made to existing files.

`noglob` Inhibit filename substitution. This is most useful in shell scripts once filenames (if any) are obtained and no further expansion is desired.

`nonomatch` Returns the filename substitution pattern, rather than an error, if the pattern is not matched. Malformed patterns still result in errors.

`notify` If set, the shell notifies you immediately as jobs are completed, rather than waiting until just before issuing a prompt.

`path` The list of directories in which to search for commands. `path` is initialized from the environment variable `PATH`, which the C shell updates whenever `path` changes. A null word specifies the current directory. The default is typically: `(. /usr/ucb /usr/bin)`. If `path` becomes unset only full pathnames will execute. An interactive C shell will normally hash the contents of the directories listed after reading `.cshrc`, and whenever `path` is reset. If new commands are added, use the `rehash` command to update the table.

`prompt` The string an interactive C shell prompts with. Noninteractive shells leave the `prompt` variable unset. Aliases and other commands in the `.cshrc` file that are only useful interactively, can be placed after the following test: `'if ($?prompt == 0) exit'`, to reduce startup time for noninteractive shells. A `!` in the `prompt` string is replaced by the current event number. The default prompt is *hostname*`%` for mere mortals, or *hostname*`#` for the privileged user.

`savehist` The number of lines from the history list that are saved in `~/.history` when the user logs out. Large values for `savehist` slow down the C shell during startup.

`shell` The file in which the C shell resides. This is used in forking shells to interpret files that have execute bits set, but that are not executable by the system.

`status` The status returned by the most recent command. If that command terminated abnormally, 0200 is added to the status. Built-in commands that fail return exit status 1, all other built-in commands set status to 0.

`time` Control automatic timing of commands. Can be supplied with one or two values. The first is the reporting threshold in CPU seconds. The second is a string of tags and text indicating which resources to report on. A tag is a percent sign (`%`) followed by a single *upper-case* letter (unrecognized tags print as text):

- `%D` Average amount of unshared data space used in Kilobytes.
- `%E` Elapsed (wallclock) time for the command.
- `%F` Page faults.
- `%I` Number of block input operations.
- `%K` Average amount of unshared stack space used in Kilobytes.
- `%M` Maximum real memory used during execution of the process.
- `%O` Number of block output operations.
- `%P` Total CPU time — U (user) plus S (system) — as a percentage of E (elapsed) time.
- `%S` Number of seconds of CPU time consumed by the kernel on behalf of the user's process.
- `%U` Number of seconds of CPU time devoted to the user's process.
- `%W` Number of swaps.
- `%X` Average amount of shared memory used in Kilobytes.

The default summary display outputs from the `%U`, `%S`, `%E`, `%P`, `%X`, `%D`, `%I`, `%O`, `%F` and `%W` tags, in that order.

`verbose` Display each command after history substitution takes place.

FILES

`~/.cshrc`	Read at beginning of execution by each shell.
`~/.login`	Read by login shells after `.cshrc` at login.
`~/.logout`	Read by login shells at logout.
`~/.history`	Saved history for use at next login.
`/usr/bin/sh`	Standard shell, for shell scripts not starting with a '`#`'.
`/tmp/sh*`	Temporary file for '`<<`'.
`/etc/passwd`	Source of home directories for '*~name*'.

SEE ALSO

`login`(1), `sh`(1) `access`(2), `exec`(2), `fork`(2), `pipe`(2), `a.out`(4), `environ`(4), `termio`(4), `ascii`(5).

DIAGNOSTICS

`You have stopped jobs.`

You attempted to exit the C shell with stopped jobs under job control. An immediate second attempt to exit will succeed, terminating the stopped jobs.

NOTES

Words can be no longer than 1024 characters. The system limits argument lists to 1,048,576 characters. However, the maximum number of arguments to a command for which filename expansion applies is 1706. Command substitutions may expand to no more characters than are allowed in the 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, the shell prints the directory it started in if this is different from the current directory; this can be misleading (that is, 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 suspend *b*, the shell never executes *c*. This is especially noticeable if the expansion results from an alias. It can be avoided by placing the sequence in parentheses to force it into a subshell.

Multiline shell procedures should be provided, as they are with the standard (Bourne) shell.

Commands within loops, prompted for by `?`, are not placed in the *history* list.

Control structures 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. There are two problems with `:` modifier usage on variable substitutions: not all of the modifiers are available, and only one modifier per substitution is allowed.

The `g` (global) flag in history substitutions applies only to the first match in each word, rather than all matches in all words. The the standard text editors consistently do the latter when given the `g` flag in a substitution command.

Quoting conventions are confusing. Overriding the escape character to force variable substitutions within double quotes is counterintuitive and inconsistent with the Bourne shell.

Symbolic links can fool the shell. Setting the `hardpaths` variable alleviates this.

'`set path`' should remove duplicate pathnames from the pathname list. These often occur because a shell script or a `.cshrc` file does something like '`set path=(/usr/local /usr/hosts $path)`' to ensure that the named directories are in the pathname list.

The only way to direct the standard output and standard error separately is by invoking a subshell, as follows:

`example%` (*command* > *outfile*) >& *errorfile*

Although robust enough for general use, adventures into the esoteric periphery of the C shell may reveal unexpected quirks.

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.

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*. Characters from supplementary code sets can be used for *prefix*.

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). See ed(1) for a description of how to specify a regular expression.

%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 user's responsibility to remove it if it is no longer wanted.

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 splits 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}
```

If `prog.c` follows the normal C coding convention (the last line of a routine consists only of a `}` in the first character position), this example creates a file for each separate C routine (up to 21) in `prog.c`.

INTERNATIONAL FUNCTIONS

`csplit` can process characters from supplementary code sets. In regular expressions, searches are performed on characters, not bytes.

The indicated size of the files created is in bytes, not the number of characters.

SEE ALSO

`ed`(1), `sh`(1).

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.

NAME

ct - spawn login to a remote terminal

SYNOPSIS

ct [*options*] *telno* . . .

DESCRIPTION

ct dials the telephone number of a modem that is attached to a terminal and spawns a login process to that terminal. *Telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. (The set of legal characters for *telno* is 0 through 9, -, =, **, and #. 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.

ct tries each line listed in the file /etc/uucp/Devices until it finds an available line with appropriate attributes, or runs out of entries. ct uses the following options:

-h Normally, ct hangs up the current line so it can be used to answer the incoming call. The -h option prevents this action. The -h option also waits for the termination of the specified ct process before returning control to the user's terminal.

-s *speed* The data rate may be set with the -s option. *speed* is expressed in baud rates. The default baud rate is 1200.

-v If the -v (verbose) option is used, ct sends a running narrative to the standard error output stream.

-w *n* If there are no free lines ct asks if it should wait for one, and if so, for how many minutes it should wait before it gives up. ct continues to try to open the dialers at one-minute intervals until the specified limit is exceeded. This dialogue may be overridden by specifying the -w *n* option where *n* is the maximum number of minutes that ct is to wait for a line.

-x*n* This option is used for debugging; it produces a detailed output of the program execution on standard error. *n* is a single number between 0 and 9. As *n* increases to 9, more detailed debugging information is given.

After the user on the destination terminal logs out, there are two things that could occur, depending on what type of port monitor is monitoring the port. In the case of no port monitor, ct prompts: Reconnect? If the response begins with the letter n, the line is dropped; otherwise, ttymon is started again and the login: prompt is printed. In the second case, where a port monitor is monitoring the port, the port monitor reissues the login: prompt.

The user should log out properly before disconnecting.

FILES

/etc/uucp/Devices
/var/adm/ctlog

SEE ALSO

cu(1C), login(1), uucp(1C), ttymon(1M).

NOTES

The ct program will not work with a DATAKIT Multiplex interface.

For a shared port, one used for both dial-in and dial-out, the ttymon program running on the line must have the -r and -b options specified [see ttymon(1M)].

NAME

ctags - create a tags file for use with vi

SYNOPSIS

ctags [-aBFtuvwx] [-f *tagsfile*] *filename* . . .

DESCRIPTION

ctags makes a tags file for ex(1) from the specified C, Pascal, FORTRAN, YACC, and LEX sources. A tags file gives the locations of specified objects (in this case functions and typedefs) 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. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by SPACE or TAB characters. Using the tags file, ex can quickly find these objects definitions.

Normally ctags places the tag descriptions in a file called tags; this may be overridden with the -f option.

Files with names ending in .c or .h are assumed to be C source files and are searched for C routine and macro definitions. Files with names ending in .y are assumed to be YACC source files. Files with names ending in .l are assumed to be LEX files. 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.

The tag main is treated specially in C programs. The tag formed is created by prepending M to *filename*, 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.

The following options are available:

-a Append output to an existing tags file.

-B Use backward searching patterns (?...?).

-F Use forward searching patterns (/.../) (default).

-t Create tags for typedefs.

-u Update the specified files in tags, that is, all references to them are deleted, and the new values are appended to the file. Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the tags file.

-v Produce on the standard output an index listing the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through sort -f.

-w Suppress warning diagnostics.

-x Produce a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

FILES

`tags`	output tags file

USAGE

The `-v` option is mainly used with `vgrind` which will be part of the optional BSD Compatibility Package.

SEE ALSO

`ex`(1), `vgrind`(1), `vi`(1)

NOTES

Recognition of `functions`, `subroutines` and `procedures` for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

`ctags` does not know about `#ifdefs`.

`ctags` should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of `-tx` shows only the last line of typedefs.

NAME

ctrace - C program debugger

SYNOPSIS

ctrace [*options*] [*file*]

DESCRIPTION

The ctrace command allows the user to monitor the sequential execution of a C program as each program statement executes. The effect is similar to executing a shell procedure with the -x option. ctrace reads the C program in *file* (or from standard input if the user does not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. The output of ctrace must be placed into a temporary file because the cc(1) command does not allow the use of a pipe. This file can then be compiled and executed.

As each statement in the program executes, it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement; these variable names and values will be followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed after each 1000 loop cycles to help the user detect infinite loops. The trace output goes to the standard output so the user can put it into a file for examination with an editor or the bfs(1) or tail(1) commands.

The options commonly used are:

-f *functions* Trace only these *functions.*
-v *functions* Trace all but these *functions.*

The user may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. char, short, and int variables are also printed as signed integers and, if appropriate, as characters. double variables are printed as floating point numbers in scientific notation. The user can request that variables be printed in additional formats, if appropriate, with these options:

-o Octal
-x Hexadecimal
-u Unsigned
-e Floating point

These options are used only in special circumstances:

-l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.
-s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.
-t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The diagnostics section explains when to use this option.
-P Preprocess the input before tracing it. The user can also use the -D, -I, and -U cc(1) options.

-p *string*
: Change the trace print function from the default of `printf`. For example, `fprintf(stderr`, would send the trace to the standard error output.

-r*f*
: Use file *f* in place of the `runtime.c` trace function package. This replacement lets the user change the entire print function, instead of just the name and leading arguments (see the `-p` option).

-V
: Prints version information on the standard error.

-Q*arg*
: If *arg* is `y`, identification information about `ctrace` will be added to the output files. This can be useful for software administration. Giving `n` for *arg* explicitly asks for no such information, which is the default behavior.

EXAMPLE

If the file `lc.c` contains this C program:

```
 1 #include <stdio.h>
 2 main()  /* count lines in input */
 3 {
 4    int c, nl;
 5
 6    nl = 0;
 7    while ((c = getchar()) != EOF)
 8         if (c = '\n')
 9              ++nl;
10    printf("%d\n", nl);
11 }
```

these commands and test data are entered:

```
cc lc.c
a.out
1
(cntl-d)
```

the program will be compiled and executed. The output of the program will be the number `2`, which is incorrect because there is only one line in the test data. The error in this program is common, but subtle. If the user invokes `ctrace` with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
 2 main()
 6    nl = 0;
      /* nl == 0 */
 7    while ((c = getchar()) != EOF)
```

The program is now waiting for input. If the user enters the same test data as before, the output will be:

```
      /* c == 49 or '1' */
 8         if (c = '\n')
           /* c == 10 or '\n' */
 9              ++nl;
```

```
                /* nl == 1 */
 7    while ((c = getchar()) != EOF)
      /* c == 10 or '\n' */
 8          if (c = '\n')
            /* c == 10 or '\n' */
 9                ++nl;
                  /* nl == 2 */
 7    while ((c = getchar()) != EOF)
```

If an end-of-file character (cntl-d) is entered, the final output will be:

```
      /* c == -1 */
10    printf("%d\n", nl);
      /* nl == 2 */2
       return
```

Note the information printed out at the end of the trace line for the `nl` variable following line 10. Also note the `return` comment added by `ctrace` at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable `c` is assigned the value '`1`' in line 7, but in line 8 it has the value '`\n`'. Once user attention is drawn to this `if` statement, he or she will probably realize that the assignment operator (`=`) was used in place of the equality operator (`==`). This error can easily be missed during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for `ctrace` is to trace the entire program file, unless the `-f` or `-v` options are used to trace specific functions. The default operation does not give the user statement-by-statement control of the tracing, nor does it let the user turn the tracing off and on when executing the traced program.

The user can do both of these by adding `ctroff()` and `ctron()` function calls to the program to turn the tracing off and on, respectively, at execution time. Thus, complex criteria can be arbitrarily coded for trace control with `if` statements, and this code can even be conditionally included because `ctrace` defines the `CTRACE` preprocessor variable. For example:

```
#ifdef CTRACE
     if (c == '!' && i > 1000)
          ctron();
#endif
```

These functions can also be called from `tbx(1)` if they are compiled with the `-g` option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
tbx a.out
associate main:7 "call ctroff"
associate main:11 "call ctron"
run
```

The trace can be turned off and on by setting static variable `tr_ct_` to `0` and `1`, respectively. This on/off option is useful if a user is using a debugger that can not call these functions directly.

FILES

/usr/ccs/lib/ctrace/runtime.c	run-time trace package

DIAGNOSTICS

This section contains diagnostic messages from both ctrace and cc(1), since the traced code often gets some cc warning messages. The user can get cc error messages in some rare cases, all of which can be avoided.

ctrace Diagnostics

`warning: some variables are not traced in this statement`
: Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the -t option to increase this number.

`warning: statement too long to trace`
: This statement is over 400 characters long. Make sure that tabs are used to indent the code, not spaces.

`cannot handle preprocessor code, use -P option`
: This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

`'if ... else if' sequence too long`
: Split the sequence by removing an else from the middle.

`possible syntax error, try -P option`
: Use the -P option to preprocess the ctrace input, along with any appropriate -D, -I, and -U preprocessor options.

NOTES

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g., "int EOF;", will cause a syntax error.

Pointer values are always treated as pointers to character strings.

ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. ctrace may choose to print the address of an aggregate or use the wrong format (e.g., 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate.

The loop trace output elimination is done separately for each file of a multi-file program. Separate output elimination can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

SEE ALSO

bfs(1), tbx(1), tail(1), ctype(3C), fclose(3S), printf(3S), string(3C).

NAME

cu - call another UNIX system

SYNOPSIS

cu [options] [destination]

DESCRIPTION

cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of files. It is convenient to think of cu as operating in two phases. The first phase is the connection phase in which the connection is established. cu then enters the conversation phase. The -d option is the only one that applies to both phases.

-d Causes diagnostic traces to be printed.

Connection Phase

cu uses the same mechanism that uucp does to establish a connection. This means that it will use the uucp control files /etc/uucp/Devices and /etc/uucp/Systems. This gives cu the ability to choose from several different media to establish the connection. The possible media include telephone lines, direct connections, and local area networks (LAN). The Devices file contains a list of media that are available on your system. The Systems file contains information for connecting to remote systems, but it is not generally readable.

The *destination* parameter from the command line is used to tell cu what system you wish to connect to. *destination* can be blank, a telephone number, a system name, or a LAN specific address. A telephone number is a string consisting of the tone dial characters (the digits 0 through 9, *, and #) plus the special characters = and -. The equal sign designates a secondary dial tone and the minus sign creates a 4 second delay. A system name is the name of any computer that uucp can call; the uuname command prints a list of these names. The documentation for your LAN will show the form of the LAN specific address.

If cu's default behavior is invoked (not using the -c or -l options), cu will use *destination* to determine which medium to use. If *destination* is a telephone number, cu will assume that you wish to use a telephone line and it will select an automatic call unit (ACU). If the *destination* is not a telephone number, then cu will assume that it is a system name. cu will follow the uucp calling mechanism and use the Systems and Devices files to obtain the best available connection. Since cu will choose a speed that is appropriate for the medium that it selects, you may not use the -s option when *destination* is a system name.

The -c and -l options modify this default behavior. -c is most often used to select a LAN by specifying a Type field from the Devices file. Here, *destination* is assumed to be a system name. If the connection attempt to system name fails, a connection will be attempted using *destination* as a LAN specific address. The -l option is used to specify a device associated with a direct connection. If the connection is truly a direct connection to the remote machine, then there is no need to specify a *destination*. This is the only case where a blank *destination* is allowed. On the other hand, there may be cases in which the specified device connects to a dialer, so it is valid to specify a telephone number as a *destination*. The -c and -l options should not be specified on the same command line.

cu accepts many options. The -c, -l, and -s options play a part in selecting the medium; the remaining options are used in configuring the line.

-s*speed* Specifies the transmission speed (300, 1200, 2400, 4800, 9600). The default value is "Any" speed which will depend on the order of the lines in the /etc/uucp/Devices file. Most modems are either 300, 1200, or 2400 baud. Directly connected lines may be set to a speed higher than 2400 baud.

-c*type* The first field in the Devices file is the "Type" field. The -c option forces cu to only use entries in the "Type" field that match the user specified *type*. The specified *type* is usually the name of a local area network.

-l*line* Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place 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 Devices file record in which *line* matches the second field (the Line field). When the -l and -s options are both used together, cu will search the Devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed, otherwise, an error message will be printed and the call will not be made. In the general case where a specified device is a directly connected asynchronous line (for example, /dev/term/*ab*), a telephone number (*telno*) is not required. The specified device need not be in the /dev directory. If the specified device is associated with an auto dialer, a telephone number must be provided. If *destination* is used with this option, it must be a telephone number.

-b*n* Forces *n* to be the number of bits processed on the line. *n* is either 7 or 8. This allows connection between systems with different character sizes. By default, the character size of the line is set to the same as the current local terminal.

-e Set an EVEN data parity. This option designates that EVEN parity is to be generated for data sent to the remote system.

-h Set communication mode to half-duplex. This option emulates the local echo(1) command in order to support calls to other computer systems that expect terminals to be set to half-duplex mode.

-n Request user prompt for telephone number. For added security, this option will prompt the user to provide the telephone number to be dialed, rather than taking it from the command line.

-o Set an ODD data parity. This option designates that ODD parity is to be generated for data sent to the remote system.

-t Used to dial a terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

Conversation Phase

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 so the buffer is not overrun. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following user initiated commands:

~.	terminate the conversation.
~!	escape to an interactive shell on the local system.
~!*cmd*...	run *cmd* on the local system (via `sh -c`).
~$*cmd*...	run *cmd* locally and send its output to the remote system.
~%cd	change the directory on the local system. Note: ~!cd will cause the command to be run by a sub-shell, probably not what was intended.
~%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.
~~ line	send the line ~ line to the remote system.
~%break	transmit a BREAK to the remote system (which can also be specified as ~%b).
~%debug	toggles the -d debugging option on or off (which can also be specified as ~%d).
~t	prints the values of the termio structure variables for the user's terminal (useful for debugging).
~l	prints the values of the termio structure variables for the remote communication line (useful for debugging).
~%ifc	toggles between DC3/DC1 input control protocol and no input control. This is useful when the remote system does not respond properly to the DC3 and DC1 characters. (can also be specified as ~%nostop).
~%ofc	toggles the output flow control setting. When enabled, outgoing data may be flow controlled by the remote host (can also be specified as ~%noostop).
~%divert	allow/disallow unsolicited diversions. That is, diversions not specified by ~%take.
~%old	allow/disallow old style syntax for received diversions.

The *receive* process normally copies data from the remote system to the standard output of the local system. It may also direct the output to local files.

The use of ~%put requires stty(1) and cat(1) on the remote side. It also requires that the current control characters on the remote system be identical to the current control characters on the local system. Backslashes are inserted at appropriate places for these control characters.

The use of ~%take requires the existence of echo(1) and cat(1) on the remote system. Also, tabs mode [see stty(1)] should be set on the remote system if tabs are to be copied without expansion to spaces.

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 by using ~~. Executing a tilde command reminds the user of the local system uname. For example, uname can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!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.

EXAMPLES

To dial a system whose telephone number is 9 1 201 555 1234 using 1200 baud (where dialtone is expected after the 9):

```
cu -s1200 9=12015551234
```

If the speed is not specified, "Any" is the default value.

To login to a system that is on a Datakit VCS local area network, but which has not been defined by your administrator (i.e., is not entered in the /etc/uucp/Systems file(s)):

```
cu -c DK address
```

DK is the name of the Datakit local area network, and *address* is the Datakit address which is of the form, /area/exchange/machine.

To login to a system connected by a direct line:

```
cu -l /dev/term/XX
```

or

```
cu -l term/XX
```

To dial a system with a specific line and speed:

```
cu -s1200 -l term/XX
```

To dial a system using a specific line associated with an auto dialer:

```
cu -l culXX 9=12015551234
```

To use a system name:

```
cu systemname
```

FILES

/etc/uucp/Sysfiles
/etc/uucp/Systems
/etc/uucp/Devices
/var/spool/locks/*

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uucp(1C), uname(1), uuname(1)

DIAGNOSTICS

Exit code is zero for normal exit, otherwise, one.

NOTES

The cu command does not do any integrity checking on data it transfers. Data fields with special cu characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a ~. to terminate the conversion, even if stty 0 has been used. Non-printing characters are not dependably transmitted using either the ~%put or ~%take commands. cu, between an IMBR1 and a PENRIL modem, will not return a login prompt immediately upon connection. A carriage return will return the prompt.

~%put and ~%take cannot be used over multiple links. Files must be moved one link at a time.

There is an artificial slowing of transmission by cu during the ~%put operation so that loss of data is unlikely. Files transferred using ~%take or ~%put must contain a trailing newline, otherwise, the operation will hang. Entering a CTRL-d command usually clears the hang condition.

NAME

cunix - configure a new bootable operating system

SYNOPSIS

cunix [-a *"ld_args"*] [-b *boot_dir*] [-c *config_dir*] [-d] [-e *edtfile*]
[-f system] [-g] [-i *loader_directive_file*] [-j *assembler*] [-l *link_ed*]
[-o *outfile*] [-s *script_file*] [-v] [-x *multiplier*] [-z *"as_args"*] [-E] [-I]

DESCRIPTION

The cunix command creates a new bootable operating system file from the object files (drivers) specified in the given system file.

The configuration of a new bootable operating system is usually done when new hardware or software is added to or removed from the system; most frequently it is done during a powerup or reboot of the system. The cunix command allows this procedure to be performed at the user level, without a powerdown or system reboot. The options to cunix also allow the user to create customized input files for the configuration process, and to choose the location for the resulting bootable operating system. The ICDDEV: field in the system(4) file directs cunix to incorporate an In-Core Disk (ICD) image within the bootable operating system.

Both COFF and ELF format object files can be used as input to cunix.

The options to cunix are as follows:

-a Pass the specified *ld_args* as arguments to the link editor; the entire set of arguments must be enclosed in double quotes, with each argument surrounded by white space. By default (no -a specified), -x is passed to the link editor as an argument for COFF format object files (directs the link editor to omit local symbols from the output symbol table, saving some space in the output file); if one or more object files is in ELF format, then no loader arguments are passed by default. The link editor ld is used by default, unless another is specified with the -l option (see below).

-b *boot_dir* specifies the directory where driver object files reside; the default is /boot.

-c *config_dir* specifies the directory that contains working files for cunix; the default is /config.

-d Build the operating system with debug mode on; the default is debug mode off. Debug mode populates the sysm68ksym or sysm88ksym symbol table with symbols from the kernel object file and drivers specified in the system file. The -d option causes cunix to use more disk space and time. The sysm68ksym table is accessible through the sysm68k system call; the sysm88ksym table is accessible through the sysm88k system call.

-e Use the EDT data from *edtfile* rather than /stand/edt_data.

-f system specifies the file that contains configuration information; the default is /stand/system.

-g Do not remove the *config_dir*/conf.o or *config_dir*/conf.s files after the bootable operating system has been created; the default is to remove conf.o and conf.s. The directory *config_dir* is either /config or the directory specified by -c, above.

`-i` *loader_directive_file* to be used for configuration; a *loader_directive_file* specifies memory locations for loading the operating system at boot time. A *loader_directive_file* for a COFF system is called an `ifile`, while a *loader_directive_file* for an ELF system is called a `mapfile`. Normally, it is not necessary to specify a *loader_directive_file*. Only use the `-i` option with a custom *loader_directive_file*.

`-j` Use the assembler to assemble `conf.s`. By default, `as` is invoked as if it had been typed as a command to the terminal.

`-l` Use the *link_ed* link editor to bind object files; the link editor `ld` is used by default. The *PATH-* variable is searched for the `ld` program. See NOTES.

`-o` *outfile* specifies the output file name for the bootable operating system; the default is `/stand/unix_test`.

`-s` Run the script file after creating `conf.s` but before invoking the assembler. *Script_file* may be a shell script or an executable object. The script file may be used to massage `conf.s` before the assembler is invoked. In this use, *script_file* must reference `conf.s` explicitly since `cunix` does not pass it as an argument to *script_file*.

`-v` Verbose mode on; `cunix` displays all the modules and drivers being linked. The default is verbose mode off.

`-x` Cunix must allocate certain areas of the kernel being built with fixed sizes. This option specifies a multiplier that will be applied to the compiled in sizes so the areas can be made sufficiently large. Use this option if the error `Internal space problems when building conf.o, try '-x <number> option` occurs.

`-z` Pass the specified *"as_args"* to the assembler. The entire set of arguments must be enclosed in double quotes, with each argument surrounded by white space. No flags are passed by default.

`-E` Use only the EDT data file. Do not read the in-core EDT. Normally `cunix` reads the in-core EDT and then the EDT data file and combines them, ignoring entries in the data file that duplicate entries in the in-core EDT and replacing entries in the in-core EDT with corresponding entries from the data file, if they match on name and board fields, but differ otherwise. By default `cunix` reads `/stand/edt_data` for the data file. Use the `-e` flag to change this.

`-I` Use only the in-core EDT. Do not use the EDT data file.

NOTES

Do not execute a separate `ld ... -o /stand/unix` command for the operating system; the output file is processed by `cunix` after loading. `cunix` calls the `setrlimit` system call to set the file size limit to infinity. It proceeds without complaint if `setrlimit` fails.

FILES

*/boot_dir/**	drivers to be configured into the operating system
/config_dir`/conf.o`	object file created by `cunix`

/config_dir/ifile*	loader directive file(s) for COFF system
/config_dir/mapfile*	loader directive file(s) for ELF system
/stand/system	system file
/stand/unix	bootable operating system
/dev/rSA/disk1	default location of root file system

SEE ALSO

buildsys(1M), ld(1), mkboot(1M), rc6(1M), sysm68k(2), sysm88k(2), system(4).

NAME

cut - cut out selected fields of each line of a file

SYNOPSIS

```
cut -clist [file ...]
cut -flist [-dchar] [-s] [file ...]
```

DESCRIPTION

Use cut to cut out columns from a table or fields from each line of a file; in database 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. In addition, a filename of "-" explicitly refers to standard input.

The options are as follows:

list A comma-separated list of integer field numbers (in increasing order), with optional - to indicate 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 column positions (e.g., -c1-72 would pass the first 72 column positions of each line). When multibyte characters are split at a specified position, the remaining column positions are filled with an appropriate number of ASCII spaces instead of characters.

-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. The field delimiter *char* can be a character from a supplementary code set.

-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.

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.

EXAMPLES

```
cut -d: -f1,5 /etc/passwd
```
mapping of user IDs to names

```
name=`who am i | cut -f1 -d" "`
```
to set name to current login name

DIAGNOSTICS

ERROR: line too long
A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: 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 called for by the *list* .

`ERROR: no fields` The *list* is empty.

`ERROR: no delimiter`
Missing *char* on `-d` option.

`ERROR: cannot handle multiple adjacent backspaces`
Adjacent backspaces cannot be processed correctly.

`WARNING: cannot open <filename>`
Either *filename* cannot be read or does not exist. If multiple filenames are present, processing continues.

INTERNATIONAL FUNCTIONS

`cut` can process characters from supplementary code sets.

SEE ALSO

`grep`(1), `paste`(1).

NAME

`cvtomflib` - convert OMF (XENIX) libraries to ELF

SYNOPSIS

`cvtomflib` [`-v`] [`-o` *outfile*] *library* [*library*. . .]

DESCRIPTION

`cvtomflib` converts libraries of OMF objects to libraries of ELF objects. It is intended for use with application packages that provide only OMF libraries that could not otherwise be used with the Standard C Development Environment.

The options have the following meanings.

`-v` Verbose output is produced for each converted object. Without this option, `cvtomflib` does its work silently.

`-o` This option allows the user to specify a new name, *outfile*, for the converted library without changing the original. This option is only available when a single library is being converted.

NOTES

The original order of objects within the library is retained.

Each library is converted in the directory in which it's located. Without the `-o` option, the converted library will overwrite the original; therefore, you may want to copy the original library before conversion.

NAME

cxref - generate C program cross-reference

SYNOPSIS

cxref [*options*] *files*

DESCRIPTION

The cxref command analyzes a collection of C files and builds a cross-reference table. cxref uses a special version of cc to include #define'd information in its symbol table. It generates a list of all symbols (auto, static, and global) in each individual file, or, with the -c option, in combination. The table includes four fields: NAME, FILE, FUNCTION, and LINE. The line numbers appearing in the LINE field also show reference marks as appropriate. The reference marks include:

assignment	=
declaration	-
definition	*

If no reference marks appear, you can assume a general reference.

OPTIONS

cxref interprets the -D, -I, -U options in the same manner that cc does. In addition, cxref interprets the following options:

-c	Combine the source files into a single report. Without the -c option, cxref generates a separate report for each file on the command line.
-d	Disables printing declarations, making the report easier to read.
-l	Does not print local variables. Prints only global and file scope statistics.
-o*file*	Direct output to *file*.
-s	Operates silently; does not print input file names.
-t	Format listing for 80-column width.
-w*num*	Width option that formats output no wider than *num* (decimal) columns. This option will default to 80 if *num* is not specified or is less than 51.
-C	Runs only the first pass of cxref, creating a .cx file that can later be passed to cxref. This is similar to the -c option of cc or lint.
-F	Prints the full path of the referenced file names.
-L*cols*	Modifies the number of columns in the LINE field. If you do not specify a number, cxref defaults to five columns.
-V	Prints version information on the standard error.

-Wname,file,function,line
Changes the default width of at least one field. The default widths are:

Field	Characters
NAME	15
FILE	13
FUNCTION	15
LINE	20 (4 per column)

FILES

TMPDIR`/tcx.*`	temporary files
TMPDIR`/cx.*`	temporary files
LIBDIR`/xref`	accessed by `cxref`
LIBDIR	usually `/usr/ccs/lib`
TMPDIR	usually `/var/tmp` but can be redefined by setting the environment variable `TMPDIR` [see `tempnam` in `tmpnam`(3S)].

EXAMPLE

`a.c`

```
1    main()
2    {
3         int i;
4         extern char c;
5
6         i=65;
7         c=(char)i;
8    }
```

Resulting cross-reference table:

```
NAME   FILE         FUNCTION   LINE
c      a.c          ---        4-     7=
i      a.c          main       3*     6=   7
main   a.c          ---        2*
u3b2   predefined   ---        0*
unix   predefined   ---        0*
```

SEE ALSO

`cc`(1), `lint`(1)

DIAGNOSTICS

Error messages usually mean you cannot compile the files.

NAME

`date` - print and set the date

SYNOPSIS

`date` [`-n`] [`-u`] [+ *format*]
`date` [`-n`] [`-a`] [-] *sss.fff*] [`-u`] [[*mmdd*] *HHMM* | *mmddHHMM* [*cc*] *yy*]

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 (only by super-user). If `in.timed` is running, the date will be set via the time daemon. Otherwise, it will be set locally.

`-a` [-] *sss.fff*
: Slowly adjust the time by *sss.fff* seconds (*fff* represents fractions of a second). This adjustment can be positive or negative. The system's clock will be sped up or slowed down until it has drifted by the number of seconds specified.

`-n`
: Do not set the date via the time daemon.

`-u`
: Display (or set) the date in Greenwich Mean Time (GMT—universal time), bypassing the normal conversion to (or from) local time.

mm
: Month number

dd
: Day number in the month

HH
: Hour number (24 hour system)

MM
: Minute number

cc
: Century minus one

yy
: Last 2 digits of the year number

The month, day, year, and century may be omitted; the current values are supplied as defaults. For example:

```
date 10080045
```

sets the date to Oct 8, 12:45 AM. The current year is the default because no year is supplied. The system operates in GMT. `date` takes care of the conversion to and from local standard and daylight time. Only the super-user may change the date. After successfully setting the date and time, `date` displays the new date according to the default format. The `date` command uses `TZ` to determine the correct time zone information [see `environ`(5)].

+ *format*
: If the argument begins with +, the output of `date` is under the control of the user. Each Field Descriptor, described below, is preceded by `%` and is replaced in the output by its corresponding value. A single `%` is encoded by `%%`. All other characters are copied to the output without change. The string is always terminated with a new-line character. If the argument contains embedded blanks it must be quoted (see the EXAMPLE section).

Specifications of native language translations of month and weekday names are supported. The month and weekday names used for a language are based on the locale specified by the environment variables `LC_TIME` and `LANG` (see `environ`(5)).

The current date and time can be set and displayed using single-byte or multibyte characters in accordance with the customary local format. Characters from supplementary code sets can be used in *+format*.

The month and weekday names used for a language are taken from a file whose format is specified in `strftime`(4). This file also defines country-specific date and time formats such as `%c`, which specifies the default date format. The following form is the default for `%c`:

```
%a %b %e %T %Z %Y
```

e.g., Fri Dec 23 10:10:42 EST 1988

Field Descriptors (must be preceded by a `%`):

- `a` abbreviated weekday name
- `A` full weekday name
- `b` abbreviated month name
- `B` full month name
- `c` country-specific date and time format
- `d` day of month - 01 to 31
- `D` date as `%m/%d/%y`
- `e` day of month - 1 to 31 (single digits are preceded by a blank)
- `h` abbreviated month name (alias for `%b`)
- `H` hour - 00 to 23
- `I` hour - 01 to 12
- `j` day of year - 001 to 366
- `m` month of year - 01 to 12
- `M` minute - 00 to 59
- `n` insert a new-line character
- `p` string containing ante meridiem or post meridiem indicator (by default, AM or PM)
- `r` time as `%I:%M:%S %p`
- `R` time as `%H:%M`
- `S` second - 00 to 61, allows for leap seconds
- `t` insert a tab character
- `T` time as `%H:%M:%S`
- `U` week number of year (Sunday as the first day of the week) - 00 to 53
- `w` day of week - Sunday = 0
- `W` week number of year (Monday as the first day of the week) - 00 to 53
- `x` Country-specific date format
- `X` Country-specific time format
- `y` year within century - 00 to 99
- `Y` year as *ccyy* (4 digits)
- `Z` timezone name

Date and Time Daemon Interaction

The following summarizes the effects of setting the date on a machine which is running a time daemon.

On a machine on which the time daemon is running as a master, you type:

`date some_date`
: The time is propagated to the slaves immediatedly.

`date -n some_date`
: The time is not propagated to the slaves immediately but will be propagated on the next master initiated network time synchronization.

On a machine on which the time daemon is running as a slave, you type:

`date some_date`
: The time is propagated to, and resets the master's notion of network time.

`date -n some_date`
: The local time is set to `some_date`, but only until the next network time synchronization at which point the machine's time will revert to network time.

EXAMPLE

The command

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

generates as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

DIAGNOSTICS

Exit Codes: 0 - success; 1 - failure; 2 - failed setting the date via the time daemon so the date was set locally.

`No permission` You are not the super-user and you try to change the date.

`bad conversion` The date set is syntactically incorrect.

NOTES

Should you need to change the date while the system is running multi-user, use the `datetime` command of `sysadm`(1M).

If you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

The current date and time can be set and displayed using single-byte or multibyte characters in accordance with the customary local format. Characters from supplementary code sets can be used in *+format*.

SEE ALSO

`sysadm`(1M), `in.timed`(1M), `strftime`(4), `environ`(5).

NAME

dbcmd - load command and macro files into a kernel executable file

SYNOPSIS

dbcmd *file macro*

DESCRIPTION

dbcmd loads the contents of the specified macros into the kernel executable *file*. The next time the kernel is rebooted with *file*, the loaded commands are part of the kernel debugger.

SEE ALSO

kdb(1M), dbsym(1M), kcrash(1M)

NAME

dbsym - add symbols to kernel debugger

SYNOPSIS

dbsym [-v] *file1 file2*

DESCRIPTION

dbsym extracts the symbolic names and addresses from the kernel executable file, *file1*, and enters the data into *file2*. When the system is rebooted with *file2*, the symbolic information can now be used by the kernel debugger. Note that *file1* and *file2* can be the same.

-v The verbose option , -v, displays various symbol information.

SEE ALSO

kdb(1M), dbcmd(1M), kcrash(1M)

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. [bc is a preprocessor for dc that provides infix notation and a C-like syntax that implements functions. bc also provides reasonable control structures for programs. See bc(1).] 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. 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. 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.

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.

Q
: Exits the program. 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.

<*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 a UNIX system command.

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.

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.

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(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1)

DIAGNOSTICS

`x is unimplemented`: *x* is an octal number.

`stack empty`: not enough 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 being kept around.

`Out of pushdown`: too many items on the stack.

`Nesting Depth`: too many levels of nested execution.

NAME

dcon - control dual console operation

SYNOPSIS

`dcon [on|off|get|help]`

DESCRIPTION

The utility command `dcon` is used to control and report the dual console state of the system.

`on` Starts dual console mode. If dual console mode is already started this fact is reported and no state change occurs. This operation can only be performed by `root`. This command must be performed on the terminal port which will eventually become the dual console.

`off` Stops dual console mode. If dual console mode is not started then this fact is reported and no state change occurs. This operation can only be performed by `root`.

`get` Report the dual console state. This is the default operation if no specific operation is specified.

`help` Print out the usage for the command `dcon`.

DIAGNOSTICS

Dual console is ON.
Dual console is ON and was started at system boot.
Dual console is already ON.
Dual console is OFF.
Dual console is already OFF.
Dual console mode can't be started from the system console.
Dual console mode must be stopped from the system console.
Dual console state can only be changed by ROOT.
Open of `/dev/conctl` failed, errno = <errno>.
The system console is not open.
GC_DCON_GET ioctl failed, errno = <errno>.
GC_DCON_ON ioctl failed, errno = <errno>.
GC_DCON_OFF ioctl failed, errno = <errno>.

EXIT VALUES

0 Desired operation completed successfully.

1 `dcon` command was given too many arguments.

2 `dcon` command was given an unknown argument.

3 One of the required ioctl operations failed.

4 The open of `/dev/conctl` failed.

FILES

`/dev/conctl`
`/dev/console`

SEE ALSO

console(7)

NAME

dcopy (generic) - copy file systems for optimal access time

SYNOPSIS

dcopy [-F *FSType*] [-V] [*current_options*] [-o *specific_options*] *inputfs outputfs*

DESCRIPTION

dcopy copies file system *inputfs* to *outputfs*. *inputfs* is the device file for the existing file system; *outputfs* is the device file to hold the reorganized result. For the most effective optimization *inputfs* should be the raw device and *outputfs* should be the block device. Both *inputfs* and *outputfs* should be unmounted file systems.

current_options are options supported by the s5-specific module of dcopy. Other FSTypes do not necessarily support these options. *specific_options* indicate suboptions specified in a comma-separated list of suboptions and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command. See dcopy_*FSType*(1M) for details.

The options are:

-F Specify the *FSType* on which to operate. The *FSType* should either be specified here or be determinable from /etc/vfstab by matching the *inputfs* (device) with an entry in the table.

-V Echo the complete command line, but do not execute the command. The command line is generated by using the options and arguments provided by the user and adding to them information derived from /etc/vfstab. This option should be used to verify and validate the command line.

-o Specify FSType-specific options.

NOTE

This command may not be supported for all FSTypes.

FILES

/etc/vfstab list of default parameters for each file system

SEE ALSO

dcopy_s5(1M), vfstab(4).

NAME

dcopy (s5) - copy s5 file systems for optimal access time

SYNOPSIS

dcopy [-F s5] [*generic_options*] [-s*X*] [-a*n*] [-d] [-v] [-f*fsize*[:*isize*]] *inputfs outputfs*

DESCRIPTION

generic_options are options supported by the generic dcopy command.

With no options, dcopy copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap.

The options are:

-F s5	Specifies the s5-FSType. Need not be supplied if the information may be obtained from /etc/vfstab by matching the *inputfs* device with an entry in the file.
-s*X*	Supply device information for creating an optimal organization of blocks in a file. *X* must be of the form *cylinder size:gap size.*
-a*n*	Place the files not accessed in *n* days after the free blocks of the destination file system If no *n* is specified then no movement occurs.
-d	Leave order of directory entries as is. The default is to move subdirectories to the beginning of directories.
-v	Reports how many files were processed and how big the source and destination freelists are.
-f *fsize*[:*isize*]	Specify the *outputfs* file system (*fsize*) and inode list (*isize*) sizes in logical blocks. If the suboption (or :*isize*) is not given, the values from *inputfs* are used.

dcopy catches interrupts and quits and reports on its progress. To terminate dcopy, send a quit signal followed by an interrupt or quit.

NOTES

fsck should be run on the new file system created by dcopy before it is mounted.

FILES

/etc/mnttab	list of file systems currently mounted

SEE ALSO

generic dcopy(1M), fsck(1M), mkfs(1M)

NAME

dd - convert and copy a 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. The input and output block sizes may be specified to take advantage of raw physical I/O.

option	values
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size *n* bytes (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, preserve the input block size instead of packing short blocks into the output buffer (this is particularly efficient since no in-core copy need be done)
cbs=*n*	conversion buffer size (logical record length)
files=*n*	copy and concatenate *n* input files before terminating (makes sense only where input is a magnetic tape or similar device)
skip=*n*	skip *n* input blocks before starting copy (appropriate for magnetic tape, where *iseek* is undefined)
iseek=*n*	seek *n* blocks from beginning of input file before copying (appropriate for disk files, where *skip* can be incredibly slow)
oseek=*n*	seek *n* blocks from beginning of output file before copying
seek=*n*	identical to *oseek*, retained for backward compatibility
count=*n*	copy only *n* input blocks
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
block	convert new-line terminated ASCII records to fixed length
unblock	convert fixed length ASCII records to new-line terminated records
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error (limit of 5 consecutive errors)
sync	pad every input block to *ibs*
. . . , . . .	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with k, b, or w to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by x to indicate multiplication.

cbs is used only if ascii, *unblock*, *ebcdic*, *ibm*, or *block* conversion is specified. In the first two cases, *cbs* characters are copied into the conversion buffer, any specified character mapping is done, trailing blanks are trimmed and a new-line is added before sending the line to the output. In the latter three cases, characters are read

into the conversion buffer and blanks are added to make up an output record of size *cbs*. If *cbs* is unspecified or zero, the `ascii`, *ebcdic*, and *ibm* options convert the character set without changing the block structure of the input file; the *unblock* and *block* options become a simple file copy.

After completion, `dd` reports the number of whole and partial input and output blocks.

EXAMPLE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per tape block into the ASCII file *x*:

```
dd if=/dev/rmt/0h of=x ibs=800 obs=8k cbs=80
conv=ascii,lcase
```

Note the use of raw magnetic tape. `dd` is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary block sizes.

SEE ALSO

`cp`(1)

NOTES

Do not use `dd` to copy files between filesystems having different block sizes.

Using a blocked device to copy a file will result in extra nulls being added to the file to pad the final block to the block boundary.

DIAGNOSTICS

f+p records in(out) numbers of full and partial blocks read(written)

NAME

`ddefs` - disk definition information manager

SYNOPSIS

`/etc/ddefs` [`-d` ddefsdir] `-n` [diskname]
`/etc/ddefs` [`-d` ddefsdir] `-erp` diskname

DESCRIPTION

The *ddefs* utility is used to add to or modify the information that describes disks. The disk definitions are contained in files in the `/etc/dskdefs` directory. The files in `/etc/dskdefs` are read by the `dinit`(1M) program to obtain format information about the disks.

To specify the directory, use the `-d` option followed by a directory name on the command line, as shown in the SYNOPSIS section. If no *ddefsdir* is given, the default is `/etc/dskdefs`.

The *diskname* is either a "device", specified as /dev/rdsk/*devicename*, or a "*ddefs file*" in the directory *ddefsdir*.

The *ddefs* program provides several options both for interactive and non-interactive use. UNIX System V/68 and UNIX System V/88 only support the 182Mb CDC WREN and the 390Mb ITI ESDI drives with a MVME323 controller, however this command contains additional options for other controllers and drives. The options are:

`-n` [*diskname*]
: (Interactive) Create a new disk definition, where *diskname* corresponds to the *type* values listed for `dinit`(1M) (for example, m323182 for a 182Mb CDC ESDI drive). If a name is given that already exists, the program automatically shifts to edit mode (`-e` option). If no *diskname* is given, the user is prompted for a name. When creating a new definition, *ddefs* will prompt for the name of a "template" disk. The template disk is usually one with similar, but not identical, attributes.

`-e` *diskname*
: (Interactive) Edit an existing disk definition. If the *diskname* given does not exist, the program switches automatically to create mode (`-n` option).

`-r` *diskname*
: (Interactive) Edit an existing disk definition (read-only).

`-p` *diskname*
: (Non-interactive) Print an existing disk definition.

For every type of operation, the user has the option of specifying a directory for the disk definition file. If an existing definition is being edited, this is the directory where the definition file can be found. If a new definition is being created, this is the directory where the definition file will be placed. It is also the directory where the template can be found when this feature is used.

Each disk definition is formatted as a series of lines, each line giving a parameter name followed by a value. To modify a definition, move the cursor to the appropriate line and type the new value. When creating a new definition file, *ddefs* will not write the definition until all parameters are initialized (changed).

To obtain "help" information about a parameter, type a `?` after the parameter name. The elements of the definition file and the help information for each are given in the following paragraphs.

When *diskname* is a "*ddefs file*", all parameters are readable and writeable. When *diskname* is a device, the following conventions are used to indicate how a parameter may be accessed: a single asterisk (*) following a parameter name indicates that the user has read access in interactive mode; a double asterisk (**) indicates that the parameter may be read and written in interactive mode; no asterisk indicates that the parameter is not applicable for a raw device (that is, the parameter exists only in the *ddefs file*).

Comment
: This information is general comments. It cannot contain more than one line. Usually, the comment information is a description of the drive type supported by the definition file. If no comment is desired, type `none`.

Disk type **
: This value is any unique integer that is used to identify the disk drive type. Each size drive on each controller should have a different type.

Format command
: The format program, if specified, is called by `dinit`(1M) to format a disk drive. The format program line should specify all options necessary to format the disk. All drives supported by UNIX System V/68 and UNIX System V/88 can be formatted directly by `dinit`(1M). Therefore this field should be specified as `none`.

Diagnostic tracks *
: To reserve diagnostic tracks, type `yes`. If not, type `no`.

Bad spot strategy *
: If the controller only supports perfect media, type `perfect`.

  BAD TRACKS: If bad track replacement is done by the bad track replacement software, type `software`. If the controller automatically performs bad track replacement, type `hardware`.

  BAD SECTORS: If the controller supports automatic bad sector replacement and requires the cylinder, head, and sector of each bad sector, then type `sector`.

  BAD SPOTS: If the controller supports automatic bad spot replacement and requires the cylinder, head, and byte offset of each bad spot, then type `spot`.

Maximum number of bad spots
: This value is the maximum number of bad spots expected on a disk of this type. This many alternate spots (sectors or tracks) will be allocated for this drive.

Number of sectors **
: This value is the total number of sectors on the disk drive.

Sector size (in bytes) **
: This value is the disk sector size (specified in bytes). Currently, it must be `128, 256,` or `512` bytes.

Sectors per track **
: This value is the number of usable sectors per track on the formatted media.

Cylinders **
: This value is the total number of cylinders on the disk media.

Heads **
: This value is the number of read/write heads on the drive. It is equivalent to the number of tracks per cylinder.

Precompensation cylinder **
: This value is the disk cylinder number to start write precompensation.

Sector interleave **
: This value is the sector interleave factor used during disk formatting. For no interleave (or one-to-one interleave), type `1`. Some controllers will automatically select an appropriate interleave factor when given an interleave value of `0`.

Spiral offset *
: This value is the spiral offset applied when formatting disks. If no spiral offset is wanted, then type `0`.

Step rate **
: This value is the seek step rate used when accessing the disk. Some controllers will automatically select an appropriate step rate when given a step rate of 0.

Starting head number **
: This value is the starting head number of the drive. Most drives and controllers require a starting head number of zero.

ECC error length **
: This value is the error correcting code data burst length.

Attributes mask (hex) **
: This value is the disk attributes mask. Bits in this mask determine which bits in the attributes word are valid.

Extended attributes mask (hex) **
: This value is the extended attributes mask.

Attributes word (hex) **
: This value is the disk attributes word. The (hexadecimal) bit definitions are:

NAME	BIT	FIELD USE	BIT OFF	BIT ON
ATWAS	0x0400	Alternate sectors?	no	yes
ATWFS	0x0200	Floppy size	5.25"	8"
ATWPC	0x0100	Precomp style	pre-write	post-read
ATWSK	0x0080	Seek after head change?	no	yes

ATWDD	0x0040	Track density of drive	single	double
ATWEN	0x0020	Encoding method	FM	MFM
ATWDT	0x0010	Disk type	floppy	hard
ATWSN	0x0008	Sector Numbering	Motorola	IBM
ATWDS	0x0004	Number of sides	single	double
ATWTD	0x0002	Track density of floppy	8" floppy	5.25" floppy
ATWMF	0x0001	Data density of medium	single	double

Extended attributes word (hex) **
: This value is the extended attributes word.

Gap byte 1 (hex) **
: This value is the first 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Gap byte 2 (hex) **
: This value is the second 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Gap byte 3 (hex) **
: This value is the third 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Gap byte 4 (hex) **
: This value is the fourth 'gap byte' required for formatting a disk. This parameter is controller-and drive-specific and may not be used by some controllers.

Controller attribute (hex) **
: This value is controller-and drive-specific information and may not be used by some controllers.

Unformatted sector size **
: This value is the unformatted sector size on the disk including the headers, gaps, ECC, and data. This parameter is controller and drive-specific, and may not be used by some controllers.

Sector slip count **
: This value is the sector slip count used while formatting to implement controller supported sector slipping. This count is the number of 'slip' sectors per track.

Slice count *
: This value is the dynamic slice count. If zero, the driver will not use dynamic slicing. Legal non-zero slice counts are 8, 16, 32, 64, and 128. Some controllers support only a subset of these legal slice counts.

Root file system offset *
: This value is the 1024-byte block offset of the `root` file system.

Root file system size
: This value, if non-zero, is the size of the `root` file system (in 512-byte blocks) created on slice zero of the drive after it is formatted. If no file system is desired, then type `0`.

`/usr` file system size
: This value, if non-zero, is the size of the `/usr` file system (in 512-byte blocks) created in slice one or two after the drive is formatted. If no `/usr` file system is desired, type `0`. The `/usr` file system is created only if a `root` file system is also created.

`/usr` file system slice
: This value specifies the slice for the `/usr` file system. This value must not be zero but may coincide with the swap slice.

Swap size
: This value, if non-zero, specifies the size of the system swap space (in 512-byte blocks). The swap space will be placed following the `root` file system if the swap slice is set to zero. If the swap slice and the `/usr` file system slice are both one, swap follows the `/usr` file system in slice one.

Swap slice
: This value specifies the swap slice. It may be set to zero (to share the `root` file system slice), or to one (to occupy its own slice or share with the `/usr` file system), or to two (to occupy its own slice following the `/usr` file system slice).

End-of-disk reserved area
: This value is the number of blocks reserved for system use at the end of the disk. This area may be used for bad track alternates or for diagnostic tracks.

Alternates list
: To add to the alternates list, type `add #[-#]`. To delete from the alternates list, type `delete #[-#]`. The '#' is any non-zero number. The sequence '#-#' specifies a range of non-zero numbers (including the end-points). To have no alternates, type `none`. The value printed in parentheses is the total number of alternates in the list.

When creating a disk description using *ddefs*, the following information must be given to have `dinit`(1M) create a slice table:

a non-zero slice count; currently restricted to 0, 8, 16, 32, 64, 128

the `root` file system offset

the `root` file system size

the size of swap (if any)

if swap is defined, what slice it resides on (allowable slices: 0, 1, 2)

the `/usr` file system size (if any)

if `/usr` is defined, what slice it resides on (allowable slices: 0, 1, 2)

the size of the "end-of-disk" reserved area. This area is used for diagnostic and/or alternate tracks.

NOTES

The elements "root file system size", "`/usr` file system slice", "swap size", and "swap slice" are not used and are present to maintain backwards compatibility.

FILES

`/etc/dskdefs/*` disk definition file

SEE ALSO

`dinit`(1M)

NAME

`delsysadm` - `sysadm` interface menu or task removal tool

SYNOPSIS

`delsysadm` *task* | [`-r`] *menu*

DESCRIPTION

The `delsysadm` command deletes a *task* or *menu* from the `sysadm` interface and modifies the interface directory structure on the target machine.

task | *menu* The logical name and location of the menu or task within the interface menu hierarchy. Begin with the top menu `main` and proceed to where the menu or the task resides, separating each name with colons. See EXAMPLES.

If the `-r` option is used, this command will recursively remove all sub-menus and tasks for this menu. If the `-r` option is not used, the menu must be empty.

`delsysadm` should only be used to remove items added as "on-line" changes with the `edsysadm` command. Such an addition will have a package instance tag of ONLINE. If the task or menu (and its sub-menus and tasks) have any package instance tags other than ONLINE, you are asked whether to continue with the removal or to exit. Under these circumstances, you probably do not want to continue and you should rely on the package involved to take the necessary actions to delete this type of entry.

The command exits successfully or provides the error code within an error message.

EXAMPLES

To remove the `nformat` task, execute:

```
delsysadm main:applications:ndevices:nformat.
```

DIAGNOSTICS

0 Successful execution
2 Invalid syntax
3 Menu or task does not exist
4 Menu not empty
5 Unable to update interface menu structure

NOTES

Any menu that was originally a placeholder menu (one that only appears if sub-menus exist under it) will be returned to placeholder status when a deletion leaves it empty.

When the `-r` option is used, `delsysadm` checks for dependencies before removing any subentries. (A dependency exists if the menu being removed contains an entry placed there by an application package). If a dependency is found, the user is shown a list of packages that depend on the menu being deleted and asked whether or not to continue. If the answer is yes, the menu and all of its menus and tasks are removed (even those shown to have dependencies). If the answer is no, the menu is not deleted.

`delsysadm` should only be used to remove menu or task entries that have been added to the interface with `edsysadm`.

SEE ALSO

`edsysadm`(1M), `sysadm`(1M)

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 -e (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 the **NOTES** section); 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 on 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 gets 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* Specify a *list* [see get(1) for the definition of *list*] of deltas that 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 newline 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) that will validate the correctness of the MR numbers. If a

non-zero exit status is returned from the 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 newline character terminates the comment text. Characters from supplementary code sets can be used in *comment*.

`-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

g-file	Existed before the execution of `delta`; removed after completion of `delta`.
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`.
`bdiff`	Program to compute differences between the "gotten" file and the g-file.

INTERNATIONAL FUNCTIONS

Characters from supplementary code sets can be used for the SCCS filenames *file*. However, they must begin with the `s.` ASCII characters. SCCS files may also include characters from supplementary code sets.

SEE ALSO

`admin`(1), `bdiff`(1), `cdc`(1), `get`(1), `help`(1), `prs`(1), `rmdel`(1), `sccsfile`(4).

DIAGNOSTICS

Use `help`(1) for explanations.

NOTES

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.

Comments are limited to text strings of at most 1024 characters. Line lengths greater than 1000 characters cause undefined results.

NAME

deroff - remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [-m *x*] [-w] [*file* . . .]

DESCRIPTION

deroff reads each of the *files* in sequence and removes all troff(1) requests, macro calls, backslash constructs, eqn(1) constructs (between .EQ and .EN lines, and between delimiters), and tbl(1) descriptions, perhaps 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 troff 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 may be followed by an m, s, or l. The -mm option causes the macros to be interpreted so that only running text (that is, no text from macro lines) is output. 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."

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1).

NOTES

deroff is not a complete troff interpreter, so 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.

troff(1), nroff(1), and eqn(1) are not part of this UNIX system release.

NAME

deroff - remove nroff, troff, tbl and eqn constructs

SYNOPSIS

/usr/ucb/deroff [-w] *filename* . . .

DESCRIPTION

The deroff command reads each file in sequence and removes all nroff and troff command lines, backslash constructions, macro definitions, eqn constructs (between .EQ and .EN lines or between delimiters), and table descriptions and writes the remainder on the standard output. deroff follows chains of included files (.so and .nx commands); if a file has already been included, a .so is ignored and a .nx terminates execution. If no input file is given, deroff reads from the standard input file.

OPTIONS

-w Generate a word list, one word per line. A 'word' is a string of letters, digits, and apostrophes, beginning with a letter; apostrophes are removed. All other characters are ignored.

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1)

NOTES

deroff is not a complete troff interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

deroff does not work well with files that use .so to source in the standard macro package files.

NAME

devattr - lists device attributes

SYNOPSIS

devattr [-v] *device* [*attribute* [. . .]]

DESCRIPTION

devattr displays the values for a device's attributes. The display can be presented in two formats. Used without the -v option, only the attribute values are shown. Used with the -v option, the attributes are shown in an *attribute=value* format. When no attributes are given on the command line, all attributes for the specified device are displayed in alphabetical order by attribute name. If attributes are given on the command line, only those are shown and they are displayed in command line order.

The options and arguments for this command are:

-v	Specifies verbose format. Attribute values are displayed in an *attribute=value* format.
device	Defines the device whose attributes should be displayed. Can be the pathname of the device or the device alias.
attribute	Defines which attribute, or attributes, should be shown. Default is to show all attributes for a device. See the putdev(1M) manual page for a complete listing and description of available attributes.

ERRORS

The command will exit with one of the following values:

0 = successful completion of the task.

1 = command syntax incorrect, invalid option used, or internal error occurred.

2 = device table could not be opened for reading.

3 = requested device could not be found in the device table.

4 = requested attribute not defined for specified device.

FILES

/etc/device.tab

SEE ALSO

putdev(1M), devattr(3X), listdev(3X).

NAME

devfree - release devices from exclusive use

SYNOPSIS

devfree *key* [*device* [. . .]]

DESCRIPTION

devfree releases devices from exclusive use. Exclusive use is requested with the command devreserv.

When devfree is invoked with only the *key* argument, it releases all devices that have been reserved for that *key*. When called with *key* and *device* arguments, devfree releases the specified devices that have been reserved with that *key*.

The arguments for this command are:

key Designates the unique key on which the device was reserved.

device Defines device that this command will release from exclusive use. Can be the pathname of the device or the device alias.

ERRORS

The command will exit with one of the following values:

0 = successful completion of the task.

1 = command syntax incorrect, invalid option used, or internal error occurred.

2 = device table or device reservation table could not be opened for reading.

3 = reservation release could not be completely fulfilled because one or more of the devices was not reserved or was not reserved on the specified key.

FILES

/etc/device.tab
/etc/devlkfile

NOTES

The commands devreserv and devfree are used to manage the availability of devices on a system. These commands do not place any constraints on the access to the device. They serve only as a centralized bookkeeping point for those who wish to use them. Processes that do not use devreserv may concurrently use a device with a process that has reserved that device.

SEE ALSO

devfree(3X), devreserv(1), devreserv(3X), reservdev(3X)

NAME

devinfo - print device specific information

SYNOPSIS

/usr/lbin/devinfo -i | -p *special*

DESCRIPTION

The devinfo command is used to print device specific information about devices on standard out. The special argument must be a raw device (e.g. /dev/rdsk/?).

The options have the following effect:

-a prints the following information about devices:

```
Device name            Auto-configuration device identifier
```

-i prints the following information about disks:

```
Device name            Software version
Drive id number        Device blocks per cylinder
Device bytes per block Number of device slices with
                         a block size greater than zero
```

-p prints the following disk slice information:

```
Device name            Device major and minor numbers
                         (in hexadecimal)
Slice start block      Number of blocks allocated to
                         the slice
Slice flag             Slice tag
```

This command is used by various other commands to obtain device specific information for the making of file systems and determining slice information.

SEE ALSO

prtvtoc(1M), prtconf(1M)

NAME

devnm - device name

SYNOPSIS

/usr/sbin/devnm [*name* . . .]

DESCRIPTION

The devnm command identifies the special file associated with the mounted file system where the argument *name* resides. One or more *name*s can be specified.

This command is most commonly used by the brc command to construct a mount table entry for the root device.

EXAMPLE

The command:

```
/usr/sbin/devnm /usr
```

produces:

```
/dev/dsk/c1d0s2 /usr
```

if /usr is mounted on /dev/dsk/c1d0s2.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M), mnttab(4)

NAME

devreserv - reserve devices for exclusive use

SYNOPSIS

devreserv [*key* [*devicelist* [. . .]]]

DESCRIPTION

devreserv reserves devices for exclusive use. When the device is no longer required, use devfree to release it.

devreserv reserves at most one device per *devicelist*. Each list is searched in linear order until the first available device is found. If a device cannot be reserved from each list, the entire reservation fails.

When devreserv is invoked without arguments, it lists the devices that are currently reserved and shows to which key it was reserved. When devreserv is invoked with only the *key* argument, it lists the devices that are currently reserved to that key.

The arguments for this command are:

key Designates a unique key on which the device will be reserved. The key must be a positive integer.

devicelist Defines a list of devices that devreserv will search to find an available device. (The list must be formatted as a single argument to the shell.)

EXAMPLE

To reserve a floppy disk and a cartridge tape:

```
$ key=$$
$ echo "The current Process ID is equal to: $key"
The Current Process ID is equal to: 10658
$ devreserv $key ctape1
```

To list all devices currently reserved:

```
$ devreserv
disk1          2423
ctape1         10658
```

To list all devices currently reserved to a particular key:

```
$ devreserv $key
ctape1
```

ERRORS

The command will exit with one of the following values:

0 = successful completion of the task.
1 = command syntax incorrect, invalid option used, or internal error occurred.
2 = device table or device reservation table could not be opened for reading.
3 = device reservation request could not be fulfilled.

FILES

/etc/device.tab
/etc/devlkfile

NOTES

The commands `devreserv` and `devfree` are used to manage the availability of devices on a system. Their use is on a participatory basis and they do not place any constraints on the actual access to the device. They serve as a centralized bookkeeping point for those who wish to use them. To summarize, devices which have been reserved cannot be used by processes which utilize the device reservation functions until the reservation has been canceled. However, processes that do not use device reservation may use a device that has been reserved since such a process would not have checked for its reservation status.

SEE ALSO

`devfree`(1), `devfree`(3X), `devreserv`(3X), `reservdev`(3X)

NAME

df (generic) - report number of free disk blocks and files

SYNOPSIS

df [-F *FSType*] [-begklntV] [*current_options*] [-o *specific_options*] [*directory* | *special* | *resource...*]

DESCRIPTION

df prints the allocation portions of the generic superblock for mounted or unmounted file systems, directories or mounted resources. *directory* represents a valid directory name. If *directory* is specified df reports on the device that contains the *directory*. *special* represents a special device (for example, /dev/dsk/m328_c0d0s0). *resource* is an RFS/NFS resource name. If arguments to df are pathnames, df produces a report on the file system containing the named file.

current_options are options supported by the s5-specific module of df. Other FSTypes do not necessarily support these options. *specific_options* indicate suboptions specified in a comma-separated list of suboptions and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command. See df_*FSType*(1M) for details.

The options are:

-F Specify the *FSType* on which to operate. This is only needed if the file system is unmounted. The *FSType* should be specified here or be determinable from /etc/vfstab by matching the *mount_point*, *special*, or *resource* with an entry in the table.

-b Print only the number of kilobytes free.

-e Print only the number of files free.

-g Print the entire statvfs structure. Used only for mounted file systems. Can not be used with *current_options* or with the -o option. This option will override the -b, -e, -k, -n, and -t options.

-k Print allocation in kilobytes. This option should be invoked by itself because its output format is different from that of the other options.

-l Report on local file systems only. Used only for mounted file systems. Can not be used with *current_options* or with the -o option.

-n Print only the *FSType* name. Invoked with no arguments this option prints a list of mounted file system types. Used only for mounted file systems. Can not be used with *current_options* or with the -o option.

-t Print full listings with totals. This option will override the -b, -e, and -n options.

-V Echo the complete command line, but do not execute the command. The command line is generated by using the options and arguments provided by the user and adding to them information derived from /etc/mnttab or /etc/vfstab. This option should be used to verify and validate the command line.

`-o` Specify FSType-specific options.

If no arguments or options are specified, the free space on all local and remotely mounted file systems is printed.

NOTES

The `-F` option is intended for use with unmounted file systems.

This command may not be supported for all FSTypes.

FILES

`/dev/dsk/*`	
`/etc/mnttab`	mount table
`/etc/vfstab`	list of default parameters for each file system

SEE ALSO

`df_s5`(1M), `df_ufs`(1M), `mount`(1M), `statvfs`(2), `mnttab`(4), `vfstab`(4).

NAME

df (bsd) - report free disk space on file systems

SYNOPSIS

/usr/ucb/df [-a] [-i] [-t *type* | *File . . .]*

DESCRIPTION

df displays the amount of disk space occupied by currently mounted file systems, the amount of used and available space, and how much of the file system's total capacity has been used.

If arguments to df are path names, df produces a report on the file system containing the named file. Thus 'df .' shows the amount of space on the file system containing the current directory.

The following options are available:

-a Report on all filesystems

-i Report the number of used and free inodes

-t *type*
Report on filesystems of a given type (for example nfs or ufs)

EXAMPLE

A sample of output for df looks like:

```
df
Filesystem  kbytes  used  avail  capacity  Mounted on
sparky:/     7445    4714 1986   70%       /
sparky:/usr 42277   35291 2758   93%       /usr
```

FILES

/etc/mnttab	list of file systems currently mounted
/etc/vfstab	list of default parameters for each file system

SEE ALSO

du(1M), quot(1M), tunefs(1M), mnttab(4).

NAME

df (s5) - report number of free disk blocks and i-nodes for s5 file systems

SYNOPSIS

df [-F s5] [*generic_options*] [-f] [*directory* . . . | *special* . . .]

DESCRIPTION

generic_options are options supported by the generic df command.

The df command prints out the number of free blocks and free i-nodes in s5 file systems or directories by examining the counts kept in the super-blocks. The *special* device name (for example, /dev/dsk/*, where the value of * is machine-dependent) or mount point *directory* name (for example, /usr) must be specified. If *directory* is specified, the report presents information for the device that contains the directory.

The options are:

-F s5 Specifies the s5-FSType.

-f An actual count of the blocks in the free list is made, rather than taking the figure from the super-block.

NOTE

The -f option can be used with the -t, -b, and -e options. The -k option overrides the -f option.

FILES

/dev/dsk/*

SEE ALSO

generic df(1M)

NAME

df (ufs) - report free disk space on ufs file systems

SYNOPSIS

df [-F ufs] [*generic_options*] [-o i] [*directory* | *special*]

DESCRIPTION

generic_options are options supported by the generic df command.

df displays the amount of disk space occupied by ufs file systems, the amount of used and available space, and how much of the file system's total capacity has been used.

Note that the amount of space reported as used and available is less than the amount of space in the file system; this is because the system reserves a fraction of the space in the file system to allow its file system allocation routines to work well. The amount reserved is typically about 10%; this may be adjusted using tunefs(1M). When all the space on the file system except for this reserve is in use, only the super-user can allocate new files and data blocks to existing files. When the file system is overallocated in this way, df may report that the file system is more than 100% utilized.

The options are:

-F ufs
: Specifies the ufs-FSType.

-o
: Specify ufs file system specific options. The available option is:

  i
  : Report the number of used and free inodes. May not be used with *generic_options*.

NOTES

df calculates its results differently for mounted and unmounted file systems. For mounted systems the 10% reserved space mentioned above is included in the number of kilobytes used. For unmounted systems the 10% reservation is not included in the number of kilobytes used.

The -b and -e options override the -t option.

FILES

/etc/mnttab list of file systems currently mounted

SEE ALSO

generic df(1M), du(1M), quot(1M), tunefs(1M), mnttab(4)

NAME

`dfmounts` - display mounted NFS resource information

SYNOPSIS

`dfmounts` [`-F nfs`] [`-h`] [*server*...]

DESCRIPTION

`dfmounts` shows the local resources shared through Network File System, along with a list of clients that have mounted the resource. The `-F` flag may be omitted if NFS is the only file system type listed in the file `/etc/dfs/fstypes`.

The *server* option displays information about the resources mounted from each server, where *server* can be any system on the network. If no server is specified, then *server* is assumed to be the local system.

`dfmounts` without options displays all remote resources mounted on the local system, regardless of file system type.

The output of `dfmounts` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server pathname clients . . .

where

resource	Specifies the resource name that must be given to the `mount`(1M) command.
server	Specifies the system from which the resource was mounted.
pathname	Specifies the pathname that must be given to the `share`(1M) command.
clients	A comma-separated list of systems that have mounted the resource.

FILES

`/etc/dfs/fstypes`

SEE ALSO

`mount`(1M), `share`(1M), `unshare`(1M).

NAME

dfmounts - display mounted RFS resource information

SYNOPSIS

dfmounts [-F rfs] [-h] [*resource_name* . . .]

DESCRIPTION

dfmounts shows the local resources shared through Remote File Sharing, along with a list of clients that have mounted the resource. The -F flag may be omitted if rfs is the first file system type listed in the file /etc/dfs/fstypes.

The output of *dfmounts* consists of an optional header line (suppressed with the -h flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server path clients . . .

where

resource	Specifies the resource name that must be given to the mount(1M) command.
server	Specifies the system from which the resource was mounted.
path	Specifies the full pathname that must be given to the share(1M) command.
clients	A comma-separated list of systems that have mounted the resource.

A field may be null. Each null field is indicated by a hyphen (-) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

Only a privileged user can execute this command.

FILES

/etc/dfs/fstypes

SEE ALSO

dfmounts(1M), share(1M), unshare(1M), fumount(1M), mount(1M)

NAME

`dfmounts` - display mounted resource information

SYNOPSIS

`dfmounts` [`-F` *fstype*] [`-h`] [`-o` *specific_options*] [*restriction . . .*]

DESCRIPTION

`dfmounts` shows the local resources shared through a distributed file system *fstype* along with a list of clients that have the resource mounted. If *restriction* is not specified, `dfmounts` displays remote resources mounted on the local system. *Specific_options* as well as the availability and semantics of *restriction* are specific to particular distributed file system types. See `dfmount_`*FSType*(1M) for details.

If `dfmounts` is entered without arguments, all remote resources currently mounted on the local system are displayed, regardless of file system type.

The output of `dfmounts` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server pathname clients

where

resource	Specifies the resource name that must be given to the `mount`(1M) command.
server	Specifies the system from which the resource was mounted.
pathname	Specifies the pathname that must be given to the `share`(1M) command.
clients	Lists the systems, comma-separated, by which the resource was mounted. Clients are listed in the form *domain.*, *domain.system*, or *system*, depending on the file system type.

A field may be null. Each null field is indicated by a hyphen (-) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

Fields with whitespace are enclosed in quotation marks (`" "`).

NOTES

`dfmounts` may not indicate the correct state if you mount a single resource on more than one directory.

FILES

`/etc/dfs/fstypes`

SEE ALSO

`dfmount_nfs`(1M), `dfmount_rfs`(1M), `dfshares`(1M), `mount`(1M), `share`(1M), `unshare`(1M).

NAME

dfshares - list available NFS resources from remote systems

SYNOPSIS

dfshares [-F nfs] [-h] [*server*...]

DESCRIPTION

dfshares provides information about resources available to the host through Network File System. The -F flag may be omitted if NFS is the first file system type listed in the file /etc/dfs/fstypes.

The query may be restricted to the output of resources available from one or more servers.

The *server* option displays information about the resources shared by each server, where *server* can be any system on the network. If no server is specified, then *server* is assumed to be the local system.

dfshares without arguments displays all resources shared on the local system, regardless of file system type.

The output of dfshares consists of an optional header line (suppressed with the -h flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server access transport

where

resource	Specifies the resource name that must be given to the mount(1M) command.
server	Specifies the system that is making the resource available.
access	Specifies the access permissions granted to the client systems; however, dfshares cannot determine this information for an NFS resource and populates the field with a hyphen (-).
transport	Specifies the transport provider over which the *resource* is shared; however, dfshares cannot determine this information for an NFS resource and populates the field with a hyphen (-).

FILES

/etc/dfs/fstypes

SEE ALSO

share(1M), unshare(1M), mount(1M)

NAME

`dfshares` - list available RFS resources from remote systems

SYNOPSIS

`dfshares` [`-F rfs`] [`-h`] [*server*...]

DESCRIPTION

`dfshares` provides information about resources available to the host through Remote File Sharing. The `-F` flag may be omitted if `rfs` is the first file system type listed in the file `/etc/dfs/fstypes`.

The query may be restricted to the output of resources available from one or more servers. If no *server* is specified, all resources in the host's domain are displayed. A *server* may be given in the following form:

system	Specifies a system in the host's domain.
domain.	Specifies all systems in *domain*.
domain.system	Specifies *system* in *domain*.

The output of `dfshares` consists of an optional header line (suppressed with the `-h` flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server access transport description

where

resource	Specifies the resource name that must be given to the `mount`(1M) command.
server	Specifies the system that is making the resource available.
access	Specifies the access permissions granted to the client systems, either `ro` (for read-only) or `rw` (for read and write).
transport	Specifies the transport provider over which the *resource* is shared.
description	Describes the resource.

A field may be null. Each null field is indicated by a hyphen (-) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

ERRORS

If your host machine cannot contact the domain name server, or the argument specified is syntactically incorrect, an error message is sent to standard error.

FILES

`/etc/dfs/fstypes`

SEE ALSO

`share`(1M), `unshare`(1M), `mount`(1M)

NAME

dfshares - list available resources from remote or local systems

SYNOPSIS

dfshares [-F *fstype*] [-h] [-o *specific_options*] [*server*...]

DESCRIPTION

dfshares provides information about resources available to the host through a distributed file system of type *fstype*. *Specific_options* as well as the semantics of *server* are specific to particular distributed file systems. See dfshare_*FSType*(1M) for details.

If dfshares is entered without arguments, all resources currently shared on the local system are displayed, regardless of file system type.

The output of dfshares consists of an optional header line (suppressed with the -h flag) followed by a list of lines containing whitespace-separated fields. For each resource, the fields are:

resource server access transport description ...

where

resource	Specifies the resource name that must be given to the mount(1M) command.
server	Specifies the name of the system that is making the resource available.
access	Specifies the access permissions granted to the client systems, either ro (for read-only) or rw (for read/write). If dfshares cannot determine access permissions, a hyphen (-) is displayed.
transport	Specifies the transport provider over which the *resource* is shared.
description	Describes the resource.

A field may be null. Each null field is indicated by a hyphen (-) unless the remainder of the fields on the line are also null. In this case, it may be omitted.

FILES

/etc/dfs/fstypes

SEE ALSO

dfmounts(1M), dfshare_nfs(1M), dfshare_rfs(1M), mount(1M), share(1M), unshare(1M).

NAME

diff - differential file comparator

SYNOPSIS

diff [-bitw] [-c | -e | -f | -h | -n] *filename1 filename2*
diff [-bitw] [-C *number*] *filename1 filename2*
diff [-bitw] [-D *string*] *filename1 filename2*
diff [-bitw] [-c | -e | -f | -h | -n] [-l] [-r] [-s] [-S *name*] *directory1 directory2*

DESCRIPTION

diff tells what lines must be changed in two files to bring them into agreement. If *filename1* (*filename2*) is -, the standard input is used. If *filename1* (*filename2*) is a directory, then a file in that directory with the name *filename2* (*filename1*) 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 commands to convert *filename1* into *filename2*. The numbers after the letters pertain to *filename2*. In fact, by exchanging a for d and reading backward one may ascertain equally how to convert *filename2* into *filename1*. As in ed, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

-b Ignores trailing blanks (spaces and tabs) and treats other strings of blanks as equivalent.

-i Ignores the case of letters; for example, 'A' will compare equal to 'a'.

-t Expands TAB characters in output lines. Normal or -c output adds character(s) to the front of each line that may adversely affect the indentation of the original source lines and make the output lines difficult to interpret. This option will preserve the original source's indentation.

-w Ignores all blanks (SPACE and TAB characters) and treats all other strings of blanks as equivalent; for example, 'if (a == b)' will compare equal to 'if(a==b)'.

The following options are mutually exclusive:

-c Produces a listing of differences with three lines of context. With this option output format is modified slightly: output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen *'s. The lines removed from *filename1* are marked with '—'; those added to *filename2* are marked '+'. Lines that are changed from one file to the other are marked in both files with '!'.

-C *number*
Produces a listing of differences identical to that produced by -c with *number* lines of context.

`-e` Produces a script of *a*, *c*, and *d* commands for the editor `ed`, which will recreate *filename2* from *filename1*. In connection with `-e`, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version `ed` scripts ($2,$3,...) made by `diff` need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Except in rare circumstances, `diff` finds a smallest sufficient set of file differences.

`-f` Produces a similar script, not useful with `ed`, in the opposite order.

`-h` Does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options `-e` and `-f` are unavailable with `-h`.

`-n` Produces a script similar to `-e`, but in the opposite order and with a count of changed lines on each insert or delete command.

`-D` *string*
Creates a merged version of *filename1* and *filename2* with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *filename1*, while defining *string* will yield *filename2*.

The following options are used for comparing directories:

`-l` Produce output in long format. Before the `diff`, each text file is piped through `pr`(1) to paginate it. Other differences are remembered and summarized after all text file differences are reported.

`-r` Applies `diff` recursively to common subdirectories encountered.

`-s` Reports files that are the identical; these would not otherwise be mentioned.

`-S` *name*
Starts a directory `diff` in the middle, beginning with the file *name*.

FILES

```
/tmp/d?????
/usr/lib/diffh for -h
/usr/bin/pr
```

SEE ALSO

`bdiff`(1), `cmp`(1), `comm`(1), `ed`(1), `pr`(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

NOTES

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.

NAME

`diff3` - 3-way differential file comparison

SYNOPSIS

`diff3` [`-exEX3`] *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.

`-e` Produce a script for the editor `ed`(1) that will incorporate into *file1* all changes between *file2* and *file3*, that is, the changes that normally would be flagged `====` and `====3`.

`-x` Produce a script to incorporate only changes flagged `====`.

`-3` Produce a script to incorporate only changes flagged `====3`.

`-E` Produce a script that will incorporate all changes between *file2* and *file3*, but treat overlapping changes (that is, changes that would be flagged with `====` in the normal listing) differently. The overlapping lines from both files will be inserted by the edit script, bracketed by `<<<<<<` and `>>>>>>` lines.

`-X` Produce a script that will incorporate only changes flagged `====`, but treat these changes in the manner of the `-E` option.

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)

NOTES

Text lines that consist of a single . will defeat `-e`.
Files longer than 64K bytes will not work.

NAME

diffmk - mark differences between versions of a troff input file

SYNOPSIS

/usr/ucb/diffmk *oldfile newfile markedfile*

DESCRIPTION

The diffmk command compares two versions of a file and creates a third version that includes "change mark" (.mc) commands for nroff and troff. *oldfile* and *newfile* are the old and new versions of the file. diffmk generates *markedfile*, which, contains the text from *newfile* with troff(1) "change mark" requests (.mc) inserted where *newfile* differs from *oldfile*. When *markedfile* is formatted, changed or inserted text is shown by a | at the right margin of each line. The position of deleted text is shown by a single *.

diffmk can also be used in conjunction with the proper troff requests to produce program listings with marked changes. In the following command line:

```
diffmk old.c new.c marked.c ; nroff reqs marked.c | pr
```

the file reqs contains the following troff requests:

```
.pl 1
.ll 77
.nf
.eo
.nh
```

which eliminate page breaks, adjust the line length, set no-fill mode, ignore escape characters, and turn off hyphenation, respectively.

If the characters | and * are inappropriate, you might run *markedfile* through sed to globally change them.

SEE ALSO

diff(1), nroff(1), sed(1), troff(1).

NOTES

Aesthetic considerations may dictate manual adjustment of some output. File differences involving only formatting requests may produce undesirable output, that is, replacing .sp by .sp 2 will produce a "change mark" on the preceding or following line of output.

NAME

`dig` - send domain name query packets to name servers

SYNOPSIS

`dig` [*@server*] *domain* [*query-type*] [*query-class*\ ] [*+query-option*] [*-<dig-option>*] [*%comment*]

DESCRIPTION

The Domain Information Groper, `dig`, is a flexible command line tool which can be used to gather information from the Domain Name System servers. The `dig` tool has two modes:

- simple interactive mode which makes a single query
- batch which executes a query for each in a list of several query lines

All query options are accessible from the command line.

The simple use of `dig` takes the following form:

`dig` *@server domain query-type query-class*

where:

server may be either a domain name or a dot-notation Internet address. If this optional field is omitted, `dig` will attempt to use the default name server for your machine.

As an option, the user may set the environment variable LOCALRES to name a file which is to be used instead of `/etc/resolv.conf`; LOCALRES is specific to the `dig` resolver and is not referenced by the standard resolver. If the LOCALRES variable is not set, or if the file is not readable, then `/etc/resolv.conf` will be used.

domain

is the domain name for which you are requesting information. See "OPTIONS" [`-x`] for a convenient way to specify an inverse address query.

query-type

is the type of information [Domain Name System (DNS) query type] that you are requesting. If omitted, the default is "`a`" (`T_A` = network-address). The following types will be recognized:

`a`	`T_A`	network address
`any`	`T_ANY`	all/any information about specified domain
`mx`	`T_MX`	mail exchanger for the domain
`ns`	`T_NS`	name servers
`soa`	`T_SOA`	zone of authority record
`hinfo`	`T_HINFO`	host information
`axfr`	`T_AXFR`	zone transfer (must ask an authoritative server)
`txt`	`T_TXT`	arbitrary number of strings (not yet supported by BIND)

(See RFC 1035 for the complete list.)

query-class

is the network class requested in the query. If omitted, the default is "`in`" (`C_IN` = Internet). The following classes are recognized:

in	C_IN	Internet class domain
any	C_ANY	all/any class information

(See RFC 1035 for the complete list.)

OPTIONS

`%ignored_comment`

"%" is used to include an argument which should not be parsed. This may be useful if running `dig` in batch mode. Instead of resolving every @server-domain-name in a list of queries, you can avoid the overhead of doing so, but still have the domain name on the command line as a reference. For example:

```
dig @128.9.0.32 %venera.isi.edu mx isi.edu
```

`-<`*dig_option*`>`

"-" is used to specify an option which affects the operation of `dig`. The following options are currently available (although not guaranteed to be useful):

`-x` *dot-notation-address*

A convenient form to specify inverse address mapping: instead of `dig 32.0.9.128.in-addr.arpa` one can simply enter `dig -x 128.9.0.32`.

`-f`*file* A file for `dig` batch mode: the *file* contains a list of query specifications (i.e., `dig` command lines) which are to be executed sequentially. Lines beginning with ';', '#', or '\n' are ignored. Other options may still appear on the command line and will be in effect for each batch query.

`-T` *time*

Time in seconds between start of successive queries when running in batch mode; can be used to keep two or more batch `dig` commands running roughly in sync. The default value is zero.

`-p` *port*

Port Number: query a name server listening to a non-standard Port Number; the default value is 53.

`-P[`*ping-string*`]`

After the query returns, execute a `ping`(1M) command for a response time comparison. This rather unelegantly makes a call to the shell.

If the option `-P`*ping-string* is present, it will replace `ping -s` in the shell command.

The last three lines of statistics will be printed for the command:

```
ping -s server_name 56 3
```

`-t` *query-type*

Specify the type of query: may specify either an integer value to be included in the type field or use the abbreviated mnemonic as discussed above (i.e., `mx` = `T_MX`).

`-c` *query-class*
: Specify the class of query: may specify either an integer value to be included in the class field or use the abbreviated mnemonic as discussed above (i.e., `in` = `C_IN`).

`-envsav`
: This flag specifies that the `dig` environment (defaults, print options, etc.), - after all of the arguments are parsed - should be saved to a file to become the default environment. This will be useful if you do not like the standard set of defaults and if do not want to include a large number of options whenever `dig` is used. The environment will consist of `resolver` state variable flags, timeout values, and numbers of allowable retry attempts, as well as of the flags detailing the `dig` output contents (see below). If the shell environment variable LOCALDEF is set to the name of a file, this is where the default `dig` environment will be saved. If not, the file `DiG.env` will be created in the current working directory.

Whenever `dig` is executed, it will look for `./DiG.env` or for the file specified by the shell environment variable LOCALDEF. If such a file exists and if it is readable, then the environment will be restored from this file before any arguments are parsed.

`-envset`
: This flag will affect batch query runs only. When `-envset` is specified on a line in a `dig` batch file, the `dig` environment following the arguments will be parsed; then these values will be used as the default environment for the duration of the batch file (or until the system finds another command line which specifies "`-envset`").

`-[no]stick`
: This flag will affect batch query runs only: it specifies that the `dig` environment (as read initially or set by the `-envset` switch) is to be restored before each query (line) in a `dig` batch file.

The default value "`-nostick`" means that the `dig` environment should not stick; hence options specified on a single line in a `dig` batch file will remain in effect for subsequent lines (i.e., they will not be restored to the "sticky" default).

+<*query-option*>
: "+" is used to specify an option to be changed in the query packet or to change some `dig` output specifics. Many of these are the same parameters accepted by `nslookup`(1M). If an option requires a parameter, the format will be as follows:

    `+keyword[=value]`

Most keywords can be abbreviated. Parsing of the "+" options is very simplistic — a value must not be separated from its keyword by white space.

The following keywords are currently available:

Keyword	Abbrev.	Meaning [default]
`[no]debug`	(deb)	turn on/off debugging mode [deb]

[no]d2		turn on/off extra debugging mode [nod2]
[no]recurse	(rec)	use/don't use recursive lookup [rec]
retry=#	(ret)	set number of retries to # [4]
time=#	(ti)	set timeout length to # seconds [4]
[no]ko		keep open option (implies vc) [noko]
[no]vc		use/don't use virtual circuit [novc]
[no]defname	(def)	use/don't use default domain name [def]
[no]search	(sea)	use/don't use domain search list [sea]
domain=*NAME*	(do)	set default domain name to *NAME*
[no]ignore	(i)	ignore/don't ignore truncation errors [noi]
[no]primary	(pr)	use/don't use primary server [nopr]
[no]aaonly	(aa)	authoritative query only flag [noaa]
[no]sort	(sor)	sort resource records [nosor]
[no]cmd		echo parsed arguments [cmd]
[no]stats	(st)	print query statistics (RTT, etc) [st]
[no]Header	(H)	print basic header [H]
[no]header	(he)	print header flags [he]
[no]ttlid	(tt)	print TTLs [tt]
[no]cl		print class info [nocl]
[no]qr		print outgoing query [noqr]
[no]reply	(rep)	print reply [rep]
[no]ques	(qu)	print question section [qu]
[no]answer	(an)	print answer section [an]
[no]author	(au)	print authoritative section [au]
[no]addit	(ad)	print additional section [ad]
pfdef		set to default print flags
pfmin		set to minimal default print flags
pfset=#		set print flags to # (# can be hexadecimal/ octal/decimal)
pfand=#		bitwise "and" print flags with #
pfor=#		bitwise "or" print flags with #

The retry and time options will affect the retransmission strategy used by the `resolver` library when sending datagram queries. The algorithm is as follows:

```
for i = 0 to retry-1
    for j = 1 to num_servers
                send_query
                wait((time * (2**i)) / num_servers)
    end
end
```

(Note: `dig` always uses a value of 1 for `num_servers`.)

The `pfset`, `pfand`, and `pfor` options have been included to make the manipulation of the various print options less tedious. Below is a list of the currently defined meanings for the various print flag bits which may be combined (ANDed) to achieve various output formats.

PRF_STATS	0x0001	RTT, query and server host, date, message size

PRF_CLASS	0x0004	Resource record class information
PRF_CMD	0x0008	echo the dig command line
PRF_QUES	0x0010	questions section
PRF_ANS	0x0020	answers section
PRF_AUTH	0x0040	authoritative section
PRF_ADD	0x0080	additional records section
PRF_HEAD1	0x0100	RR section headers & counts
PRF_HEAD2	0x0200	packet header flags
PRF_TTLID	0x0400	Resource record Time-to-Live (ttl)
PRF_HEADX	0x0800	basic header
PRF_QUERY	0x1000	outgoing query packet
PRF_REPLY	0x2000	reply packet
PRF_SORT	0x8000	sort various response sections
PRF_DEF	0x2ff9	default dig settings
PRF_ZONE	0x24f9	default setting for zone transfer
PRF_MIN	0xa930	minimalistic dig settings for (future) automated server testing

When setting the print options and if you want to see information other than the statistics, you should choose to examine the outgoing (0x1000) packet type, the incoming (0x2000) packet type, or both packet types, as well as the specific sections of the packet(s) of particular interest to you.

DETAILS

The dig tool requires a slightly modified version of the BIND resolver(3) library to gather count and time statistics. Otherwise, it is a straight forward (but not pretty) effort of parsing arguments and setting appropriate parameters. The dig tool uses resolver routines res_init(), res_mkquery(), res_send(); it also accesses the _res structure.

It is possible to compile dig with the standard resolver library, but this procedure will change the dig output format, make the dig print options meaningless, and not gather RTT and packet count statistics.

FILES

/etc/resolv.conf	initial domain name and name server addresses
./DiG.env	default save file for default options

ENVIRONMENT

LOCALRES	file to use in place of /etc/resolv.conf
LOCALDEF	default environment file

NOTES

If a domain name is specified, this will be resolved using the (DNS) resolver. If your system does not support DNS, you may *have* to specify a dot-notation address. Alternatively, if there is a server at your disposal somewhere, all that is required is that /etc/resolv.conf be present and indicate where the default name servers reside, so that server itself can be resolved. [See resolv.conf(4) for information on /etc/resolv.conf.]

"any" can be used to specify a class and/or a type of query: dig will parse the first occurrence of "any" to mean query-type = T_ANY. To specify query-class = C_ANY you must either specify "any" twice, or set the query-class using the "-c" option (see "OPTIONS").

LOCALDEF is specific to the dig resolver and will not affect the operation of the standard resolver library.

CAUTION

Changing /etc/resolv.conf will affect the standard resolver library and potentially several programs which use it.

BUGS

dig has a serious case of "creeping featurism" - the result of considering several potential uses during its development. It would probably benefit from a rigorous diet. Similarly, the print flags and granularity of the items they specify make their rather *ad hoc* genesis evident.

dig does not exit consistently with an appropriate status when a problem occurs somewhere in the resolver.

Most of the common exit cases are handled. This becomes particularly annoying when running in batch mode. If dig exits abnormally - and is not caught - the entire batch will abort; when such an event is trapped, dig simply will continue with the next query.

SEE ALSO

nslookup(1M), resolver(3N), resolv.conf(4), named(1M), ping(1M).

NAME

`dinit` - disk initializer

SYNOPSIS

`/etc/dinit` [`-afqnrRsTx`] [`-v` *lvl*] [`-d` *desc*] [`-b` *boot file*] [`-t` *file*] *type* `/dev/rdsk/`*prefix*`_c`*X*`d`*Y*`s7`

DESCRIPTION

The `dinit` program can be used to initialize specified disk *types*. The *type* must be a file in the directory `/etc/dskdefs`. See `mvme328`(7), `scsi1x7`(7), and `mvme323`(7) for lists of supported devices and their associated *type* values.

For disk types with bad track handling, the alternate track numbers will be taken from the file `/etc/dskdefs/`*type*. Note that although sector slip format type is supported, the disk driver does not use this for dynamic error correction. Because the sector slip format decreases the amount of usable space on the disk, its use is discouraged. Use the `-n` option to add bad spots to a disk format.

The `/dev/rdsk/`*prefix*`_c`*X*`d`*Y*`s7` file must be slice seven of the character special file of the device. *prefix* is the device type, `c`*X* is the controller number of that device, `d`*Y* is the device number for the controller, and `s`*Z* is the slice number. See `intro`(7) for complete lists of controllers, devices, and slices.

The following options are provided for `dinit`:

`-r` Read bad spot list from disk and print it on `stdout`.

`-a` Use with `-r` option to print alternates.

`-x` Use with `-r` option to print in hexadecimal.

`-n` Add a new bad spot (see below).

`-s` Skip saving and restoring of data when adding a new spot.

`-f` Format disk. When formatting an unformatted disk, two read errors appear on the screen. These errors occur because the controller is trying to read configuration information from the disk. The messages can be ignored; the disk will be formatted as requested.

`-q` Format quickly. This option may be used in conjunction with the `-f` option. `dinit` will perform all of the work involved in formatting the disk except actually (physically) formatting it. This can be useful when a disk has been formatted in the past but the disk identification information has been destroyed.

`-R` Read manufacturer's defect list and print it on `stdout`.

`-d` *desc* Use *desc* as description string in sector 0.

`-b` *boot-file*
Use *boot file* (`a.out` format) as the bootloader program. The standard bootloader is `/usr/lib/boot`.

`-t` *file* Take bad track numbers from *file*, instead of interactively. This option only works when the `-f` option is used.

`-T` *file* Take bad track numbers in track format; default for SCSI disks is block number.

`-v` *lvl* Format the disk and perform disk surface verification using *lvl* test patterns. *lvl* must be from 0 to 4. A value of *lvl=0* is identical to the `-f` option.

Unless the `-f` option is given, `dinit` will use the parameters and bad spot information existing on the disk. Therefore, it is not necessary to re-enter bad track numbers on subsequent uses of `dinit` on a disk. This is useful for changing the bootloader, description string, and so on. To change other disk parameters, refer to `ddefs`(1M).

The disk driver may occasionally report an I/O error for a bad spot not mapped out in the original format to the system console. In such cases the new bad spot may be mapped without formatting the entire disk using the `-n` option only.

The the error message from the disk driver will contain the bad block number of the new bad spot. The bad block number should be entered when prompted by `dinit`.

`dinit` will attempt to save the data from the new bad track unless the `-s` option is specified. If attempting to save data, expect I/O errors from the disk driver while `dinit` is executing. This process may not be able to recover all the data because there may be unreadable sectors on the new bad track. If the new bad track is within a vital area on the disk, the disk may become unbootable or there may be extensive file system damage. Examples of vital areas include the configuration area (sector 0), bootloader (sectors 1 through 12), slice table (usually sector 20), and super blocks (location depends on disk slicing).

FILES

`/etc/dskdefs`	disk definition file
`/usr/lib/boot`	bootstrap program

SEE ALSO

`ddefs`(1M), `fmthard`(1M), `mvme323`(7), `mvme328`(7), `scsi1x7`(7).

NOTES

Currently, only the MVME323 disk controller supports the `-R` option.

`dinit` must be executed over the slice representing the entire raw device (that is, `slice 7`).

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. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names 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* columns. Multibyte characters spanning over the specified width are not displayed. The default width is 72.

INTERNATIONAL FUNCTIONS

Characters from supplementary code sets can be used for directory names, and the specified directory can contain files with names using supplementary characters.

SEE ALSO

cmp(1), diff(1)

NAME

dis - object code disassembler

SYNOPSIS

dis [-o] [-V] [-L] [-s] [-d *sec*] [-D *sec*] [-F *function*] [-t *sec*] [-l *string*] *file* . . .

DESCRIPTION

The dis command produces an assembly language listing of *file*, which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

The following *options* are interpreted by the disassembler and may be specified in any order.

-d *sec* Disassemble the named section as data, printing the offset of the data from the beginning of the section.

-D *sec* Disassemble the named section as data, printing the actual address of the data.

-F *function* Disassemble only the named function in each object file specified on the command line. The -F option may be specified multiple times on the command line.

-L Lookup source labels for subsequent printing. This option works only if the file was compiled with additional debugging information (for example, the -g option of cc).

-l *string* Disassemble the archive file specified by *string*. For example, one would issue the command dis -l x -l z to disassemble libx.a and libz.a, which are assumed to be in *LIBDIR*.

-o Print numbers in octal. The default is hexadecimal.

-s Perform symbolic disassembly where possible. Symbolic disassembly output will appear on the line following the instruction. Symbol names will be printed using C syntax.

-t *sec* Disassemble the named section as text.

-V Print, on standard error, the version number of the disassembler being executed.

If the -d, -D or -t options are specified, only those named sections from each user-supplied file name will be disassembled. Otherwise, all sections containing text will be disassembled.

On output, a number enclosed in brackets at the beginning of a line, such as [5], indicates that the break-pointable line number starts with the following instruction. These line numbers will be printed only if the file was compiled with additional debugging information [for example, the -g option of cc]. An expression such as <40> in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control will be transferred. A function name will appear in the first column, followed by ( ) if the object file contains a symbol table.

FILES

LIBDIR usually `/usr/ccs/lib`

SEE ALSO

`as`(1), `cc`(1), `ld`(1), `a.out`(4)

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NOTES

Since the `-da` option did not adhere to the command syntax rules, it has been replaced by `-D`.

At this time, symbolic disassembly does not take advantage of additional information available if the file is compiled with the `-g` option.

NAME

diskusg - generate disk accounting data by user ID

SYNOPSIS

/usr/lib/acct/diskusg [*options*] [*files*]

DESCRIPTION

diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. diskusg output lines on the standard output, one per user, in the following format: *uid login #blocks*

where

uid	the numerical user ID of the user.
login	the login name of the user; and
#blocks	the total number of disk blocks allocated to this user.

diskusg normally reads only the inodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

diskusg recognizes the following options:

-s	the input data is already in diskusg output format. diskusg combines all lines for a single user into a single line.
-v	verbose. Print a list on standard error of all files that are charged to no one.
-i *fnmlist*	ignore the data on those file systems whose file system name is in *fnmlist*. *fnmlist* is a list of file system names separated by commas or enclosed within quotes. diskusg compares each name in this list with the file system name stored in the volume ID [see labelit(1M)].
-p *file*	use *file* as the name of the password file to generate login names. /etc/passwd is used by default.
-u *file*	write records to *file* of files that are charged to no one. Records consist of the special file name, the inode number, and the user ID.

The output of diskusg is normally the input to acctdisk [see acct(1M)] which generates total accounting records that can be merged with other accounting records. diskusg is normally run in dodisk [see acctsh(1M)].

EXAMPLES

The following will generate daily disk accounting information for root on /dev/rdsk/m328_c1d0s0:

```
diskusg /dev/rdsk/m328_c1d0s0 | acctdisk > disktacct
```

FILES

/etc/passwd	used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M), acct(4)

NOTES

diskusg only works for S5 file systems and the raw device special files associated with them. acctdusg (see acct(1M)) works for all file systems, but is slower than diskusg.

NAME

`dispadmin` - process scheduler administration

SYNOPSIS

```
dispadmin -l
dispadmin -c class -g [-r res]
dispadmin -c class -s file
```

DESCRIPTION

The `dispadmin` command displays or changes process scheduler parameters while the system is running.

The `-l` option lists the scheduler classes currently configured in the system.

The `-c` option specifies the class whose parameters are to be displayed or changed. Valid *class* values are `RT` for the real-time class and `TS` for the time-sharing class.

The `-g` option gets the parameters for the specified class and writes them to the standard output. Parameters for the real-time class are described on `rt_dptbl`(4). Parameters for the time-sharing class are described on `ts_dptbl`(4).

When using the `-g` option you may also use the `-r` option to specify a resolution to be used for outputting the time quantum values. If no resolution is specified, time quantum values are in milliseconds. If *res* is specified it must be a positive integer between 1 and 1000000000 inclusive, and the resolution used is the reciprocal of *res* in seconds. For example, a *res* value of 10 yields time quantum values expressed in tenths of a second; a *res* value of 1000000 yields time quantum values expressed in microseconds. If the time quantum cannot be expressed as an integer in the specified resolution, it is rounded up to the next integral multiple of the specified resolution.

The `-s` option sets scheduler parameters for the specified class using the values in *file*. These values overwrite the current values in memory—they become the parameters that control scheduling of processes in the specified class. The values in *file* must be in the format output by the `-g` option. Moreover, the values must describe a table that is the same size (has same number of priority levels) as the table being overwritten. Super-user privileges are required in order to use the `-s` option.

The `-g` and `-s` options are mutually exclusive: you may not retrieve the table at the same time you are overwriting it.

`dispadmin` does some limited sanity checking on the values supplied in *file* to verify that they are within their required bounds. The sanity checking, however, does not attempt to analyze the effect that the new values have on the performance of the system. Inappropriate values can have a dramatic negative effect on system performance.

EXAMPLES

The following command retrieves the current scheduler parameters for the real-time class from kernel memory and writes them to the standard output. Time quantum values are in microseconds.

```
dispadmin -c RT -g -r 1000000
```

The following command overwrites the current scheduler parameters for the real-time class with the values specified in `rt.config`.

```
dispadmin -c RT -s rt.config
```

The following command retrieves the current scheduler parameters for the time-sharing class from kernel memory and writes them to the standard output. Time quantum values are in nanoseconds.

```
dispadmin -c TS -g -r 1000000000
```

The following command overwrites the current scheduler parameters for the time-sharing class with the values specified in `ts.config`.

```
dispadmin -c TS -s ts.config
```

DIAGNOSTICS

`dispadmin` prints an appropriate diagnostic message if it fails to overwrite the current scheduler parameters due to lack of required permissions or a problem with the specified input file.

SEE ALSO

`priocntl`(1), `priocntl`(2), rt_dptbl(4), ts_dptbl(4)

NAME

`dispgid` - displays a list of all valid group names

SYNOPSIS

`dispgid`

DESCRIPTION

`dispgid` displays a list of all group names on the system (one group per line).

EXIT CODES

0 = Successful execution
1 = Cannot read the group file

NAME

`dispuid` - displays a list of all valid user names

SYNOPSIS

`dispuid`

DESCRIPTION

`dispuid` displays a list of all user names on the system (one line per name).

EXIT CODES

0 = Successful execution
1 = Cannot read the password file

NAME

`dl` - Common Environment download utility

SYNOPSIS

`dl` [*-options*] [*file*] *dst_cpu*

DESCRIPTION

`dl` performs the download and/or execution of object code to a remote CPU executing the Common Environment. The remote CPU is indicated by *dst_cpu*. The COFF/ELF format object file read during the download is indicated by *file*. The default object file format is COFF.

The following options direct the action of `dl`:

`-l` Directs `dl` only to download the object code from *file* to the remote Common Environment.

`-x` *section*
Directs `dl` to exclude the section *section* from the download. This option may be repeated for each section to be excluded. *Section* is the name of the section to be excluded. This option is applicable only to COFF format files.

`-E` Specifies that the object file format is ELF.

`-e` Directs `dl` only to perform the execution of the object code at the address contained in the entry_point field of *file*.

`-a` *address,argument*
Directs `dl` to perform execution of the object code at the address given by *address*. The execution is performed as a function call from the Common Environment. *Argument* is supplied as an unsigned long integer to the function. *Argument* must be an integer in octal, decimal or hexadecimal.

`-s` *symbol,argument*
Directs `dl` to perform execution of the object code at the address of *symbol* found in *file*. The execution is performed as a function call from the Common Environment and *argument* is supplied as an unsigned long integer to the function. *Argument* must be an integer in octal, decimal or hexadecimal.

If no options are supplied to `dl`, then *file* must be supplied and `dl` downloads the object code and performs execution at the entry_point found in *file*.

EXAMPLES

`dl a.out 2`
Download and execute the file `a.out`.

`dl -l a.out 2`
Download the object from `a.out` without executing.

`dl -e a.out 2`
Execute the object at the entry_point in `a.out`.

`dl -a 0xfff09000,1 2`
Execute object at the address 0xfff09000 as a function call with the argument of 1.

`dl -x.bss a.out 2`
Download all sections from `a.out` except the .bss section and then execute at the entrypoint in `a.out`.

`dl -s set_pr,0 a.out 2`
Execute object at the address of the "set_pr" function as a function call with the argument of 0. Set_pr's address is found in `a.out`.

NAME

dname - print Remote File Sharing domain and network names

SYNOPSIS

dname [-D *domain*] [-N *netspeclist*] [-dna]

DESCRIPTION

dname prints or defines a host's Remote File Sharing domain name or the network(s) used by Remote File Sharing as transport provider(s). When used with d, n, or a options, dname can be run by any user to print the domain name, transport provider name(s), or both. Only a user with root permission can use the -D *domain* option to set the domain name for the host or -N *netspeclist* to set the network specification used for Remote File Sharing. *netspeclist* is a comma-separated list of transport providers (*tp1,tp2,. . .*). The value of each transport provider is the network device name, relative to the */dev* directory. For example, the STARLAN NETWORK uses starlan.

domain must consist of no more than 14 characters, consisting of any combination of letters (upper and lower case), digits, hyphens (-), and underscores (_).

When dname is used to change a domain name, the host's password is removed. The administrator will be prompted for a new password the next time Remote File Sharing is started [rfstart(1M)].

If dname is used with no options, it will default to dname -d.

NOTES

You cannot use the -N or -D options while Remote File Sharing is running.

SEE ALSO

rfstart(1M)

NAME

domainname - get/set name of current secure RPC domain

SYNOPSIS

domainname [*newname*]

DESCRIPTION

The domainname command is used on secure RPC machines. With no argument, the name of the machine's secure RPC domain is written to standard output.

The domainname command with an argument sets the name of the secure RPC domain to *newname*. *newname* may be up to 255 characters long.

domainname is normally run by the RPC administrator on all machines to set the name of the secure RPC domain. To use secure RPC, machines must have secure RPC domain names.

NOTES

Secure RPC domain names are not related to and should not be confused with RFS domains.

The RPC package expects the *newname* argument to be a valid filename for the underlying file system in use on the networked machines using secure RPC. For example, machines based on the s5 file system should not have domain names longer than 14 characters in length or problems may occur when using secure RPC.

The secure RPC domain name set by domainname will not be remembered across reboots. To give a machine a "permanent" name, set the SRPC_DOMAIN tunable in /etc/master.d/name to the secure RPC domain name.

NAME

download - host resident PostScript font downloader

SYNOPSIS

download [*options*] [*files*]

DESCRIPTION

download prepends host resident fonts to *files* and writes the results on the standard output. If no *files* are specified, or if - is one of the input *files*, the standard input is read. download assumes the input *files* make up a single PostScript job and that requested fonts can be included at the start of each input *file*. The following *options* are understood:

-f
: Force a complete scan of each input *file*. In the absence of an explicit comment pointing download to the end of the file, the default scan stops immediately after the PostScript header comments.

-p *printer*
: Before downloading, check the list of printer-resident fonts in /etc/lp/printers/*printer*/residentfonts.

-m *name*
: Use *name* as the font map table. A *name* that begins with / is the full pathname of the map table and is used as is. Otherwise *name* is appended to the pathname of the host font directory.

-H *dir*
: Use *dir* as the host font directory. The default is /usr/lib/lp/postscript.

Requested fonts are named in a comment (marked with %%DocumentFonts:) in the input *files*. Available fonts are the ones listed in the map table selected using the -m option.

The map table consists of fontname-filename pairs. The fontname is the full name of the PostScript font, exactly as it would appear in a %%DocumentFonts: comment. The filename is the pathname of the host resident font. A filename that begins with a / is used as is. Otherwise the pathname is relative to the host font directory. Comments are introduced by % (as in PostScript) and extend to the end of the line.

The only candidates for downloading are fonts listed in the map table that point download to readable files. A font is downloaded once, at most. Requests for unlisted fonts or inaccessible files are ignored. All requests are ignored if the map table can't be read.

EXAMPLES

The following map table could be used to control the downloading of the Bookman font family:

```
%
% The first string is the full PostScript font name.
% The second string is the file name - relative to the
% host font directory unless it begins with a /.
%
  Bookman-Light            bookman/light
  Bookman-LightItalic      bookman/lightitalic
  Bookman-Demi             bookman/demi
  Bookman-DemiItalic       bookman/demiitalic
```

Using the file `myprinter/map` (in the default host font directory) as the map table, you could download fonts by issuing the following command:

`download -m` *myprinter/map file*

DIAGNOSTICS

An exit status of `0` is returned if *files* were successfully processed.

NOTES

The `download` program should be part of a more general program.

`download` does not look for `%%PageFonts:` comments and there is no way to force multiple downloads of a particular font.

We do not recommend the use of full pathnames in either map tables or the names of map tables.

SEE ALSO

`dpost`(1), `postdaisy`(1), `postdmd`(1), `postio`(1), `postmd`(1), `postprint`(1), `posttek`(1)

NAME

dpost - troff postprocessor for PostScript printers

SYNOPSIS

/usr/lib/lp/postscript/dpost [*options*] [*files*]

DESCRIPTION

dpost translates *files* created by troff(1) into PostScript and writes the results on the standard output. If no *files* are specified, or if - is one of the input *files*, the standard input is read. The following *options* are understood:

-c *num* Print *num* copies of each page. By default only one copy is printed.

-e *num* Sets the text encoding level to *num*. The recognized choices are 0, 1, and 2. The size of the output file and print time should decrease as *num* increases. Level 2 encoding will typically be about 20 percent faster than level 0, which is the default and produces output essentially identical to previous versions of dpost.

-m *num* Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, which is located near the upper left corner of each page. The default magnification is 1.0.

-n *num* Print *num* logical pages on each piece of paper, where *num* can be any positive integer. By default, *num* is set to 1.

-o *list* Print those pages for which numbers are given in the comma-separated *list*. The list contains single numbers *N* and ranges *N1-N2*. A missing *N1* means the lowest numbered page, a missing *N2* means the highest.

-p *mode* Print *files* in either portrait or landscape *mode*. Only the first character of *mode* is significant. The default *mode* is portrait.

-w *num* Set the line width used to implement *troff* graphics commands to *num* points, where a point is approximately 1/72 of an inch. By default, *num* is set to 0.3 points.

-x *num* Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed near the upper left corner of the page, with positive x to the right and positive y down the page. Positive *num* moves everything right. The default offset is 0 inches.

-y *num* Translate the origin *num* inches along the positive y axis. Positive *num* moves text up the page. The default offset is 0.

-F *dir* Use *dir* as the font directory. The default *dir* is /usr/lib/font, and *dpost* reads binary font files from directory /usr/lib/font/devpost.

-H *dir* Use *dir* as the host resident font directory. Files in this directory should be complete PostScript font descriptions, and must be assigned a name that corresponds to the appropriate two-character troff font name. Each font file is copied to the output file only when needed and at most once during each job. There is no default directory.

`-L` *file* Use *file* as the PostScript prologue which, by default, is `/usr/lib/postscript/dpost.ps`.

`-O` Disables PostScript picture inclusion. A recommended option when `dpost` is run by a spooler in a networked environment.

`-T` *name* Use font files for device *name* as the best description of available PostScript fonts. By default, *name* is set to `post` and `dpost` reads binary files from `/usr/lib/font/devpost`.

The *files* should be prepared by `troff`. The default font files in `/usr/lib/font/devpost` produce the best and most efficient output. They assume a resolution of 720 dpi, and can be used to format files by adding the `-Tpost` option to the `troff` call. Older versions of the `eqn` and `pic` preprocessors need to know the resolution that `troff` will be using to format the *files*. If those are the versions installed on your system, use the `-r720` option with `eqn` and `-T720` with `pic`.

`dpost` makes no assumptions about resolutions. The first `x res` command sets the resolution used to translate the input *files*, the `DESC.out` file, usually `/usr/lib/font/devpost/DESC.out`, defines the resolution used in the binary font files, and the PostScript prologue is responsible for setting up an appropriate user coordinate system.

EXAMPLES

If the old versions of `eqn` and `pic` are installed on your system, you can obtain the best possible looking output by issuing a command line such as the following:

```
pic -T720 file | tbl | eqn -r720 | troff -mm -Tpost | dpost
```

Otherwise,

```
pic file | tbl | eqn | troff -mm -Tpost | dpost
```

should give the best results.

NOTES

Output files often do not conform to Adobe's file structuring conventions. Piping the output of `dpost` through `postreverse` should produce a minimally conforming PostScript file.

Although `dpost` can handle files formatted for any device, emulation is expensive and can easily double the print time and the size of the output file. No attempt has been made to implement the character sets or fonts available on all devices supported by `troff`. Missing characters will be replaced by white space, and unrecognized fonts will usually default to one of the Times fonts (that is, `R`, `I`, `B`, or `BI`).

An `x res` command must precede the first `x init` command, and all the input *files* should have been prepared for the same output device.

Use of the `-T` option is not encouraged. Its only purpose is to enable the use of other PostScript font and device description files, that perhaps use different resolutions, character sets, or fonts.

Although level 0 encoding is the only scheme that has been thoroughly tested, level 2 is fast and may be worth a try.

DIAGNOSTICS

An exit status of 0 is returned if *files* have been translated successfully, while 2 often indicates a syntax error in the input *files*.

FILES

```
/usr/lib/font/devpost/*.out
/usr/lib/font/devpost/charlib/*
/usr/lib/lp/postscript/dpost.ps
/usr/lib/lp/postscript/color.ps
/usr/lib/lp/postscript/draw.ps
/usr/lib/lp/postscript/forms.ps
/usr/lib/lp/postscript/ps.requests
/usr/lib/macros/pictures
/usr/lib/macros/color
```

SEE ALSO

download(1), postdaisy(1), postdmd(1), postio(1), postmd(1), postprint(1), postreverse(1), posttek(1), troff(1) devpost(5), troff(5)

NAME

drvinstall - install/uninstall a driver

SYNOPSIS

/usr/sbin/drvinstall [-m *master*] [-d *object*] [-s *system*]
[-e *edt_data*] [-i *editdata*]
[-o *directory*] [-c *minor*]
-v *version* [-ufbnx]

DESCRIPTION

The drvinstall command accepts an *object* file, master file, system file and EDT data as inputs, and creates the corresponding specially formatted file for use in the configuration of a new bootable operating system. In addition, the master, edt_data, and system files may be modified. The -u option is used for uninstalling a driver. Pathnames specified for the options below can be either relative or absolute pathnames.

-m *master*
: specifies the path name of the master directory to be used. One or both of the -m or -d options must be specified, and at least one must specify a file name as the last component of the path name. If this flag is omitted, the /etc/master.d directory is used.

-d *object*
: specifies the path name of the input *object* directory to be used. One or both of the -m or -d options must be specified, and at least one must specify a file name as the last component of the path name. If this flag is omitted, the /boot directory is used.

-s *system*
: specifies the path name of the system file to be used. If this flag is omitted, the /stand/system file is used.

-e
: specifies the path name of the edt_data file to be updated. If this flag is omitted, /stand/edt_data is used. This flag is meaningful for hardware drives only.

-i
: Specifies a file containing EDT data for the driver (hardware only). This flag is required for hardware drivers and is invalid for anything else. The format of *edtfile* is the same as the edt_data file.

-o *directory*
: specifies the path name of the output bootable file. If this flag is omitted, the /boot directory is used.

-c *minor*
: specifies the *minor* number to be inserted at the end of an INCLUDE statement for the driver. The INCLUDE statement is inserted in the *system* file. *Minor* is optional in an INCLUDE and specifies the quantity (default of 1) of minor devices to be controlled by the driver. If the driver is a hardware driver, -c is ignored.

-v *version*
: specifies the *version* number of the drvinstall command compatible with the master file being used. The -v option is required on the command line and currently supports "1.0".

`-u` specifies that the named driver is to be uninstalled. A driver dependency check is made and if a dependency is found, a warning message is issued and the command is aborted. If no dependency is found, then `drvinstall` will:

- Remove corresponding entries in the `edt_data` file (hardware drivers only).
- Remove the bootable *object* file.
- Replace the major number with a "-" in the `master` file if the driver is a software driver.
- Delete the INCLUDE statement from the `system` file if the driver is not a hardware driver.
- Print the major number if the driver is a software or hardware driver.

`-f` when used with the `-u` option, disables the dependency check. This results in the driver being uninstalled regardless of dependencies.

`-b` inhibits generation of the *object* file. This option is ignored for uninstall.

`-n` Inhibits any edit of the `system` file.

`-x` Enables debugging output.

For any driver installed, `drvinstall` calls the `mkboot` command to produce a bootable *object* file. The resultant output file is placed into the directory determined by the `-o` argument.

If the driver to be installed is a software or hardware driver, `drvinstall` will:

- Assign a major number to that driver if there is a "-" entry in the major number field of the associated `master` file entry. The `drvinstall` command expects any unused field of the `master` file to be filled with a "-".

  The `drvinstall` command determines the available major numbers by scanning all existing files in `/etc/master.d` for major numbers; it then assigns the first unused number in the above range (note that the directory specified by *master* on the command line is not searched for major numbers). This value replaces the corresponding "-" value in the major number field of the `master` file.

- For hardware drivers, `drvinstall` adds entries (install) or removes entries (uninstall) from `edt_data`. For uninstall, only those drivers' entries in *editfile* that match by name and board number in `edt_data` are removed. The format of *edtdata* is the same as `edt_data`.

- Print the major number found or assigned in the `master` file.

If the driver to be installed is not a hardware driver (it is, e.g., a software driver or a loadable type of module), `drvinstall` will insert an INCLUDE statement for the driver in the `system` file.

SEE ALSO

`cunix`(1M), `mkboot`(1M), `master`(4), `system`(4)

DIAGNOSTICS

The major number assigned or found for a software driver is printed on `stdout`. A zero is returned for success and a non-zero is returned for failures.

NAME

`dsconfig` - display data storage device configuration

SYNOPSIS

`/usr/bin/dsconfig` [*simple_administration_device_name*]

DESCRIPTION

The `dsconfig` command produces the mapping of the simple administration names for data storage devices found in `/dev/rSA` to the device names found in `/dev/rdsk` or `/dev/rmt` and prints the physical location of the associated peripheral on the machine. The `dsconfig` command with no arguments prints the mapping for every entry in `/dev/rSA`.

EXAMPLE

```
dsconfig disk1 disk6

SA: disk1
device: /dev/rdsk/c1d0s6
configuration: Integral Disk Drive 0

SA: ctape1
device: /dev/rmt/ctape1
configuration: Tape Drive 1
```

NAME

du - summarize disk usage

SYNOPSIS

du [-aemrs] [*name* ...]

DESCRIPTION

The du command reports the number of blocks contained in all files and (recursively) directories within each directory and file specified. The block count includes the indirect blocks of the file. If no *name*s are given, the current directory is used.

The optional arguments are as follows:

-a causes an output line to be generated for each file.

If neither -s or -a is specified, an output line is generated for each directory only.

-e causes an du to exit with a return code of 3 if it was not possible to account for every multi-linked file.

-m causes du to only search the file system containing the argument file.

-r will cause du to generate messages about directories that cannot be be read, files that cannot be opened, etc., rather than being silent (the default).

-s causes only the grand total (for each of the specified *name*s) to be given.

A file with two or more links is only counted once.

NOTES

If the -a option is not used, non-directories given as arguments are not listed.

If there are links between files in different directories where the directories are on separate branches of the file system hierarchy, du will count the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

du - display the number of disk blocks used per directory or file

SYNOPSIS

/usr/ucb/du [-F ufs]

/usr/ucb/du [-F ufs] [-a] [-s] [*filename* . . .]

DESCRIPTION

du gives the number of kilobytes contained in all files and, recursively, directories within each specified directory or file *filename*. If *filename* is missing, '.' (the current directory) is used.

A file which has multiple links to it is only counted once.

OPTIONS

-a Generate an entry for each file.

-s Only display the grand total for each of the specified *filename*s.

Entries are generated only for each directory in the absence of options.

EXAMPLE

Here is an example of using du in a directory. We used the pwd(1) command to identify the directory, then used du to show the usage of all the subdirectories in that directory. The grand total for the directory is the last entry in the display:

```
% pwd
/usr/ralph/misc
% du
5     ./jokes
33    ./squash
44    ./tech.papers/lpr.document
217   ./tech.papers/new.manager
401   ./tech.papers
144   ./memos
80    ./letters
388   ./window
93    ./messages
15    ./useful.news
1211  .
%
```

SEE ALSO

df(1M), pwd(1), quot(1M)

NOTES

Filename arguments that are not directory names are ignored, unless you use -a.

If there are too many distinct linked files, du will count the excess files more than once.

NAME

dump - dump selected parts of an object file

SYNOPSIS

dump *options files*

DESCRIPTION

The dump command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

-a Dump the archive header of each member of an archive.

-C Dump decoded C++ symbol table names.

-c Dump the string table(s).

-D Dump debugging information.

-f Dump each file header.

-g Dump the global symbols in the symbol table of an archive.

-h Dump the section headers.

-L Dump dynamic linking information and static shared library information, if available.

-l Dump line number information.

-o Dump each program execution header.

-r Dump relocation information.

-s Dump section contents in hexadecimal.

-T *index* or -T *index1*, *index2*
Dump only the indexed symbol table entry defined by *index* or a range of entries defined by *index1*, *index2*.

-t Dump symbol table entries.

-u When reading a COFF object file, dump translates the file to ELF internally (this translation does not affect the file contents). This option controls how much translation occurs from COFF values to ELF. Normally (without -u), the COFF values are preserved as much as possible, showing the actual bytes in the file. If -u is used, dump updates the values and completes the internal translation, giving a consistent ELF view of the contents. Although the bytes displayed under this option might not match the file itself, they show how the file would look if it were converted to ELF. (See cof2elf(1) for more information.)

-V Print version information.

The following modifiers are used in conjunction with the options listed above to modify their capabilities.

-d *number* or -d *number1*, *number2*
Dump the section number indicated by *number* or the range of sections starting at *number1* and ending at *number2*. This modifier can be used with -h, -s, and -r. When -d is used with -h or -s, the

argument is treated as the number of a section or range of sections. When -d is used with -r, the argument is treated as the number of the section or range of sections to which the relocation applies. For example, to print out all relocation entries associated with the .text section, specify the number of the section as the argument to -d. If .text is section number 2 in the file, dump -r -d 2 will print all associated entries. To print out a specific relocation section use dump -s -n *name* for raw data output, or dump -sv -n *name* for interpreted output.

-n *name* Dump information pertaining only to the named entity. This modifier can be used with -h, -s, -r, and -t. When -n is used with -h or -s, the argument will be treated as the name of a section. When -n is used with -t or -r, the argument will be treated as the name of a symbol. For example, dump -t -n .text will dump the symbol table entry associated with the symbol whose name is .text, where dump -h -n .text will dump the section header information for the .text section.

-p Suppress printing of the headings.

-v Dump information in symbolic representation rather than numeric. This modifier can be used with -a (date, user id, group id), -f (class, data, type, machine, version, flags), -h (type, flags), -o (type, flags), -r (name, type), -s (interpret section contents wherever possible), -t (type, bind), and -L (value). When -v is used with -s, all sections that can be interpreted, such as the string table or symbol table, will be interpreted. For example, dump -sv -n .symtab *files* will produce the same formatted output as dump -tv *files*, but dump -s -n .symtab *files* will print raw data in hexadecimal. Without additional modifiers, dump -sv *files* will dump all sections in the files interpreting all those that it can and dumping the rest (such as .text or .data) as raw data.

The dump command attempts to format the information it dumps in a meaningful way, printing certain information in character, hexadecimal, octal or decimal representation as appropriate.

SEE ALSO

a.out(4), ar(4).

NAME

echo - echo arguments

SYNOPSIS

echo [-n] [*arg*] ...

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a newline on the standard output. This syntax is used by the shell [sh(1) and ksh(1)] built-in echo commands as well as the user command /usr/bin/echo.

The -n option is available only using the sh(1) built-in echo command and is only available when the directory /usr/ucb preceeds the directory /usr/bin in the user's PATH environment variable. When the -n option is available, none of the other C-like escape sequences described below are available to the user.

The following C-like escape conventions are supported; beware of conflicts with the shell's use of \ :

\b	backspace
\c	print line without newline
\f	form-feed
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\0*n*	where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

echo is useful for producing diagnostics in command files, for sending known data into a pipe, and for displaying the contents of environment variables.

INTERNATIONAL FUNCTIONS

Arguments containing characters from supplementary code sets can be specified. Note that when octal notation is used, each byte of multibyte characters should be preceded by a backslash (\).

SEE ALSO

csh(1), ksh(1), sh(1), ascii(5).

NOTES

The -n option is a transition aid for BSD applications, and may not be supported in future releases.

When representing an 8-bit character by using the escape convention \0*n*, the *n* must always be preceded by the digit zero (0).

For example, entering: echo ´WARNING:\07´ will print the phrase WARNING: and sound the bell on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the "\" that precedes the "07".

Following the \0, up to three digits are used in constructing the octal output character. If, following the \0*n*, you want to echo additional digits that are not part of the octal representation, you must use the full 3-digit *n*. For example, if you want to echo "ESC 7" you must use the three digits "033" rather than just the two digits "33" after the \0.

2 digits	Incorrect:	`echo "\0337" | od -xc`	
	produces:	`df0a`	(hex)
		`337`	(ascii)
3 digits	Correct:	`echo "\00337" | od -xc`	
	produces:	`1b37 0a00`	(hex)
		`033 7`	(ascii)

For the octal equivalents of each character, see `ascii`(5).

NAME

echo - put string on virtual output

SYNOPSIS

echo [*string . . .*]

DESCRIPTION

The echo function directs each string it is passed to *stdout*. It is often used in conditional execution or for passing a string to another command.

EXAMPLES

Set the done descriptor to help if a test fails:

```
done=`if [ -s $F1 ];
      then echo close;
      else echo help;
      fi`
```

SEE ALSO

echo(1)

NAME

echo - echo arguments

SYNOPSIS

echo [-n] [arg] ...

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output. This echo syntax is used by the csh(1) built-in echo command as well as user command /usr/ucb/echo. The -n option supresses the trailing new-line.

echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

csh(1), echo(1)

NAME

ed, red - text editor

SYNOPSIS

ed [-s] [-p *string*] [-x] [-C] [*file*]

red [-s] [-p *string*] [-x] [-C] [*file*]

DESCRIPTION

ed is the standard 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.

-s 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*.

-p Allows the user to specify a prompt string. The prompt string can contain characters from supplementary code sets.

-x Encryption option; when used, ed simulates an X command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of crypt(1). The X command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the -x option. See crypt(1). Also, see the NOTES section at the end of this manual page.

-C Encryption option; the same as the -x option, except that ed simulates a C command. The C command is like the X command, except that all text read in is assumed to have been encrypted.

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: when you are entering text into the file, this format is not in effect; instead, because of being in stty -tabs or stty tab3 mode, tabs are expanded to every eighth column.

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. Leave input mode by typing a period (.) at the beginning of a line, followed immediately by a carriage return.

`ed` supports a limited form of regular expression notation; regular expressions are used in addresses to specify lines and in some commands (for example, `s`) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be matched by the regular expression. The regular expressions allowed by `ed` are constructed as follows:

The following one-character regular expressions match a single character:

1.1 An ordinary character (not one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash (`\`) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

a. `.`, `*`, `[`, and `\` (period, asterisk, left square bracket, and backslash, respectively), which are always special, except when they appear within square brackets (`[ ]`; see 1.4 below).

b. `^` (caret or circumflex), which is special at the beginning of an entire regular expression (see 4.1 and 4.3 below), or when it immediately follows the left of a pair of square brackets (`[ ]`) (see 1.4 below).

c. `$` (dollar sign), which is special at the `end` of an entire regular expression (see 4.2 below).

d. The character used to bound (that is, delimit) an entire regular expression, which is special for that regular expression (for example, see how slash (`/`) is used in the `g` command, below.)

1.3 A period (.) is a one-character regular expression that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets (`[ ]`) is a one-character regular expression that matches any one character in that string. If, however, the first character of the string is a circumflex (`^`), the one-character regular expression matches any character except new-line and the remaining characters in the string. The `^` has this special meaning only if it occurs first in the string. The minus (`-`) may be used to indicate a range of consecutive characters; for example, `[0-9]` is equivalent to `[0123456789]`. The `-` loses this special meaning if it occurs first (after an initial `^`, if any) or last in the string. The right square bracket (`]`) does not terminate such a string when it is the first character within it (after an initial `^`, if any); for example, `[ ]a-f]` matches either a right square bracket (`]`) or one of the ASCII letters `a` through `f` inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct regular expressions from one-character regular expressions:

2.1 A one-character regular expression is a regular expression that matches whatever the one-character regular expression matches.

2.2 A one-character regular expression followed by an asterisk (*) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character regular expression followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a regular expression that matches a range of occurrences of the one-character regular expression. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches exactly *m* occurrences; \{*m*,\} matches at least *m* occurrences; \{*m*,*n*\} matches any number of occurrences between *m* and *n* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

2.4 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.5 A regular expression enclosed between the character sequences \(and \) is a regular expression that matches whatever the unadorned regular expression matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) earlier in the same regular expression. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

A regular expression may be constrained to match words.

3.1 \< constrains a regular expression to match the beginning of a string or to follow a character that is not a digit, underscore, or letter. The first character matching the regular expression must be a digit, underscore, or letter.

3.2 \> constrains a regular expression to match the end of a string or to precede a character that is not a digit, underscore, or letter.

An entire regular expression may be constrained to match only an initial segment or final segment of a line (or both).

4.1 A circumflex (^) at the beginning of an entire regular expression constrains that regular expression to match an initial segment of a line.

4.2 A dollar sign ($) at the end of an entire regular expression constrains that regular expression to match a final segment of a line.

4.3 The construction ^*entire regular expression*$ constrains the entire regular expression to match the entire line.

The null regular expression (for example, //) is equivalent to the last regular expression encountered. See also the last paragraph before FILES below.

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. 'x addresses the line marked with the mark name character *x*, which must be an ASCII lower-case letter (a-z). Lines are marked with the k command described below.
5. A regular expression 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 regular expression. 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 regular expression 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 regular expression. 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. A shorthand for .+5 is .5.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; for example, -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 first address is calculated, the current line (.) is set to that value, and then the second address is 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 in the buffer that follows the line corresponding to the first address.

In the following list of ed commands, the parentheses shown prior to the command are not part of the address; rather they show the default address(es) for the command.

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 either listed, numbered or printed, respectively, as discussed below under the l, n, and p commands.

(.)a
<text>
.

The append command accepts zero or more lines of text and appends it after the addressed line in the buffer. The current 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 may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The change command deletes the addressed lines from the buffer, then accepts zero or more lines of text that replaces these lines in the buffer. The current line (.) is left at the last line input, or, if there were none, at the first line that was not deleted.

C

Same as the X command, described later, except that ed assumes all text read in for the e and r commands is encrypted unless a null key is typed in.

(.,.)d

The delete 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 edit command deletes the entire contents of the buffer and then reads the contents of *file* into the buffer. The current line (.) is set to the last line of the buffer. If *file* is not given, the currently remembered file name, if any, is used (see the f command). The number of characters read in is printed; *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 in. Such a shell command is not remembered as the current file name. See also DIAGNOSTICS below.

E *file*

The Edit 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/*regular expression*/*command list*

In the global command, the first step is to mark every line that matches the given regular expression. Then, for every such line, the given *command list* is executed with the current line (.) 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 the NOTES and the last paragraph before FILES below.

(1,$)G/*regular expression*/

In the interactive Global command, the first step is to mark every line that matches the given regular expression. Then, for every such line, that line is printed, the current line (.) 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 help command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The Help 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 insert command accepts zero or more lines of text and inserts it before the addressed line in the buffer. The current 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 join command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mark command marks the addressed line with name *x*, which must be an ASCII lower-case letter (a-z). The address ´*x* then addresses this line; the current line (.) is unchanged.

(.,.)l

The list command prints the addressed lines in an unambiguous way: a few non-printing characters (for example, *tab, backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An l command may be appended to any command other than e, f, r, or w.

(.,.)m*a*

The move 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; the current line (.) is left at the last line moved.

(.,.)n

The number command prints the addressed lines, preceding each line by its line number and a tab character; the current line (.) is left at the last line printed. The n command may be appended to any command other than e, f, r, or w.

(.,.)p

The print command prints the addressed lines; the current line (.) is left at the last line printed. The p command may be appended to any 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 quit command causes ed to exit. No automatic write of a file is done; however, 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 read command reads the contents of *file* into the buffer. If *file* is not given, the currently remembered file name, if any, is used (see the 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 in at the beginning of the buffer. If the read is successful, the number of characters read in is printed; the current line (.) is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell [see sh(1)] command whose output is to be read in. 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/*regular expression*/*replacement*/ or
(. , .) s/*regular expression*/*replacement*/g or
(. , .) s/*regular expression*/*replacement*/*n* *n* = 1-512

The substitute command searches each addressed line for an occurrence of the specified regular expression. 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 if the substitution fails on all addressed lines. Any character other than space or new-line may be used instead of / to delimit the regular expression and the *replacement*; the current line (.) 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 regular expression 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 regular expression 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 newline 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); the current line (.) is left at the last line copied.

u

The undo 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/*regular expression/command list*

This command is the same as the global command g, except that the lines marked during the first step are those that do not match the regular expression.

(1,$)V/*regular expression/*

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 regular expression.

(1,$)w *file*

The write command writes the addressed lines into *file*. If *file* does not exist, it is created with mode 666 (readable and writable by everyone), unless your file creation mask dictates otherwise; see the description of the umask special command on sh(1). 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 the e and f commands); the current line (.) is unchanged. If the command is successful, the number of characters written is printed. If *file* is replaced by !, the rest of the line is taken to be a shell [see sh(1)] command whose standard input is the addressed lines. Such a shell command is not remembered as the current file name.

(1,$)W *file*

This command is the same as the write command above, except that it appends the addressed lines to the end of *file* if it exists. If *file* does not exist, it is created as described above for the *w* command.

X

A key is prompted for, and it is used in subsequent e, r, and w commands to decrypt and encrypt text using the crypt(1) algorithm. An educated guess is made to determine whether text read in for the e and r commands is encrypted. A null key turns off encryption. Subsequent e, r, and w commands will use this key to encrypt or decrypt the text [see crypt(1)]. An explicitly empty key turns off encryption. Also, see the -x option of ed.

($)=

The line number of the addressed line is typed; the current line (.) is unchanged by this command.

!*shell command*

The remainder of the line after the ! is sent to the UNIX system shell [see 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; the current line (`.`) 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 in a line, 256 characters in a global command list, and 64 characters in the pathname of a file (counting slashes). 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.

If a file is not terminated by a new-line character, `ed` adds one and puts out a message explaining what it did.

If the closing delimiter of a regular expression or of a replacement string (for example, `/`) 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?
```

FILES

`$TMPDIR`	if this environmental variable is not null, its value is used in place of `/var/tmp` as the directory name for the temporary work file.
`/var/tmp`	if `/var/tmp` exists, it is used as the directory name for the temporary work file.
`/tmp`	if the environmental variable `TMPDIR` does not exist or is null, and if `/var/tmp` does not exist, then `/tmp` is used as the directory name for the temporary work file.
`ed.hup`	work is saved here if the terminal is hung up.

INTERNATIONAL FUNCTIONS

`ed` can process characters from supplementary code sets as well as ASCII characters.

Searches and pattern matching with regular expressions are performed in character units, not in individual bytes.

`.`	Matches any character from supplementary code sets.
`*`	Matches zero or more occurrances of the given regular expression.
`.*`	Matches any character string including the null string.
`[ ]`	Matches any one character in the string enclosed by square brackets, or any one character with a code value within the range designated using a minus (-) sign. When the characters in the range are from different code sets, one of the characters specifing the range is matched.
`[^ ]`	Excludes the specified character from all characters from supplementary code sets matched.

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1), fspec(4), regexp(5).

DIAGNOSTICS

?	for command errors.
?*file*	for an inaccessible file. (use the help and Help 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 -s command-line option inhibits this feature.

NOTES

The - option, although it continues to be supported, has been replaced in the documentation by the -s option that follows the Command Syntax Standard [see intro(1)].

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

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 editor is invoked from a restricted shell [see sh(1)].

The sequence \n in a regular expression does not match a new-line character.

If the editor input is coming from a command file (for example, ed *file* < *ed_cmd_file*), the editor exits at the first failure.

The size of the file displayed at first and after read/write by the e, r, and w commands is in bytes, not characters.

NAME

edit - text editor (variant of ex for casual users)

SYNOPSIS

edit [-r] [-x] [-C] *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. It operates precisely as ex with the following options automatically set:

novice	ON
report	ON
showmode	ON
magic	OFF

These options can be turned on or off via the set command in ex(1).

-r Recover file after an editor or system crash.

-x Encryption option; when used the file will be encrypted as it is being written and will require an encryption key to be read. edit makes an educated guess to determine if a file is encrypted or not. See crypt(1). Also, see the **NOTES** section at the end of this manual page.

-C Encryption option; the same as -x except that edit assumes files are encrypted.

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.

To edit the contents of an existing file, begin with the command edit *name* to the shell. edit makes a copy of the file that you can then edit, and tells you how many lines and characters are in the file. To create a new file, you also begin with the command edit with a filename: edit *name*; the editor will tell you it is a [New File].

The edit command prompt is the colon (:) which you should see after starting the editor. If you are editing an existing file, you will have some lines in edit's buffer (its name for the copy of the file you are editing). When you start editing, edit makes the last line of the file the current line. Most commands to edit use the current line if you do not tell them which line to use. Thus, if you enter print (which can be abbreviated p) and enter carriage return (as you should after all edit commands), the current line will be printed. If you delete (d) the current line, edit will print the new current line, which is usually the next line in the file. If you delete the last line, the new last line becomes the current one.

If you start with an empty file or wish to add some new lines, the append (a) command can be used. After you execute this command (enter a carriage return after the word append), edit will read lines from your terminal until you enter a line consisting of just a dot (.); it places these lines after the current line. The last line you enter then becomes the current line. The insert (i) command is like append, but places the lines you enter before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you execute the command 1, then edit will enter the first line of the buffer. If you then execute the command d, 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: s/*old* /*new*/ where *old* is the string of characters you want to replace and *new* is the string of characters you want to replace *old* with.

The file (f) command will tell you how many lines there are in the buffer you are editing and will display [Modified] if you have changed the buffer. After modifying a file, you can save the contents of the file by executing a write (w) command. You can 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 receive the message No write since last change (:quit! overrides), and edit will wait for another command. If you do not want to write the buffer out, issue the quit command followed by an exclamation point (q!). The buffer is then irretrievably discarded and you return to the shell.

The change (c) command changes the current line to a sequence of lines you supply (as in append, you enter lines up to a line consisting of only a dot (.). You can tell change to change more than one line by giving the line numbers of the lines you want to change, i.e., 3,5c. You can print lines this way too: 1,23p prints the first 23 lines of the file.

The undo (u) command reverses the effect of the last command you executed that changed the buffer. Thus if you execute a substitute command that does not do what you want, enter u and the old contents of the line will be restored. You can also undo an undo command. edit will give you a warning message when a command affects more than one line of the buffer. Note that commands such as write and quit cannot be undone.

To look at the next line in the buffer, enter carriage return. To look at a number of lines, enter ^D (while holding down the control key, press d) 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 nearby text by executing the z command. The current line will appear in the middle of the text displayed, and the last line displayed will become the current line; you can get back to the line where you were before you executed the z command by entering ´´. The z command has other options: 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, enter z.11 to display five lines before and five lines after the current line. (Entering z.*n*, when *n* is an odd number, displays a total of *n* lines, centered about the current line; when *n* is an even number it displays *n*-1 lines so that the lines displayed are centered around the current line.) You can give counts after other commands; for example, you can delete 5 lines starting with the current line with the command d5 .

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 *text*, it wraps

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 the symbolic name dot (.); this is most useful in a range of lines as in .,$p which prints the current line plus the rest of the lines in the file. To move to the last line in the file, you can refer to it by its symbolic name $. Thus the command $d deletes the last line in the file, no matter what the current line is. Arithmetic with line references is also possible. Thus the line $-5 is the fifth before the last and .+20 is 20 lines after the current line.

You can find out the current line by entering .=. This is useful if you wish to move or copy a section of text within a file or between files. Find the first and last line numbers you wish to copy or move. To move lines 10 through 20, enter 10,20d a to delete these lines from the file and place them in a buffer named a. edit has 26 such buffers named a through z. To put the contents of buffer a after the current line, enter put a. If you want to move or copy these lines to another file, execute an edit (e) command after copying the lines; following the e command with the name of the other file you wish to edit, i.e., edit chapter2. To copy lines without deleting them, use yank (y) in place of d. If the text you wish to move or copy is all within one file, it is not necessary to use named buffers. For example, to move lines 10 through 20 to the end of the file, enter 10,20m $.

INTERNATIONAL FUNCTIONS

edit can process characters from supplementary code sets.

SEE ALSO

ed(1), ex(1), vi(1).

NOTES

The encryption options are provided with the Security Administration Utilities package, which is available only in the United States.

NAME

edquota - edit user quotas

SYNOPSIS

edquota [-p *proto_user*] *username...*
edquota -t

DESCRIPTION

edquota is a quota editor. One or more users may be specified on the command line. For each user a temporary file is created with an ASCII representation of the current disk quotas for that user for each mounted ufs file system that has a quotas file, and an editor is then invoked on the file. A null entry is used if no quotas file exists for a file system. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, edquota reads the temporary file and modifies the binary quota files to reflect the changes made.

The editor invoked is vi(1) unless the EDITOR environment variable specifies otherwise.

Only the super-user may edit quotas. In order for quotas to be established on a file system, the root directory of the file system must contain a file, owned by root, called quotas. See quotaon(1M) for details.

proto_user and username can be numeric, corresponding to the uid of a user. Unassigned uids may be specified; unassigned names may not. In this way, default quotas can be established for users who are later assigned a uid.

The options are:

-p Duplicate the quotas of the *proto_user* specified for each *username* specified. This is the normal mechanism used to initialize quotas for groups of users.

-t Edit the soft time limits for each file system. If the time limits are zero, the default time limits in /usr/include/sys/fs/ufs_quota.h are used. Time units of sec(onds), min(utes), hour(s), day(s), week(s), and month(s) are understood. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

FILES

quotas	quota file at the file system root
/etc/mnttab	table of mounted file systems

SEE ALSO

quota(1M), quotacheck(1M), quotaon(1M), repquota(1M), vi(1)

NAME

edsysadm - sysadm interface editing tool

SYNOPSIS

edsysadm

DESCRIPTION

edsysadm is an interactive tool that adds or changes either menu and task definitions in the sysadm interface. It can be used to make changes directly on-line on a specific machine or to create changes that will become part of a software package. The command creates the administration files necessary to achieve the requested changes in the interface and either places them in the appropriate place for on-line changes or saves them to be included in a software package.

edsysadm presents several screens, first prompting for which type of menu item you want to change, menu or task, and then for what type of action to take, add or change. When you select add, a blank menu or task definition (as described below) is provided for you to fill in. When you select change, a series of screens is presented to help identify the definition you wish to change. The final screen presented is the menu or task definition filled in with its current values, which you can then edit.

The menu definition prompts and their descriptions are:

Menu Name
: The name of the new menu (as it should appear in the lefthand column of the screen). This field has a maximum length of 16 alphanumeric characters.

Menu Description
: A description of the new menu (as it should appear in the righthand column of the screen). This field has a maximum length of 58 characters and can consist of any alphanumeric character except at sign (@), carat (^), tilde (~), back grave (`), grave ('), and double quotes (").

Menu Location
: The location of the menu in the menu hierarchy, expressed as a menu pathname. The pathname should begin with the main menu followed by all other menus that must be traversed (in the order they are traversed) to access this menu. Each menu name must be separated by colons. For example, the menu location for a menu entry being added to the Applications menu is main:applications. Do not include the menu name in this location definition. The complete pathname to this menu entry will be the menu location plus the menu name defined at the first prompt.

  This is a scrollable field, showing a maximum of 50 alphanumeric characters at a time.

`Menu Help File Name` Pathname to the item help file for this menu entry. If it resides in the directory from which you invoked `edsysadm`, you do not need to give a full pathname. If you name an item help file that does not exist, you are placed in an editor (as defined by `$EDITOR`) to create one. The new file is created in the current directory and named `Help`.

The task definition prompts and their descriptions are:

`Task Name` The name of the new task (as it should appear in the lefthand column of the screen). This field has a maximum length of 16 alphanumeric characters.

`Task Description` A description of the new task (as it should appear in the righthand column of the screen). This field has a maximum length of 58 characters and can consist of any alphanumeric character except at sign (@), carat (^), tilde (~), back grave (`), grave ('), and double quotes (").

`Task Location` The location of the task in the menu hierarchy, expressed as a pathname. The pathname should begin with the main menu followed by all other menus that must be traversed (in the order they are traversed) to access this task. Each menu name must be separated by colons. For example, the task location for a task entry being added to the applications menu is `main:applications`. Do not include the task name in this location definition. The complete pathname to this task entry will be the task location as well as the task name defined at the first prompt.

This is a scrollable field, showing a maximum of 50 alphanumeric characters at a time.

`Task Help File Name` Pathname to the item help file for this task entry. If it resides in the directory from which you invoked `edsysadm`, you do not need to give a full pathname. If you name an item help file that does not exist, you are placed in an editor (as defined by `$EDITOR`) to create one. The new file is created in the current directory and named `Help`.

`Task Action` The FACE form name or executable that will be run when this task is selected. This is a scrollable field, showing a maximum of 58 alphanumeric characters at a time. This pathname can be relative to the current directory as well as absolute.

`Task Files` Any FACE objects or other executables that support the task action listed above and might be called from within that action. *Do not include the help file name or the task action in this list*. Pathnames can be relative to the current directory as well as absolute. A dot (.) implies "all files in the current directory" and includes files in

subdirectories.

This is a scrollable field, showing a maximum of 50 alphanumeric characters at a time.

Once the menu or task has been defined, screens for installing the menu or task or saving them for packaging are presented. The package creation or on-line installation is verified and you are informed upon completion.

NOTES

For package creation or modification, this command automatically creates a menu information file and a `prototype` file in the current directory (the directory from which the command is executed). The menu information file is used during package installation to modify menus in the menu structure. A `prototype` file is an installation file which gives a listing of package contents. The `prototype` file created by `edsysadm` lists the files defined under task action and gives them the special installation class of "admin". The contents of this `prototype` file must be incorporated in the package `prototype` file.

For on-line installation, `edsysadm` automatically creates a menu information file and adds or modifies the interface menu structure directly.

SEE ALSO

`delsysadm`(1M), `pkgmk`(1), `sysadm`(1M), `prototype`(4).

NAME

edtp - Equipped Device Table Probe procedures

DESCRIPTION

The construction of the Equipped Device Table (EDT) is achieved by standalone programs probing for hardware devices on the system bus. These programs are referred to as the EDT Probe (EDTP) software.

The bootstrap procedure consists of the following basic phases.

On powerup, the boot process is generally begun automatically: a small boot program is loaded and executed, and the process moves into the second phase.

Then, it will locate any EDTP programs on the Boot File System (*/stand*). If there are any file names commencing with the letters `EDTP`. the boot program will load the file and then pass control to the loaded program.

The program will then attempt to probe for the particular board it was designed to find. If the probe is successful, the program adds the device to the EDT constructed by the boot program. The program adds the EDT data corresponding to the device found in `/stand/edt_data`. The EDTP program then passes control back to the boot program.

When the boot program has run all EDTP programs, it then checks to see whether all boards have corresponding drivers and also if all drivers have corresponding boards. If not, then a reconfiguration flag is passed to the kernel and */stand/mUNIX* is loaded instead. Otherwise, */stand/unix* is loaded.

The `probe` function will access the address specified by `addr` and return whether the access caused an exception. `size` can be one of `CHARSZ`, `SHORTSZ`, `INTSZ` or `LONGSZ` (`sizeof(char)`, `sizeof(short)`, `sizeof(int)` and `sizeof(long)` respectively). The `rwflag` can be set to `R_ONLY`, `W_ONLY`, `WR_RD` and `RD_WR` for read only, write only, write then read and read then write, where the contents of `value_ptr` contain the value to write or will contain the value read after success. It is preferable to use `probe` to access the address than to do so directly because the `probe` function will clean up the pipeline and return cleanly after an exception.

M88000 family of processors only:

The function will return `0` for a successful access or return the exception number that occurred when accessing *addr*. Note that on System V/88 only bus errors and interrupts are valid exceptions.

M68000 family of processors only:

The function will return `0` for a successful access or `-1` for a bus error that occurred when accessing *addr*. Note that on System V/68 only bus errors are valid exceptions.

M68000 or M88000 family of processors

Standard support functions.

Some standard library functions will also be supplied. The list includes `printf`(), `strcpy`(), `strncpy`(), `strcmp`() and `strncmp`().

EXAMPLE

```
/*
 * Example EDT probe program, that tries to probe for all MVME328
 * device entries in the EDT data.
```

```
 */

#include <sys/types.h>
#include <sys/iosystm.h>
#include <sys/edt.h>
#include <sys/edtp.h>
char *name = "MVME328";


/*
 *    edtmessage
 */

static void
edtmessage(struct edt *edtp, char *string)
{
      printf("Probe for %s [%d] @ (0x%x) %s0, edtp -> dev_name,
            edtp -> board, edtp -> io_addr, string);

      return;
}

/*
 *    main
 */

main(struct edt *edt_data, struct edt *newedt, int debug)
{
      register struct edt   *e;
      register struct edt   *newe;
      char            res;
      unsigned long         val;

      /* Find first free slot at end of new edt table */

      for (newe = newedt; newe -> dev_name[0] != ' '; newe++)
            ;

      /*
       * Search parsed copy of /stand/edt_data and probe for
       * devices we are interested in. If the device is there
       * add it to the table.
       */

      for (e = edt_data; e -> dev_name[0] != ' '; e++)
      {
            if (strncmp(e -> dev_name, name, E_NAMLEN) != 0)
            {
                  continue;
            }
```

```
                if (e -> flags & E_PAD)
                {
                        if (debug & EDTP_SKIPPED)
                        {
                                edtmessage(e, "skipped");
                        }

                        continue;
                }

                if (probe(e -> io_addr, CHARSZ, R_ONLY, &val) == 0)
                {
                        if ((newe - newedt) > MAXEDT)
                        {
                                edtmessage(e, "successful but no EDT space");

                                continue;
                        }

                        if (debug & EDTP_FOUND)
                        {
                                edtmessage(e, "successful");
                        }

                        *newe++ = *e;
                }
                else
                {
                        if (debug & EDTP_NOTFOUND)
                        {
                                edtmessage(e, "unsuccessful");
                        }
                }
        }

        return;
}
```

NOTES

The probe routine must be used to access devices that may or may not be in the system address space. If the probe function is not used unpredictable behavior can occur. Currently boards that generate interrupts upon probing are not catered for.

A skeleton Makefile for building probe routines, /usr/lib/probe/splprobe.mk is provided to simplify the process. After the source for a probe routine is built following the above example, edit a copy of the Makefile and add the name of the probe routine to the line starting with the word PROBE. The make(1) utility can then be used to have the probe built.

FILES

`/usr/lib/probe/splprobe.mk`

SEE ALSO

`make`(1), `boot`(1M), `edt_data`(4).

NAME

egrep - search a file for a pattern using full regular expressions

SYNOPSIS

egrep [-bchilnc] -e *special_expr* | -f *expr_file* | *full_regular_expression* [*file* . . .]

DESCRIPTION

egrep (expression grep) searches files for a pattern of characters and prints all lines that contain that pattern. egrep uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

egrep accepts the same full regular expressions accepted by ed, with six exceptions:

```
\(    \<    \{m
\)    \>    n\}
```

(The regular expressions \(and \) should not be confused with parentheses used for grouping.) In addition, egrep accepts the following expressions:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a newline that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses ( ) for grouping.

Be careful using the characters $, *, [, ^, |, (,), and \ in *full_regular_expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full_regular_expression* in single quotes ´ . . . ´.

The order of precedence of operators is [ ], then * ? +, then concatenation, then | and newline.

If no files are specified, egrep assumes standard input. Normally, each line found is copied to the standard output. The filename is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
-c Print only a count of the lines that contain the pattern.
-i Ignore uppercase/lowercase distinction during comparisons. This option is valid only for single byte characters.
-h Suppress printing of filenames when searching multiple files.
-l Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
-n Precede each line by its line number in the file (first line is 1).

`-v` Print all lines except those that contain the pattern.

`-e` *special_expr*
Search for a *special_expr* (*full_regular_expression* that begins with a `-`).

`-f` *expr_file*
Take the list of *full_regular_expressions* from *expr_file*.

INTERNATIONAL FUNCTIONS

`egrep` can process characters from supplementary code sets. In regular expressions, searches are performed on characters, not on individual bytes.

SEE ALSO

`ed`(1), `fgrep`(1), `grep`(1), `sed`(1), `sh`(1)

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).

NOTES

Ideally there should be only one `grep` command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to `BUFSIZ` characters; longer lines are truncated. `BUFSIZ` is defined in `/usr/include/stdio.h`.

NAME

`enable`, `disable` - enable/disable LP printers

SYNOPSIS

`enable` *printers*
`disable` [*options*] *printers*

DESCRIPTION

The `enable` command activates the named *printers*, enabling them to print requests taken by the `lp` command. If the printer is remote, the command will only enable the transfer of requests to the remote system; the `enable` command must be run again, on the remote system, to activate the printer. (Run `lpstat -p` to get the status of printers.)

The `disable` command deactivates the named *printers*, disabling them from printing requests taken by `lp`. By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class of printers. If the printer is remote, this command will only stop the transmission of jobs to the remote system. The `disable` command must be run on the remote system to disable the printer. (Run `lpstat -p` to get the status of printers.) Options for use with `disable` are:

`-c` Cancel any requests that are currently printing on any of the designated printers. This option cannot be used with the `-W` option. If the printer is remote, the `-c` option will be silently ignored.

`-r` *reason* Assign a *reason* for the disabling of the printers. This *reason* applies to all printers mentioned. This *reason* is reported by `lpstat -p`. If the `-r` option is not present, then a default reason will be used.

`-W` Wait until the request currently being printed is finished before disabling the specified printer. This option cannot be used with the `-c` option. If the printer is remote, the `-W` option will be silently ignored.

FILES

`/var/spool/lp/*`

SEE ALSO

`lp`(1), `lpstat`(1).

NAME

env - set environment for command execution

SYNOPSIS

env [-] [*name=value*] . . . [*command args*]

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 - flag causes the inherited environment to be ignored completely so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed one name-value pair per line.

INTERNATIONAL FUNCTIONS

Characters from supplementary code sets can be used for *value*, *command* and *args*.

SEE ALSO

sh(1), exec(2), profile(4), environ(5).

NAME

environ - user environment

DESCRIPTION

When a process begins execution, exec routines make available an array of strings called the environment [see exec(2)]. By convention, these strings have the form *variable=value*, for example, PATH=/sbin:/usr/sbin. These environmental variables provide a way to make information about a program's environment available to programs. The following environmental variables can be used by applications and are expected to be set in the target run-time environment.

HOME — The name of the user's login directory, set by login(1) from the password file (see passwd(4)).

LANG — The string used to specify localization information that allows users to work with different national conventions. The setlocale(3C) function looks for the LANG environment variable when it is called with "" as the *locale* argument. LANG is used as the default locale if the corresponding environment variable for a particular category is unset.

For example, when setlocale() is invoked as

```
setlocale(LC_CTYPE, ""),
```

setlocale() will query the LC_CTYPE environment variable first to see if it is set and non-null. If LC_CTYPE is not set or null, then setlocale() will check the LANG environment variable to see if it is set and non-null. If both LANG and LC_CTYPE are unset or null, the default C locale will be used to set the LC_CTYPE category.

Most commands will invoke

```
setlocale(LC_ALL, "")
```

prior to any other processing. This allows the command to be used with different national conventions by setting the appropriate environment variables.

The following environment variables are supported to correspond with each category of setlocale(3C):

LC_COLLATE — This category specifies the collation sequence being used. The information corresponding to this category is stored in a database created by the colltbl(1M) command. This environment variable affects strcoll(3C) and strxfrm(3C).

LC_CTYPE — This category specifies character classification, character conversion, and widths of multibyte characters. The information corresponding to this category is stored in a database created by the chrtbl(1M) command. The default C locale corresponds to the 7-bit ASCII character set. This environment variable is used by ctype(3C), mbchar(3C), and many commands; for example: cat(1), ed(1), ls(1), and vi(1).

LC_MESSAGES This category specifies the language of the message database being used. For example, an application may have one message database with French messages, and another database with German messages. Message databases are created by the `mkmsgs`(1M) command. This environment variable is used by `exstr`(1), `gettxt`(1), `gettxt`(3C), and `srchtxt`(1).

LC_MONETARY This category specifies the monetary symbols and delimiters used for a particular locale. The information corresponding to this category is stored in a database created by the `montbl`(1M) command. This environment variable is used by `localeconv`(3C).

LC_NUMERIC This category specifies the decimal and thousands delimiters. The information corresponding to this category is stored in a database created by the `chrtbl`(1M) command. The default `C` locale corresponds to "." as the decimal delimiter and no thousands delimiter. This environment variable is used by `localeconv`(3C), `printf`(3C), and `strtod`(3C).

LC_TIME This category specifies date and time formats. The information corresponding to this category is stored in a database specified in `strftime`(4). The default `C` locale corresponds to U.S. date and time formats. This environment variable is used by many commands and functions; for example: `at`(1), `calendar`(1), `date`(1), `strftime`(3C), and `getdate`(3C).

MSGVERB Controls which standard format message components `fmtmsg` selects when messages are displayed to `stderr` [see `fmtmsg`(1) and `fmtmsg`(3C)].

SEV_LEVEL Define severity levels and associate and print strings with them in standard format error messages [see `addseverity`(3C), `fmtmsg`(1), and `fmtmsg`(3C)].

NETPATH A colon-separated list of network identifiers. A network identifier is a character string used by the Network Selection component of the system to provide application-specific default network search paths. A network identifier must consist of non-NULL characters and must have a length of at least 1. No maximum length is specified. Network identifiers are normally chosen by the system administrator. A network identifier is also the first field in any `/etc/netconfig` file entry. `NETPATH` thus provides a link into the `/etc/netconfig` file and the information about a network contained in that network's entry. `/etc/netconfig` is maintained by the system administrator. The library routines described in `getnetpath`(3N) access the `NETPATH` environment variable.

NLSPATH Contains a sequence of templates which catopen(3C) uses when attempting to locate message catalogs. Each template consists of an optional prefix, one or more substitution fields, a filename and an optional suffix.

For example:

```
NLSPATH="/system/nlslib/%N.cat"
```

defines that catopen() should look for all message catalogs in the directory /system/nlslib, where the catalog name should be constructed from the *name* parameter passed to catopen(), %N, with the suffix .cat.

Substitution fields consist of a % symbol, followed by a single-letter keyword. The following keywords are currently defined:

%N	The value of the *name* parameter passed to catopen().
%L	The value of LANG.
%l	The language element from LANG.
%t	The territory element from LANG.
%c	The codeset element from LANG.
%%	A single % character.

An empty string is substituted if the specified value is not currently defined. The separators "_" and "." are not included in %t and %c substitutions.

Templates defined in NLSPATH are separated by colons (:). A leading colon or two adjacent colons (::) is equivalent to specifying %N.

For example:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

indicates to catopen() that it should look for the requested message catalog in *name*, *name*.cat and /nlslib/$LANG/*name*.cat.

PATH The sequence of directory prefixes that sh(1), time(1), nice(1), nohup(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). login(1) sets PATH=/usr/bin. (For more detail, see sh(1).)

TERM The kind of terminal for which output is to be prepared. This information is used by commands, such as mm(1) or vi(1), which may exploit special capabilities of that terminal.

TZ Time zone information.
The contents of the environment variable named TZ are used by the functions ctime(3C), localtime() (see ctime(3C)), strftime(3C) and mktime(3C) to override the default timezone. If the first character of TZ is a colon (:), the behavior is implementation defined, otherwise TZ has the form:

std offset [*dst* [*offset*] , [*start* [/*time*] , *end* [/*time*]]]

where:

std and *dst*
: Three or more bytes that are the designation for the standard (*std*) and daylight savings time (*dst*) timezones. Only *std* is required, if *dst* is missing, then daylight savings time does not apply in this locale. Upper- and lower-case letters are allowed. Any characters except a leading colon (:), digits, a comma (,), a minus (-) or a plus (+) are allowed.

offset
: Indicates the value one must add to the local time to arrive at Coordinated Universal Time. The offset has the form:

  hh [: *mm* [: *ss*]]

  The minutes (*mm*) and seconds (*ss*) are optional. The hour (*hh*) is required and may be a single digit. The *offset* following *std* is required. If no *offset* follows *dst* , daylight savings time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal number. The hour must be between 0 and 24, and the minutes (and seconds) if present between 0 and 59. Out of range values may cause unpredictable behavior. If preceded by a "-", the timezone is east of the Prime Meridian; otherwise it is west (which may be indicated by an optional preceding "+" sign).

start/time,end/time
: Indicates when to change to and back from daylight savings time, where *start/time* describes when the change from standard time to daylight savings time occurs, and *end/time* describes when the change back happens. Each *time* field describes when, in current local time, the change is made.

  The formats of *start* and *end* are one of the following:

  `J`*n*
  : The Julian day *n* ($1 \le n \le 365$). Leap days are not counted. That is, in all years, February 28 is day 59 and March 1 is day 60. It is impossible to refer to the occasional February 29.

  n
  : The zero-based Julian day ($0 \le n \le 365$). Leap days are counted, and it is possible to refer to February 29.

  `M`*m.n.d*
  : The d^{th} day, ($0 \le d \le 6$) of week *n* of month *m* of the year ($1 \le n \le 5$, $1 \le m \le 12$), where week 5 means "the last *d*-day in month *m*" which may occur in either the fourth or the fifth week). Week 1 is the first week in which the d^{th} day occurs. Day zero is Sunday.

Implementation specific defaults are used for *start* and *end* if these optional fields are not given.

The *time* has the same format as *offset* except that no leading sign ("-" or "+") is allowed. The default, if *time* is not given is 02:00:00.

Further names may be placed in the environment by the `export` command and *name=value* arguments in `sh`(1), or by `exec`(2). It is unwise to conflict with certain shell variables that are frequently exported by `.profile` files: `MAIL`, `PS1`, `PS2`, `IFS` (see `profile`(4)).

SEE ALSO

`cat`(1), `chrtbl`(1M), `colltbl`(1M), `date`(1), `ed`(1), `fmtmsg`(1), `ls`(1), `login`(1), `mkmsgs`(1M), `mm`(1), `montbl`(1M), `nice`(1), `nohup`(1), `sh`(1), `sort`(1), `time`(1), `vi`(1), `exec`(2), `addseverity`(3C), `catopen`(3C), `ctime`(3C), `ctype`(3C), `fmtmsg`(3C), `getdate`(3C), `getnetpath`(3N), `gettxt`(3C), `localeconv`(3C), `mbchar`(3C), `mktime`(3C), `printf`(3C), `strcoll`(3C), `strftime`(3C), `strtod`(3C), `strxfrm`(3C), `netconfig`(4), `passwd`(4), `profile`(4), `strftime`(4), `timezone`(4).

NAME

envmon - add /dev entries for the environmental monitor board in the Equipped Device Table

SYNOPSIS

/sbin/auto-device/envmon

DESCRIPTION

envmon searches the EDT to see if an ENVMON board is equipped. If one is found, the device nodes are created. Otherwise, they are deleted.

envmon is called each time the system is re-configured.

NOTES

The environmental monitor board is supported on the m88k architecture only.

FILES

/dev/envmon_c0 /dev/xedt/envmon_c0

SEE ALSO

makedev(1M), intro(7)

NAME

eqn, neqn, checkeq - typeset mathematics

SYNOPSIS

/usr/ucb/eqn [-d*xy*] [-f*n*] [-p*n*] [-s*n*] [*filename*] . . .

/usr/ucb/neqn [*filename*] . . .

/usr/ucb/checkeq [*filename*] . . .

DESCRIPTION

The eqn and neqn commands are language processors to assist in describing equations. eqn is a preprocessor for troff(1) and is intended for devices that can print troff's output. neqn is a preprocessor for nroff(1) and is intended for use with terminals.

checkeq reports missing or unbalanced delimiters and .EQ/.EN pairs.

If no *filename*s are specified, eqn and neqn read 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 set two characters as "delimiters"; subsequent text between delimiters is also treated as eqn input.

The following options are available for eqn and neqn:

-d*xy* Set equation delimiters set to characters *x* and *y* with the command-line argument. The more common way to do this is with delim*xy* between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by delim off appearing in the text. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

-f*n* Change font to *n* globally in the document. The font can also be changed globally in the body of the document by using the gfont directive.

-p*n* Reduce subscripts and superscripts by *n* point sizes from the previous size. In the absence of the -p option, subscripts and superscripts are reduced by 3 point sizes from the previous size.

-s*n* Set equations in point size *n* globally in the document. The point size can also be changed globally in the body of the document by using the gsize directive.

-T*dev* Prepare output for device *dev*. If no -T option is present, eqn looks at the environment variable TYPESETTER to see what the intended output device is. If no such variable is found in the environment, a system-dependent default device is assumed. Not available using neqn.

USAGE

eqn Language

Tokens within eqn are separated by braces, double quotes, tildes, circumflexes, SPACE, TAB, or NEWLINE characters. Braces { } are used for grouping; generally speaking, anywhere a single character like *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 i`' makes x_i, '`a sub i sup 2`' produces a_i^2, and '`e sup {x sup 2 + y sup 2}`' gives $e^{x^2+y^2}$.

Fractions are made with `over`: '`a over b`' yields $\frac{a}{b}$.

`sqrt` makes square roots: '`1 over down 10 sqrt {ax sup 2 +bx+c}`' results in

$$\frac{1}{\sqrt{ax^2+bx+c}}.$$

Although `eqn` tries to get most things at the right place on the paper, occasionally you will need to tune the output to make it just right. In the previous example, a local motion, *down 10* was used to get more space between the square root and the line above it.

The keywords `from` and `to` introduce lower and upper limits on arbitrary things: $\lim_{n\to\infty} \sum_{0}^{n} x_i$ is made with '`lim from {n-> inf } sum from 0 to n x sub i`'.

Left and right brackets, braces, etc., of the right height are made with `left` and `right`: '`left [ x sup 2 + y sup 2 over alpha right ] ~=~1`' produces

$$\left[x^2+\frac{y^2}{\alpha}\right] = 1.$$

The `right` clause is optional. 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).

Vertical piles of things are made with `pile`, `lpile`, `cpile`, and `rpile`: '`pile {a above b above c}`' produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. `lpile` left-justifies, `pile` and `cpile` center, 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 $\ddot{\underline{y}} = \underline{n}$, and '`x vec ~=~ y dyad`' is $\vec{x} = \overleftrightarrow{y}$.

Sizes and font can be changed with `size` *n* or `size` ±*n*, `roman`, `italic`, `bold`, and `font` *n*. Size 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*.

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* which will be replaced by *replacement* whenever it appears thereafter. The `%` may be any character that does not occur in *replacement*.

Keywords like `sum` (∑), `int` (∫), `inf` (∞), and shorthands like `>=` (≥), `->` (→), and `!=` (≠) are recognized. Greek letters are spelled out in the desired case, as in `alpha` or `GAMMA`. Mathematical words like sin, cos, and log are made Roman automatically. `troff`(1) four-character escapes like \(bu (•) can 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 `troff` when all else fails.

EXAMPLE

`eqn` *filename* . . . | `troff`

`neqn` *filename* . . . | `nroff`

SEE ALSO

`tbl`(1), `troff`(1), `eqnchar`(5), `ms`(5)

NOTES

To embolden digits, parens, etc., it is necessary to quote them, as in `bold "12.3"`.

NAME

eqnchar - special character definitions for eqn

SYNOPSIS

eqn /usr/ucblib/pub/eqnchar [*filename*] | troff [*options*]

neqn /usr/ucblib/pub/eqnchar [*filename*] | nroff [*options*]

DESCRIPTION

The eqnchar command contains troff(1) and nroff(1) character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with eqn(1) and neqn. It contains definitions for the following characters:

ciplus	⊕	| |	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	□
-wig	≈	*hbar*	ħ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	<->	↔	*empty*	∅
=wig	≅	<=>	⇔	*member*	∈
star	*	| <	≮	*nomem*	∉
bigstar	✳	| >	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	⊦	*incl*	⫝
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≙	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°		

FILES

/usr/ucblib/pub/eqnchar

SEE ALSO

eqn(1), nroff(1), troff(1)

NAME

eucset - set or get EUC code set widths

SYNOPSIS

`eucset` [*cswidth*]

`eucset -p`

DESCRIPTION

`eucset` assumes the existence of an line discipline which does canonical processing of EUC character in its *standard input STREAM* (usually a TTY). The line discipline must recognize the *eucioc* calls to `ioctl()`, as defined in the header file `/usr/include/sys/eucioctl.h`.

If given no arguments, `eucset` looks in the environment for the **cswidth** parameter in the *character class table,* which is assumed to specify the code Set widths and Screen widths in use. The format of **cswidth** parameter is described in *character class table* specification.

If given one argument which does not begin with "`-`", it is taken to be a string in the format of **cswidth** parameter, overriding whatever is in the environment.

If given the optional argument `-p`, `eucset` prints the current values of the *code set widths* and *screen widths* as returned by the line discipline. These values may be different than what is currently in the user's environment, but represent the EUC mapping that the EUC line discipline is currently using. The primary code set (ASCII) is excluded from the listing, which is in the same format as **cswidth** parameter.

RETURN VALUES

`eucset` returns `0` on success, `1` for failure of any call to `ioctl()`.

FILES

`/usr/include/sys/eucioctl.h`

`/usr/include/sys/euc.h`

SEE ALSO

`ioctl`(2), `getwidth`(3W), `ldterm`(7), `streamio`(7).

NAME

ex - text editor

SYNOPSIS

ex [-s] [-v] [-t *tag*] [-r *file*] [-L] [-R] [-x] [-C] [-c *command*] *file* ...

DESCRIPTION

ex is the root of a family of editors: ex and vi. ex is a superset of ed, with the most notable extension being a display editing facility. Display based editing is the focus of vi.

If you have a CRT terminal, you may wish to use a display based editor; in this case see vi(1), which is a command which focuses on the display-editing portion of ex.

For ed Users

If you have used ed you will find that, in addition to having all of the ed commands available, ex has a number of additional features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with vi. Generally, the ex editor uses far more of the capabilities of terminals than ed does, and uses the terminal capability data base [see terminfo(4)] and the type of the terminal you are using from the environmental variable TERM to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its visual command (which can be abbreviated vi) and which is the central mode of editing when using the vi command.

ex contains a number of features for easily viewing the text of the file. The z command gives easy access to windows of text. Typing ^D (control-d) causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just typing return. Of course, the screen-oriented visual mode gives constant access to editing context.

ex gives you help when you make mistakes. The undo (u) command allows you to reverse any single change which goes astray. ex gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files, unless you edited them, so that you do not accidentally overwrite a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor recover command (or -r *file* option) to retrieve your work. This will get you back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the next (n) command to deal with each in turn. The next command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

The editor has a group of buffers whose names are the ASCII lower-case letters (a-z). You can place text in these named buffers where it is available to be inserted elsewhere in the file. The contents of these buffers remain available when you begin editing a new file using the edit (e) command.

There is a command & in ex which repeats the last substitute command. In addition, there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore the case of letters in searches and substitutions. ex also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

ex has a set of options which you can set to tailor it to your liking. One option which is very useful is the autoindent option that allows the editor to supply leading white space to align text automatically. You can then use ^D as a backtab and space or tab to move forward to align new code easily.

Miscellaneous useful features include an intelligent join (j) command that supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as sort.

Invocation Options

The following invocation options are interpreted by ex (previously documented options are discussed in the **NOTES** section at the end of this manual page):

-s	Suppress all interactive-user feedback. This is useful in processing editor scripts.
-v	Invoke vi.
-t *tag*	Edit the file containing the *tag* and position the editor at its definition.
-r *file*	Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)
-L	List the names of all files saved as the result of an editor or system crash.
-R	Readonly mode; the readonly flag is set, preventing accidental overwriting of the file.
-x	Encryption option; when used, ex simulates an X command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of the crypt command. The X command makes an educated guess to determine whether text read in is encrypted or not. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the -x option. See crypt(1). Also, see the **NOTES** section at the end of this manual page.
-C	Encryption option; the same as the -x option, except that ex simulates a C command. The C command is like the X command, except that all text read in is assumed to have been encrypted.
-c *command*	Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

ex States

Command	Normal and initial state. Input prompted for by `:`. Your line kill character cancels a partial command.
Insert	Entered by `a`, `i`, or `c`. Arbitrary text may be entered. Insert state normally is terminated by a line having only "." on it, or, abnormally, with an interrupt.
Visual	Entered by typing `vi`; terminated by typing `Q` or `^\` (control-\).

ex Command Names and Abbreviations

abbrev	ab	map		set	se
append	a	mark	ma	shell	sh
args	ar	move	m	source	so
change	c	next	n	substitute	s
copy	co	number	nu	unabbrev	unab
delete	d	preserve	pre	undo	u
edit	e	print	p	unmap	unm
file	f	put	pu	version	ve
global	g	quit	q	visual	vi
insert	i	read	r	write	w
join	j	recover	rec	xit	x
list	l	rewind	rew	yank	ya

ex Commands

forced encryption	C	heuristic encryption	X
resubst	&	print next	CR
rshift	>	lshift	<
scroll	^D	window	z
shell escape	!		

ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
-	previous	*'x*	marked with *x*
+n	*n* forward	''	previous context
%	1,$		

Initializing options

`EXINIT`	place `set`'s here in environment variable
`$HOME/.exrc`	editor initialization file
`./.exrc`	editor initialization file
`set` *x*	enable option *x*
`set no`*x*	disable option *x*
`set` *x*`=val`	give value *val* to option *x*
`set`	show changed options

`set all`	show all options
`set` *x*`?`	show value of option *x*

Most useful options and their abbreviations

autoindent	`ai`	supply indent
autowrite	`aw`	write before changing files
directory		pathname of directory for temporary work files
exrc	`ex`	allow `vi/ex` to read the `.exrc` in the current directory. This option is set in the `EXINIT` shell variable or in the `.exrc` file in the `$HOME` directory.
ignorecase	`ic`	ignore case of letters in scanning
list		print `^I` for tab, $ at end
magic		treat `.` `[` `*` special in patterns
modelines		first five lines and last five lines executed as `vi/ex` commands if they are of the form `ex:`*command*`:` or `vi:`*command*`:`
number	`nu`	number lines
paragraphs	`para`	macro names that start paragraphs
redraw		simulate smart terminal
report		informs you if the number of lines modified by the last command is greater than the value of the `report` variable
scroll		command mode lines
sections	`sect`	macro names that start sections
shiftwidth	`sw`	for `<` `>`, and input `^D`
showmatch	`sm`	to `)` and `}` as typed
showmode	`smd`	show insert mode in `vi`
slowopen	`slow`	stop updates during insert
term		specifies to `vi` the type of terminal being used (the default is the value of the environmental variable `TERM`)
window		visual mode lines
wrapmargin	`wm`	automatic line splitting
wrapscan	`ws`	search around end (or beginning) of buffer

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[`str`]	any character in *str*
[`^str`]	any character not in *str*
[*x*-*y*]	any character between *x* and *y*
*	any number of preceding characters

AUTHOR

vi and ex are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/exstrings	error messages
/usr/lib/exrecover	recover command
/usr/lib/expreserve	preserve command
/usr/share/lib/terminfo/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/var/preserve/login	preservation directory (where login is the user's login)

NOTES

Several options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard [see intro(1)]. The - option has been replaced by -s, a -r option that is not followed with an option-argument has been replaced by -L, and +*command* has been replaced by -c *command*.

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

The z command prints the number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line -s option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

INTERNATIONAL FUNCTIONS

ex can process characters from supplementary code sets as well as ASCII characters.

In regular expressions, searches and pattern matching are performed in character units, not in individual bytes.

SEE ALSO

crypt(1), ed(1), edit(1), grep(1), sed(1), sort(1), vi(1), curses(3X), term(4), terminfo(4).

NAME

exportfs - export and unexport directories to NFS clients

SYNOPSIS

/usr/lib/nfs/exportfs [-aiuv] [-o *options*] [*pathname*]

DESCRIPTION

exportfs makes a local directory or filename available for mounting over the network by NFS clients. It uses information contained in the /etc/dfs/dfstab file to export *pathname* (which must be specified as a full pathname). The user with appropriate administrative privileges can run exportfs at any time to alter the list or characteristics of exported directories and filenames. Directories and files that are currently exported are listed in the file /etc/dfs/sharetab.

With no options or arguments, exportfs prints out the list of directories and filenames currently exported.

OPTIONS

-a All. Export all pathnames listed in /etc/dfs/dfstab, or if -u is specified, unexport all of the currently exported pathnames.

-i Ignore the options in /etc/dfs/dfstab. Normally, exportfs will consult /etc/exports for the options associated with the exported pathname.

-u Unexport the indicated pathnames.

-v Verbose. Print each directory or filename as it is exported or unexported.

-o *options*

Specify a comma-separated list of optional characteristics for the pathname being exported. *options* can be selected from among:

ro Export the pathname read-only. If not specified, the pathname is exported read-write.

rw= *hostname*[: *hostname*] . . .

Export the pathname read-mostly. Read-mostly means exported read-only to most machines, but read-write to those specified. If not specified, the pathname is exported read-write to all.

anon= *uid*

If a request comes from an unknown user, use UID as the effective user. ID. Note: root users (UID 0) are always considered unknown by the NFS server, unless they are included in the root option below. The default value for this option is -2. Setting the value of anon to -1 disables anonymous access. Note: by default secure NFS accepts insecure requests as anonymous, and those wishing for extra security can disable this feature by setting anon to -1.

root= *hostname*[: *hostname*] . . .

Give root access only to the root users from a specified *hostname.* The default is for no hosts to be granted root access.

access= *client*[: *client*] . . .

Give mount access to each *client* listed.

`secure`
Require clients to use a more secure protocol when accessing the directory.

FILES

`/etc/dfs/dfstab` static export information
`/etc/dfs/sharetab` current state of exported pathnames
`/etc/netgroup`

SEE ALSO

`showmount`(1M)

NOTES

You cannot export a directory that is either a parent- or a sub-directory of one that is currently exported and within the same filesystem. It would be illegal, for example, to export both `/usr` and `/usr/local` if both directories resided in the same disk slice.

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 length of the expression is limited to 512 characters.

The operators and keywords are listed below. Characters that need to be escaped in the shell [see `sh`(1)] 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.

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. Regular expression syntax is the same as that of `ed`(1), 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 bytes matched (`0` on failure). Alternatively, the `\( . . . \)` pattern symbols can be used to return a portion of the first argument.

EXAMPLES

Add 1 to the shell variable `a`:

```
a=`expr $a + 1`
```

The following example emulates `basename`(1)—it returns the last segment of the path name `$a`. For `$a` equal to either `/usr/abc/file` or just `file`, the example

returns `file`. (Watch out for `/` alone as an argument: `expr` takes it as the division operator. See the **NOTES** below.)

```
expr $a : '.*/\(.*\)' \| $a
```

The following is another version of the previous example. The addition of the `//` characters eliminates any ambiguity about the division operator and simplifies the whole expression.

```
expr //$a : '.*/\(.*\)'
```

Return the number of characters in `$VAR`:

```
expr $VAR : '.*'
```

INTERNATIONAL FUNCTIONS

`expr` can process characters from supplementary code sets in addition to ASCII characters. In regular expressions, pattern searches are performed on characters, not bytes.

`:` returns the matched size in bytes, not in characters.

SEE ALSO

`ed`(1), `sh`(1).

DIAGNOSTICS

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

`syntax error`	for operator/operand errors
`non-numeric argument`	if arithmetic is attempted on such a string

NOTES

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 are all taken as the `=` operator). The following works:

```
expr X$a = X=
```

NAME

exstr - extract strings from source files

SYNOPSIS

exstr *file* . . .
exstr -e *file* . . .
exstr -r [-d] *file* . . .

DESCRIPTION

The exstr utility is used to extract strings from C language source files and replace them by calls to the message retrieval function [see gettxt(3C)]. This utility will extract all character strings surrounded by double quotes, not just strings used as arguments to the printf command or the printf routine. In the first form, exstr finds all strings in the source files and writes them on the standard output. Each string is preceded by the source file name and a colon. The meanings of the options are:

-e Extract a list of strings from the named C language source files, with positional information. This list is produced on standard output in the following format:

file:line:position:msgfile:msgnum:string

file	the name of a C language source file
line	line number in the file
position	character position in the line
msgfile	null
msgnum	null
string	the extracted text string

Normally you would redirect this output into a file. Then you would edit this file to add the values you want to use for *msgfile* and *msgnum*:

msgfile the file that contains the text strings that will replace *string*. A file with this name must be created and installed in the appropriate place by the mkmsgs(1) utility.

msgnum the sequence number of the string in *msgfile*.

The next step is to use exstr -r to replace *string*s in *file*.

-r Replace strings in a C language source file with function calls to the message retrieval function gettxt.

-d This option is used together with the -r option. If the message retrieval fails when gettxt is invoked at run time, then the extracted string is printed.

You would use the capability provided by exstr on an application program that needs to run in an international environment and have messages print in more than one language. exstr replaces text strings with function calls that point at strings in a message database. The database used depends on the runtime value of the LC_MESSAGES environment variable [see environ(5)].

The first step is to use exstr -e to extract a list of strings and save it in a file. Next, examine this list and determine which strings can be translated and subsequently retrieved by the message retrieval function. Then, modify this file by deleting lines that can't be translated and, for lines that can be translated, by adding the message

file names and the message numbers as the fourth (*msgfile*) and fifth (*msgnum*) entries on a line. The message files named must have been created by `mkmsgs`(1) and exist in `/usr/lib/locale/`*locale*`/LC_MESSAGES`. The directory *locale* corresponds to the language in which the text strings are written [see `setlocale`(3C)]. The message numbers used must correspond to the sequence numbers of strings in the message files.

Now use this modified file as input to `exstr -r` to produce a new version of the original C language source file in which the strings have been replaced by calls to the message retrieval function `gettxt`. The *msgfile* and *msgnum* fields are used to construct the first argument to `gettxt`. The second argument to `gettxt` is printed if the message retrieval fails at run time. This argument is the null string, unless the `-d` option is used.

This utility cannot replace strings in all instances. For example, a static initialized character string cannot be replaced by a function call, or a string could be in the form of an escape sequence that cannot be translated. In order not to break existing code, the files created by invoking `exstr -e` must be examined and lines containing strings not replaceable by function calls must be deleted. In some cases the code may require modifications so that strings can be extracted and replaced by calls to the message retrieval function.

EXAMPLES

The following examples show uses of `exstr`.

Assume that the file `foo.c` contains two strings:

```
main()
{
     printf("This is an example\n");
     printf("Hello world!\n");
}
```

The `exstr` utility, invoked with the argument `foo.c`, extracts strings from the named file and prints them on the standard output.

`exstr foo.c` produces the following output:

```
foo.c:This is an example\n
foo.c:Hello world!\n
```

`exstr -e foo.c > foo.stringsout` produces the following output in the file `foo.stringsout`:

```
foo.c:3:8:::This is an example\n
foo.c:4:8:::Hello world!\n
```

You must edit `foo.stringsout` to add the values you want to use for the *msgfile* and *msgnum* fields before these strings can be replaced by calls to the retrieval function. If `UX` is the name of the message file, and the numbers `1` and `2` represent the sequence number of the strings in the file, here is what `foo.stringsout` looks like after you add this information:

```
foo.c:3:8:UX:1:This is an example\n
foo.c:4:8:UX:2:Hello world!\n
```

The exstr utility can now be invoked with the -r option to replace the strings in the source file by calls to the message retrieval function gettxt.

exstr -r foo.c <foo.stringsout >intlfoo.c produces the following output:

```
extern char *gettxt();
main()
{
     printf(gettxt("UX:1", ""));
     printf(gettxt("UX:2", ""));
}
```

exstr -rd foo.c <foo.stringsout >intlfoo.c uses the extracted strings as a second argument to gettxt.

```
extern char *gettxt();
main()
{
     printf(gettxt("UX:1", "This is an example\n"));
     printf(gettxt("UX:2", "Hello world!\n"));
}
```

FILES

/usr/lib/locale/*locale*/LC_MESSAGES/* files created by mkmsgs(1)

SEE ALSO

gettxt(1), mkmsgs(1), printf(1), srchtxt(1), gettxt(3C), printf(3S), setlocale(3C), environ(5)

DIAGNOSTICS

The error messages produced by exstr are intended to be self-explanatory. They indicate errors in the command line or format errors encountered within the input file.

NAME

face - executable for the Framed Access Command Environment Interface

SYNOPSIS

face [-i *init_file*] [-c *command_file*] [-a *alias_file*] [*file* . . .]

DESCRIPTION

file is the full pathname of the file describing the object to be opened initially, and must follow the naming convention Menu.*xxx* for a menu, Form.*xxx* for a form, and Text.*xxx* for a text file, where *xxx* is any string that conforms to the UNIX system file naming conventions. The FMLI descriptor lifetime will be ignored for all frames opened by argument to face. These frames have a lifetime of immortal by default. If *file* is not specified on the command line, the FACE Menu will be opened along with those objects specified by the LOGINWIN environment variables. These variables are found in the user's .environ file.

FILES

$HOME/pref/.environ

SEE ALSO

env(4)

DIAGNOSTICS

The face command will exit with a non-zero exit code if the user is not properly set up as a FACE user.

NAME

`factor` - obtain the prime factors of a number

SYNOPSIS

`factor` [*integer*]

DESCRIPTION

When you use `factor` without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. `factor` exits if it encounters a zero or any non-numeric character.

If you invoke `factor` with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. `factor` will take this time when n is prime or the square of a prime.

DIAGNOSTICS

`factor` prints the error message, `Ouch`, for input out of range or for garbage input.

NAME

`fastboot`, `fasthalt` - reboot/halt the system without checking the disks

SYNOPSIS

`/usr/ucb/fastboot` [*boot-options*]

`/usr/ucb/fasthalt` [*halt-options*]

DESCRIPTION

`fastboot` and `fasthalt` are shell scripts that invoke `reboot` and `halt` with the proper arguments.

These commands are provided for compatibility only.

FILES

`/etc/rc`

SEE ALSO

`halt`(1M), `reboot`(1M), `fsck`(1M), `init`(1M), `rc0`(1M), `rc2`(1M), `rc6`(1M).

NAME

`fdetach` - detach a name from a STREAMS-based file descriptor

SYNOPSIS

```
fdetach path
```

DESCRIPTION

The `fdetach` command detaches a STREAMS-based file descriptor from a name in the file system. *path* is the path name of the object in the file system name space, which was previously attached [see `fattach`(3C)]. The user must be the owner of the file or a user with the appropriate privileges. All subsequent operations on *path* will operate on the file system node and not on the STREAMS file. The permissions and status of the node are restored to the state the node was in before the STREAMS file was attached to it.

SEE ALSO

`fattach`(3C), `fdetach`(3C), `streamio`(7).

NAME

fdp - create, or restore from, a full file system archive

SYNOPSIS

fdp -B [-dovAENS] [-c *count*] *bkjobid odpname odpdev odplab descript*

fdp -RC [-dovAENS] [-c *count*] *odpname odpdev redpname redev rsjobid descript*

DESCRIPTION

The fdp command is invoked as a child process by other shell commands. The command name, fdp, is read either from the *bkhist.tab* file or the bkreg -m command and option. The -B, -R, and -C options are passed to fdp by the shell commands backup, and restore. The other options are passed from the bkhist.tab file or the bkreg -p command and option. The arguments are sent to fdp from various locations in the backup service.

fdp -B is invoked as a child process by the backdaemon command to perform a backup of the data slice *odpdev* (the originating data slice). All blocks in the data slice are archived. The resulting backup is created in the format described on dd(1). The backup is recorded in the backup history log, /etc/bkup/bkhist.tab.

fdp -RC is invoked as a child process by the rsoper command to restore the entire data slice from an archive created by fdp -B. The data slice archive is assumed to be in the format described on dd(1). *dd(1)* format.

The arguments to fdp are defined as follows:

bkjobid the job id assigned by *backup*. The method uses the *bkjobid* when it creates history log entries.

odpname the name of the data slice that is to be backed up. Unused by fdp, but supplied by backup for command-line compatibility with other archiving methods.

odpdev the name of the block special device on which the data slice resides.

odplab the volume name on the file system [see labelit(1M)]. Unused by fdp, but supplied by backup for command-line compatibility with other archiving methods.

descript is a description for a destination device in the form:

dgroup:dname:dchar:dlabels

dgroup specifies a device group [see devgroup.tab(4)].
dname specifies a particular device name [see device.tab(4)].
dchars specifies characteristics associated with the device. If specified, *dchar* overrides the defaults for the specified device and group. [See device.tab(4) for a further description of device characteristics].
dlabels specifies the volume names for the media to be used for reading or writing the archive.

rsjobid the job id assigned by restore.

redev if non-null, the slice to be restored to instead of *ofsdev*.

redpname unused, but provided for consistency with other methods.

Options

Some options are only significant during `fdp -B` invocations; they are accepted but ignored during `fdp -R` invocations because the command is invoked and options are specified automatically by `restore`. These options are flagged with an asterisk (*).

`c*`*count* Archives or restores only the first *count* (512 byte) blocks of data in the data slice.

`d*` Inhibits recording the archive in the backup history log.

`o` Permits the user to override media insertion requests [see `getvol`(1M) and the description of the `-o` option].

`v*` Validates the archive as it is written. A checksum is computed as the archive is being written; as each medium is completed, it is re-read and the checksum recomputed to verify that each block is readable and correct. If either check fails, the medium is considered unreadable. If `-A` has been specified, the archiving operation fails; otherwise, the operator is prompted to replace the failed medium.

`A` Establishes automated mode, (that is, does not prompt the user to insert or remove media).

`E*` Reports an estimate of media usage for the archive; then performs the backup.

`N*` Reports an estimate of media usage for the archive; does not perform the backup.

`S` Displays a period (.) for every 100 (512 byte) blocks read-from or written-to the archive on the destination device.

User Interactions

The connection between an archiving method and `backup` is more complex than a simple fork/exec or pipe. The `backup` command is responsible for all interactions with the user, either directly, or through the `bkoper` command. Therefore, `fdp` neither reads from standard-input nor writes to standard-output or standard-error. A method library must be used [see `libbrmeth`(3)] to communicate reports (estimates, filenames, periods, status, and so on) to `backup`.

DIAGNOSTICS

The exit codes for `fdp` are the following:

0 successful completion of the task
1 one or more parameters to `fdp` are invalid.
2 an error has occurred which caused `fdp` to fail to complete all portions of its task.

FILES

`/etc/bkup/bkexcept.tab`	lists the files that are to be excluded from an incremental file system backup.
`/etc/bkup/bkhist.tab`	lists the labels of all volumes that have been used for backup operations.
`/etc/bkup/rsstatus.tab`	tracks the status ofall restore requests from users.
`/etc/bkup/bklog`	logs errors generated by the backup methods and the `backup` command
`/etc/bkup/rslog`	logs errors generated by the restore methods and the `restore` command
`$TMP/filelist$$`	temporarily stores a table of contents for a backup archive.

SEE ALSO

`backup`(1M), `device.tab`(4), `fdp`(1), `ffile`(1), `fimage`(1), `getvol`(1M), `incfile`(1), `labelit`(1M), `libbrmeth`(3), `prtvtoc`(1M), `rsoper`(1M)

NAME

ff (generic) - list file names and statistics for a file system

SYNOPSIS

ff [-F *FSType*] [-V] [*current_options*] [-o *specific_options*] *special* . . .

DESCRIPTION

ff reads the files and directories of the *special* file. I-node data is saved for files which match the selection criteria which is either the *inode* number and/or *inode* age. Output consists of the path name and other file information. Output fields are positional. The output is produced in i-node order. The default line produced by ff is:

path-name i-number

current_options are options supported by the s5-specific module of ff. Other FSTypes do not necessarily support these options. *specific_options* indicate suboptions specified in a comma-separated list of suboptions and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command. See ff_*FSType*(1M) for details.

The options are:

-F Specify the *FSType* on which to operate. The *FSType* should either be specified here or be determinable from /etc/vfstab by matching the *special* with an entry in the table.

-V Echo the complete command line, but do not execute the command. The command line is generated by using the options and arguments provided by the user and adding to them information derived from /etc/vfstab. This option should be used to verify and validate the command line.

-o Specify FSType-specific options.

NOTE

This command may not be supported for all FSTypes.

FILES

/etc/vfstab list of default parameters for each file system

SEE ALSO

ff_s5(1M), ff_ufs(1M), ncheck(1M), vfstab(4).

NAME

`ff` (s5) - display i-list information

SYNOPSIS

`ff` [`-F s5`] [*generic_options*] [`-I`] [`-l`] [`-p`*prefix*] [`-s`] [`-u`] [`-a`*n*] [`-m`*n*] [`-c`*n*] [`-n`*file*] [`-i`*i-node-list*] *special*...

DESCRIPTION

generic_options are options supported by the generic `ff` command.

`ff` reads the i-list and directories of the *special* file, assuming it is an `s5` file system. I-node data is saved for files which match the selection criteria. Output consists of the pathname for each saved i-node, plus other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by `ff` is:

pathname i-number

The pathname is preceded by a . (dot) unless the `-p` option is specified.

The maximum information the command will provide is:

pathname i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where + *n* means more than *n*, - *n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

The options are:

`-F s5` Specifies the `s5`-FSType.

`-I` Do not print the i-node number after each pathname.

`-l` Generate a supplementary list of all pathnames for multiply-linked files.

`-p`*prefix* The specified *prefix* will be added to each generated pathname. The default is . (dot).

`-s` Print the file size, in bytes, after each pathname.

`-u` Print the owner's login name after each pathname.

`-a`*n* Select if the i-node has been accessed in *n* days.

`-m`*n* Select if the i-node has been modified in *n* days.

`-c`*n* Select if the i-node has been changed in *n* days.

`-n`*file* Select if the i-node has been modified more recently than the argument *file*.

`-i`*i-node-list* Generate names for only those i-nodes specified in *i-node-list*. *i-node-list* is a list of numbers separated by commas and without spaces.

NOTE

If the `-l` option is not specified, only a single pathname out of all possible ones is generated for a multiply-linked i-node. If `-l` is specified, all possible names for every linked file on the file system are included in the output. If `-l` and `-i` are both specified, then only the names for linked files matching an i-node listed in the i-node list are displayed.

SEE ALSO

`find`(1), generic `ff`(1M), `ncheck`(1M).

NAME

ff (ufs) - list file names and statistics for a ufs file system

SYNOPSIS

ff [-F ufs] [*generic_options*] [-o a,m,s] *special* ...

DESCRIPTION

generic_options are options supported by the generic ff command.

ff reads the i-list and directories of the *special* file, assuming it is a file system. Inode data is saved for files which match the selection criteria. Output consists of the path name for each saved inode, plus other file information requested using the options below. Output fields are positional. The output is produced in inode order; fields are separated by TAB characters. The default line produced by ff is:

path-name i-number

The options are:

-F ufs
: Specifies the ufs-FSType.

-o
: Specify ufs file system specific options. The options available are:

  a
  : Print the '.' and '..' directory entries.

  m
  : Print mode information.

  s
  : Print only special files and files with set-user-ID mode.

NOTE

If the -l option is not specified, only a single path name out of all possible ones is generated for a multiply-linked inode. If -l is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

SEE ALSO

find(1), generic ff(1M), ncheck(1M).

NAME

ffile - create, or restore from, a full file system archive

SYNOPSIS

ffile -B [-dlmortvAENSV] *bkjobid ofsname ofsdev ofslab descript*

ffile -RC [-dlmortvAENSV] *ofsname ofsdev refsname redev rsjobid descript*

ffile -RF [-dlmortvAENSV] *ofsname ofsdev descript rsjobid:uid:date:type:name* [:[*rename*]:[*inode*]] . . .

DESCRIPTION

The ffile command is invoked as a child process by other shell commands. The command name, ffile, is read either from the bkhist.tab file or the bkreg -m command and option. The -B, -R, -F, and -C options are passed to ffile by the shell commands backup, restore, and urestore. The other options are passed from the bkhist.tab or the bkreg -p command and option. The arguments are sent to ffile from various locations in the backup service.

ffile -B is invoked as a child process by bkdaemon to perform a full backup of the file system *ofsname* (the originating file system). All files in *ofsname* are archived. The resulting backup is created in the format described on cpio(4). The backup is recorded in the backup history log, /usr/oam/bkrs/tables/bkhist.tab.

ffile -RC and RF are invoked as child processes by rsoper to extract files from an full file system archive created by ffile -B. The file system archive is assumed to be in the format described on cpio(4).

If the -RC option is selected, the entire file system is restored.

If the -RF option is specified, only selected objects from the archive are restored. Each 7-tuple, composed of *rsjobid:uid:date:type:name:rename:inode*, specifies an object to be restored from the file system archive. The 7-tuple objects come to ffile from rsstatus.tab.

The arguments to ffile are defined as follows:

bkjobid the job id assigned by backup. The method uses the *bkjobid* when it creates history log and table-of-contents entries.

ofsname the name of the file system that is to be backed up.

ofsdev the name of the block special device on which the file system resides.

ofslab the volume name on the file system [see labelit(1M)].

descript is a description for a destination device in the form:

dgroup:dname:dchar:dlabels

dgroup specifies a device group [see devgroup.tab(4)].
dname specifies a particular device name [see device.tab(4)].
dchars specifies characteristics associated with the device. If specified, *dchar* overrides the defaults for the specified device and group. [See device.tab(4) for a further description of device characteristics.]
dlabels specifies the volume names for the media to be used for reading or writing the archive.

refsname if non-null, the name of the file system to be restored to instead of *ofsname*. At least one of *refsname* and *redev* must be null.

redev if non-null, the slice to be restored to instead of *ofsdev*. At least one of *refsname* and *redev* must be null.

rsjobid the restore jobid assigned by `restore` or `urestore`.

uid the real uid of the user who requested the object to be restored. It must match the uid of the owner of the object at the time the archive was made, or it must be the superuser uid.

date the newest "last modification time" that is acceptable for a restorable object. The object is restored from the archive immediately older than this date. *date* is a hexadecimal representation of the date and time provided by the `time` system call [see `time`(2)].

type either `F` or `D`, indicating that the object is a file or a directory, respectively.

name the name the object had in the file system archive.

rename the name that the object should be restored to (it may differ from the name the object had in the file system archive). If omitted, the object is restored to *name.*

inode the inode number of the object as it was stored in the file system archive. *[inode]* is not used by `ffile -R`, and is provided only for command-line compatibility with other restoration methods.

Options

Some options are only significant during `ffile -B` invocations; they are accepted but ignored during `ffile -R` invocations because the command is invoked and options are specified automatically by `restore`. These options are flagged with an asterisk (*).

`d*` Inhibits recording of the archive in the backup history log.

`l*` Creates a long form of the backup history log that includes a table-of-contents for the archive. This includes the data used to generate a listing of each file in the archive (like that produced by the `ls -l` command).

`m*` Mounts the originating file system read-only before starting the backup and remounts it with its original permissions after completing the backup. Cannot be used with `root` or `/usr` file systems.

`o` Permits the user to override media insertion requests [see `getvol`(1M) and the description of the `-o` option].

`r*` Includes remotely mounted resources in the archive.

`t*` Creates a table of contents for the backup on additional media instead of in the backup history log.

`v*` Validates the archive as it is written. A checksum is computed as the archive is being written; as each medium is completed, it is re-read and the checksum recomputed to verify that each block is readable and correct. If either check fails, the medium is considered unreadable. If `-A` has been specified, the archiving operation fails; otherwise, the operator

is prompted to replace the failed medium.

`A` Establishes automated mode, (i.e., does not prompt the user to insert or remove media).

`E*` Reports an estimate of media usage for the archive; then performs the backup.

`N*` Reports an estimate of media usage for the archive; does not perform the backup.

`S` Displays a period (.) for every 100 (512 byte) blocks read-from or written-to the archive on the destination device.

`V` Displays the name of each file written-to or extracted-from the archive on the destination device.

User Interactions

The connection between an archiving method and `backup` is more complex than a simple fork/exec or pipe. The `backup` command is responsible for all interactions with the user, either directly, or through `bkoper`. Therefore, `ffile` neither reads from standard-input nor writes to standard-output or standard-error. A method library must be used [see `libbrmeth`(3)] to communicate reports (estimates, filenames, periods, status, etc.) to `backup`.

DIAGNOSTICS

The exit codes for `ffile` are the following:

0 successful completion of the task
1 one or more parameters to `ffile` are invalid.
2 an error has occurred which caused `ffile` to fail to complete all portions of its task.

FILES

`/usr/oam/bkrs/tables/bkexcept.tab`
lists the files that are to be excluded from an incremental file system backup.

`/usr/oam/bkrs/tables/bkhist.tab`
lists the labels of all volumes that have been used for backup operations.

`/usr/oam/bkrs/tables/rsstatus.tab`
tracks the status ofall restore requests from users.

`/usr/oam/bkrs/logs/bklog` logs errors generated by the backup methods and the `backup` command

`/usr/oam/bkrs/logs/rslog` logs errors generated by the restore methods and the `restore` command

`$TMP/filelist$$` temporarily stores a table of contents for a backup archive.

SEE ALSO

`backup`(1M), `bkoper`(1M) `cpio`(1), `cpio`(4), `device.tab`(4), `fdp`(1), `ffile`(1), `fimage`(1), `getvol`(1M), `incfile`(1), `labelit`(1M), `libbrmeth`(3), `ls`(1), `restore`(1M), `rsoper`(1M), `time`(2), `urestore`(1)

NAME

fgrep - search a file for a character string

SYNOPSIS

fgrep [-bchilncx] -e *special_str* | -f *str_file* | *string* [*file* . . .]

DESCRIPTION

fgrep (fixed string grep) searches files for a character string and prints all lines that contain that string. fgrep is different from grep and egrep because it searches for a string instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters $, *, [, ^, |, (,), and \ are interpreted literally by fgrep, that is, fgrep does not recognize full regular expressions as does egrep. Because these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes ´ . . . ´.

If no files are specified, fgrep assumes standard input. Normally, each line found is copied to the standard output. The filename is printed before each line found if there is more than one input file.

Command line options are:

-b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
-c Print only a count of the lines that contain the pattern.
-h Suppress printing of filenames when searching multiple files.
-i Ignore uppercase/lowercase distinction during comparisons. This option is valid only for single byte characters.
-l Print the names of files with matching lines once, separated by newlines. Does not repeat the names of files when the pattern is found more than once.
-n Precede each line by its line number in the file (first line is 1).
-v Print all lines except those that contain the pattern.
-x Print only lines matched entirely.
-e *special_str*
Search for a *special_str* (*string* begins with a -).
-f *str_file*
Take the list of *strings* from *str_file*.

INTERNATIONAL FUNCTIONS

fgrep can process characters from supplementary code sets. In regular expressions, searches are performed on characters, not on individual bytes.

SEE ALSO

ed(1), egrep(1), grep(1), sed(1), sh(1

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).

NOTES

Ideally there should be only one grep command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in /usr/include/stdio.h.

NAME

`file` - determine file type

SYNOPSIS

`file` [`-h`] [`-m` *mfile*] [`-f` *ffile*] *arg* ...
`file` [`-h`] [`-m` *mfile*] `-f` *ffile*
`file -c` [`-m` *mfile*]

DESCRIPTION

`file` performs a series of tests on each file supplied by *arg* and, optionally, on each file supplied in *ffile* in an attempt to classify it. If *arg* appears to be a text file, `file` examines the first 512 bytes and tries to guess its programming language. If *arg* is an executable `a.out`, `file` prints the version stamp, provided it is greater than 0. If *arg* is a symbolic link, by default the link is followed and `file` tests the file that the symbolic link references.

`-c` Check the magic file for format errors. For reasons of efficiency, this validation is normally not carried out.

`-f` *ffile* *ffile* contains the names of the files to be examined.

`-h` Do not follow symbolic links.

`-m` *mfile* Use *mfile* as an alternate magic file, instead of `/etc/magic`.

`file` uses `/etc/magic` to identify files that have a magic number. A magic number is a numeric or string constant that indicates the file type. Commentary at the beginning of `/etc/magic` explains its format.

FILES

`/etc/magic`

INTERNATIONAL FUNCTIONS

`file` can classify files containing characters from supplementary code sets. `file` reads each argument and can distinguish data files, program text files, shell scripts and executable files as follows:

Files	*Classification*
Data files containing supplementary characters	data
Shell scripts containing supplementary characters	command text
Language program text files containing literals or comments using supplementary characters	*xxx* text
Executable files	executable

SEE ALSO

`filehdr`(4).

DIAGNOSTICS

If the `-h` option is specified and *arg* is a symbolic link, `file` prints the error message:

`symbolic link to` *arg*

NAME

`fimage` - create, restore an image archive of a filesystem

SYNOPSIS

`fimage -B` [`-dlmotuvAENS`] *bkjobid ofsname ofsdev ofslab descript*

`fimage -RC` [`-dlmotuvAENS`] *ofsname ofsdev refsname redev rsjobid descript*

`fimage -RF` [`-dlmotuvAENS`] *ofsname ofsdev descript rsjobid:uid:date:type:name* [:[*rename*]:[*inode*]] . . .

DESCRIPTION

The `fimage` command is invoked as a child process by other shell commands. The command name, `fimage`, is read either from the `bkhist.tab` file or the `bkreg -m` command and option. The `-B`, `-R`, `-F`, and `-C` options are passed to `fimage` by the shell commands `backup`, `restore`, and `urestore` described below. The other options are passed from the `bkhist.tab` file or the `bkreg -p` command and option. The arguments are sent to `fimage` from various locations in the backup service. `fimage` neither reads from standard-input nor writes to standard-output or standard-error.

`fimage -B` is invoked as a child process by `bkdaemon` to perform an image backup of the filesystem *ofsname* (the originating filesystem). All files in *ofsname* are archived. The resulting backup is created in the format described on `volcopy`(1M). The backup is recorded in the backup history log, `/etc/bkup/bkhist.tab`.

`fimage -RC` and `-RF` are invoked as child processes by the `rsoper` command to extract files from an image archive created by `fimage -B`. The filesystem archive is assumed to be in the format described on `volcopy` format.

If the `-RC` option is selected, the entire filesystem is restored.

If the `-RF` option is specified, only selected objects from the archive are restored. Each 7-tuple, composed of *rsjobid:uid:date:type:name:rename:inode*, specifies an object to be restored from the filesystem archive. The 7-tuple objects come to `fimage` from the `rsstatus.tab` file.

The arguments to `fimage` are defined as follows:

bkjobid — the job id assigned by `backup`. The method uses the *bkjobid* when it creates history log and table-of-contents entries.

ofsname — the name of the file system that is to be backed up.

ofsdev — the name of the block special device on which the file system resides.

ofslab — the volume name on the file system [see `labelit`(1M)].

descript — is a description for a destination device in the form:

dgroup:dname:dchar:dlabels

dgroup specifies a device group [see `devgroup.tab`(4)].
dname specifies a particular device name [see `device.tab`(4)].
dchars specifies characteristics associated with the device. If specified, *dchar* overrides the defaults for the specified device and group. [See `device.tab`(4) for a further description of device characteristics.] *dlabels* specifies the volume names for the media to be used for reading or writing the archive.

refsname if non-null, the name of the file system to be restored to instead of *ofsname*. At least one of *refsname* and *redev* must be null.

redev if non-null, the slice to be restored to instead of *ofsdev*. At least one of *refsname* and *redev* must be null.

rsjobid the restore jobid assigned by `restore` or `urestore`.

uid the real uid of the user who requested the object to be restored. It must match the uid of the owner of the object at the time the archive was made, or it must be the superuser uid.

date the newest "last modification time" that is acceptable for a restorable object. The object is restored from the archive immediately older than this date. *date* is a hexadecimal representation of the date and time provided by the `time` system call [see `time`(2)].

type either `F` or `D`, indicating that the object is a file or a directory, respectively.

name the name the object had in the file system archive.

rename the name that the object should be restored to (it may differ from the name the object had in the file system archive). If omitted, the object is restored to *name.*

inode the inode number of the object as it was stored in the file system archive. *[inode]* is not used by `ffile -R`, and is provided only for command-line compatibility with other restoration methods.

Options

Some options are only significant during `fimage -B` invocations; they are accepted but ignored during `fimage -R` invocations because the command is invoked and options are specified automatically by `restore`. These options are flagged with an asterisk (*).

`d*` Inhibits recording the archive in the backup history log.

`l*` Creates a long form of the backup history log that includes a table-of-contents for the archive. This includes the data used to generate a listing of each file in the archive (like that produced by the `ls -l` command).

`m*` Mounts the originating filesystem read-only before starting the backup and remounts it with its original permissions after completing the backup. Cannot be used with `root` or `/usr` filesystems.

`o` Permits the user to override media insertion requests [see `getvol`(1M) and the description of the `-o` option].

`t*` Creates a table of contents for the backup on additional media instead of in the backup history log.

`u*` Unmounts the originating filesystem before the backup is begun. After the backup is complete, remounts the filesystem under its original permission. This option cannot be used with a `root` or `usr` filesystem. The `-u` option overrides the `-m` option.

v* Validates the archive as it is written. A checksum is computed as the archive is being written; as each medium is completed, it is re-read and the checksum recomputed to verify that each block is readable and correct. If either check fails, the medium is considered unreadable. If -A has been specified, the archiving operation fails; otherwise, the operator is prompted to replace the failed medium.

A Do not prompt the user for removable media operations (automated operation).

E* Reports an estimate of media usage for the archive; then performs the backup.

N* Reports an estimate of media usage for the archive; does not perform the backup.

S Displays a period (.) for every 100 (512 byte) blocks read-from or written-to the archive on the destination device.

User Interactions

The connection between an archiving method and backup is more complex than a simple fork/exec or pipe. The backup command is responsible for all interactions with the user, either directly, or through bkoper. Therefore, ffile neither reads from standard-input nor writes to standard-output or standard-error. A method library must be used [see libbrmeth(3)] to communicate reports (estimates, filenames, periods, status, etc.) to backup.

DIAGNOSTICS

The exit codes for ffile are the following:

0 successful completion of the task
1 one or more parameters to ffile are invalid.
2 an error has occurred which caused ffile to fail to complete all portions of its task.

FILES

/etc/bkup/bkhist.tab	lists the labels of all volumes that have been used for backup operations.
/etc/bkup/rsstatus.tab	tracks the status ofall restore requests from users.
/etc/bkup/bklog	logs errors generated by the backup methods and the backup command
/etc/bkup/rslog	logs errors generated by the restore methods and the restore command
$TMP/filelist$$	temporarily stores a table of contents for a backup archive.

SEE ALSO

backup(1M), bkoper(1M) device.tab(4), fdp(1), ffile(1), fimage(1), getvol(1M), incfile(1), labelit(1M), libbrmeth(3), ls(1), restore(1M), rsoper(1M), time(2), urestore(1), volcopy(1M)

NAME

finc - fast incremental backup

SYNOPSIS

/usr/sbin/finc [*selection-criteria*] *file-system raw-tape*

DESCRIPTION

finc selectively copies the input *file-system* to the output *raw-tape*. The cautious will want to mount the input *file-system* read-only to ensure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by labelit. The selection is controlled by the *selection-criteria*, accepting only those inodes/files for whom the conditions are true.

It is recommended that production of a finc tape be preceded by the ff command, and the output of ff be saved as an index of the tape's contents. Files on a finc tape may be recovered with the frec command.

The argument *n* in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, -*n* means less than *n*, and *n* means exactly *n*. A day is defined as a period of 24 hours.

-a *n*	True if the file has been accessed in *n* days.
-m *n*	True if the file has been modified in *n* days.
-c *n*	True if the i-node has been changed in *n* days.
-n *file*	True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a tape consisting of all files from file-system /usr modified in the last 48 hours:

```
finc  -m  -2  /dev/rdsk/m328_c1d0s2  /dev/rSA/ctape1
```

SEE ALSO

ff(1M), frec(1M), labelit(1M), cpio(1).

NOTE

The raw device is required when providing both the file system and the tape to finc. Failure to do so will cause an error.

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. In the descriptions, the argument *n* is used as a decimal integer where *+n* means more than *n*, *-n* means less than *n* and *n* means exactly *n*. Valid expressions are:

-name *pattern* — True if *pattern* matches the current filename. Normal shell filename generation characters [see sh(1)] may be used. A backslash (\ ) is used as an escape character within the pattern. The pattern should be escaped or quoted when find is invoked from the shell. Characters from supplementary code sets can be used in *pattern*.

-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 (-), only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.

-size *n*[c] — True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a c, the size is in characters.

-atime *n* — True if the file was accessed *n* days ago. The access time of directories in *path-name-list* is changed by find itself.

-mtime *n* — True if the file's data was modified *n* days ago.

-ctime *n* — True if the file's status was changed *n* days ago.

-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. Characters from supplementary code sets can be used in *cmd*.

-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 entering y. Characters from supplementary code sets can be used in *cmd*.

-print — Always true; causes the current path name to be printed.

-cpio *device* — Always true; write the current file on *device* in cpio(1) format.

-newer *file* — True if the current file has been modified more recently than the argument *file*.

-depth — Always true; 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.

`-mount`	Always true; restricts the search to the file system containing the directory specified.
`-local`	True if the file physically resides on the local system.
`(` *expression* `)`	True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).
`-type` *c*	True if the type of the file is *c*, where *c* is `b`, `c`, `d`, `l`, `p`, or `f` for block special file, character special file, directory, symbolic link, fifo (named pipe), or plain file, respectively.
`-follow`	Always true; causes symbolic links to be followed. When following symbolic links, `find` keeps track of the directories visited so that it can detect infinite loops; for example, such a loop would occur if a symbolic link pointed to an ancestor. This expression should not be used with the `-type l` expression.
`-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.
`-nouser`	True if the file belongs to a user not in the `/etc/passwd` file.
`-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.
`-nogroup`	True if the file belongs to a group not in the `/etc/group` file.
`-fstype` *type*	True if the filesystem to which the file belongs is of type *type*.
`-inum` *n*	True if the file has inode number *n*.
`-prune`	Always true; do not examine any directories or files in the directory structure below the *pattern* just matched. See the examples below.

The primaries may be combined using the following operators (in order of decreasing precedence):

1. The negation of a primary (`!` is the unary *not* operator).
2. Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
3. Alternation of primaries (`-o` is the *or* operator).

Note that when you use `find` in conjunction with `cpio`, if you use the `-L` option with `cpio`, you must use the `-follow` expression with `find` and vice versa. Otherwise, there will be undesirable results.

EXAMPLES

Remove all files in your home directory named `a.out` or `*.o` that have not been accessed for a week:

```
find $HOME \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

Recursively print all filenames in the current directory and below, but skipping SCCS directories:

```
find . -name SCCS -prune -o -print
```

Recursively print all filenames in the current directory and below, skipping the contents of SCCS directories, but printing out the SCCS directory name:

```
find . -print -name SCCS -prune
```

FILES

```
/etc/passwd
/etc/group
```

INTERNATIONAL FUNCTIONS

`find` can process characters from supplementary code sets in addition to ASCII characters. Searches are performed on characters, not individual bytes.

Characters from supplementary code sets can be used in *path-name-list*.

SEE ALSO

`chmod`(1), `sh`(1), `test`(1) `stat`(2), `umask`(2), `fs`(4).

NOTE

When using `find` to determine files modified within a range of time, one must use the `?time` argument before the `-print` argument, otherwise `find` will give all files.

The following option is obsolete and will not be supported in future releases.

`-cpio` *device* Always true; write the current file on *device* in `cpio`(1) format (5120-byte records).

NAME

`finger` - display information about local and remote users

SYNOPSIS

`finger` [`-bfhilmpqsw`] *username...*

`finger` [`-l`] *username@hostname...* (`TCP/IP`)

DESCRIPTION

By default, the `finger` command displays information about each , logged-in user, including login name, full name, terminal name (prepended with a '`*`' if write-permission is denied, a space if write-permission is allowed, and a '`?`' if write-permission can't be determined), idle time, login time, and location if known.

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.

When one or more *username* arguments are given, more detailed information is given for each *username* specified, whether they are logged in or not. *username* must be that of a local user, and may be a first or last name, or an account name. When `finger` is used to find users on a remote device, the user and the name of the remote device are specified in the form *username@hostname*. Information is presented in a multi-line format, and includes, in addition to the information mentioned above:

the user's home directory and login shell

time the user logged in if currently logged in, or the time the user last logged in if not, as well as the terminal or host from which the user logged in and, if a terminal.

last time the user received mail, and the last time the user read their mail

any plan contained in the file `.plan` in the user's home directory

and any project on which the user is working described in the file `.project` (also in the user's home directory)

The following options are available:

`-b` Suppress printing the user's home directory and shell in a long format printout.

`-f` Suppress printing the header that is normally printed in a non-long format printout.

`-h` Suppress printing of the `.project` file in a long format printout.

`-i` Force "idle" output format, which is similar to short format except that only the login name, terminal, login time, and idle time are printed.

`-l` Force long output format.

`-m` Match arguments only on user name (not first or last name).

`-p` Suppress printing of the `.plan` file in a long format printout.

`-q` Force quick output format, which is similar to short format except that only the login name, terminal, and login time are printed.

`-s` Force short output format.

`-w` Suppress printing the full name in a short format printout.

Within the `TCP/IP` network, the `-l` option can be used remotely.

FILES

`/var/adm/utmp`	who is logged in
`/etc/passwd`	for users' names
`/var/adm/lastlog`	last login times
`~/.plan`	plans
`~/.project`	projects

SEE ALSO

`passwd`(1), `who`(1), `whois`(1)

NOTES

Only the first line of the `~/.project` file is printed.

NAME

`fingerd`, `in.fingerd` - remote user information server

SYNOPSIS

`in.fingerd`

DESCRIPTION

`fingerd` implements the server side of the Name/Finger protocol, specified in RFC 742. The Name/Finger protocol provides a remote interface to programs which display information on system status and individual users. The protocol imposes little structure on the format of the exchange between client and server. The client provides a single command line to the finger server which returns a printable reply.

`fingerd` waits for connections on TCP port 79. Once connected it reads a single command line terminated by a <RETURN-LINE-FEED> which is passed to `finger`(1). `fingerd` closes its connections as soon as the output is finished.

If the line is null (only a RETURN-LINEFEED is sent) then `finger` returns a default report that lists all users logged into the system at that moment.

If a user name is specified (for instance, `eric`<RETURN-LINE-FEED>) then the response lists more extended information for only that particular user, whether logged in or not. Allowable names in the command line include both login names and user names. If a name is ambiguous, all possible derivations are returned.

FILES

`/var/utmp`	who is logged in
`/etc/passwd`	for users' names
`/var/adm/lastlog`	last login times
`$HOME/.plan`	plans
`$HOME/.project`	projects

SEE ALSO

`finger`(1)

Harrenstien, Ken, *NAME/FINGER*, RFC 742, Network Information Center, SRI International, Menlo Park, Calif., December 1977

NOTES

Connecting directly to the server from a TIP or an equally narrow-minded TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation. `fingerd` should be taught to filter out IAC's and perhaps even respond negatively (IAC *will not)* to all option commands received.

NAME

fmlcut - cut out selected fields of each line of a file

SYNOPSIS

fmlcut -c*list* [*file* . . .]
fmlcut -f*list* [-d*char*] [-s] [*file* . . .]

DESCRIPTION

The fmlcut function cuts out columns from a table or fields from each line in *file*; in database parlance, it implements the projection of a relation. fmlcut can be used as a filter; if *file* is not specified or is -, the standard input is read. *list* specifies the fields to be selected. Fields can be fixed length (character positions) or variable length (separated by a field delimiter character), depending on whether -c or -f is specified.

Note that either the -c or the -f option must be specified.

The meanings of the options are:

list A comma-separated list of integer field numbers (in increasing order), with optional - to indicate ranges For example: 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field).

-c*list* If -c is specified, *list* specifies character positions (for example, -c1-72 would pass the first 72 characters of each line). Note that no space intervenes between -c and *list*.

-f*list* If -f is specified, *list* is a list of fields assumed to be separated in the file by the default delimiter character, TAB, or by *char* if the -d option is specified. For example, -f1,7 copies the first and seventh field only. Lines with no delimiter characters are passed through intact (useful for table subheadings), unless -s is specified. Note that no space intervenes between -f and *list*. The following options can be used if you have specified -f.

- -d*char* If -d is specified, *char* is the field delimiter. Space or other characters with special meaning to FMLI must be quoted. Note that no space intervenes between -d and *char*. The default field delimiter is TAB.
- -s Suppresses lines with no delimiter characters. If -s is not specified, lines with no delimiters will be passed through untouched.

EXAMPLES

`fmlcut -d: -f1,5 /etc/passwd`	gets login IDs and names
`` `who am i | fmlcut -f1 -d" "` ``	gets the current login name

DIAGNOSTICS

fmlcut returns the following exit values:

0 when the selected field is successfully cut out
2 on syntax errors

The following error messages may be displayed on the FMLI message line:

`ERROR: line too long`
: A line has more than 1023 characters or fields, or there is no newline character.

`ERROR: 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.

`ERROR: no fields`
: The *list* is empty.

`ERROR: no delimiter`
: Missing *char* on `-d` option.

NOTES

`fmlcut` cannot correctly process lines longer than 1023 characters, or lines with no newline character.

SEE ALSO

`fmlgrep`(1F)

NAME

fmlexpr - evaluate arguments as an expression

SYNOPSIS

fmlexpr *arguments*

DESCRIPTION

The fmlexpr function evaluates its arguments 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 FMLI 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, 2s 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.

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. Regular expression syntax is the same as that of ed(1), except that all patterns are "anchored" (that is, begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of bytes matched (0 on failure). Alternatively, the \(. . . \) pattern symbols can be used to return a portion of the first argument.

EXAMPLES

1. Add 1 to the variable a:

```
'fmlexpr $a + 1 | set -l a'
```

2. For $a equal to either "/usr/abc/*file*" or just "*file*":

```
fmlexpr  $a  :  .*/\(.*\)  \|  $a
```

returns the last segment of a path name (for example, *file*). Watch out for / alone as

an argument: `fmlexpr` will take it as the division operator (see NOTES below).

3. A better representation of example 2.

```
fmlexpr  //$a  :  .*/\(.*\)
```

The addition of the `//` characters eliminates any ambiguity about the division operator (because it makes it impossible for the left-hand expression to be interpreted as the division operator), and simplifies the whole expression.

4. Return the number of characters in `$VAR`.

```
fmlexpr $VAR : .*
```

DIAGNOSTICS

As a side effect of expression evaluation, `fmlexpr` returns the following exit values:

`0`	if the expression is neither null nor `0` (that is, TRUE)
`1`	if the expression is null or `0` (that is, FALSE)
`2`	for invalid expressions (that is, FALSE).

syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string

In the case of syntax errors and non-numeric arguments, an error message will be printed at the current cursor position. Use `refresh` to redraw the screen.

NOTES

After argument processing by FMLI, `fmlexpr` cannot tell the difference between an operator and an operand except by the value. If `$a` is an `=`, the command:

```
fmlexpr  $a  =  =
```

looks like:

```
fmlexpr  =  =  =
```

as the arguments are passed to `fmlexpr` (and they will all be taken as the `=` operator). The following works, and returns TRUE:

```
fmlexpr  X$a  =  X=
```

SEE ALSO

`ed`(1), `expr`(1), `set`(1F), `sh`(1)

NAME

fmlgrep - search a file for a pattern

SYNOPSIS

fmlgrep [*options*] *limited_regular_expression* [*file . . .*]

DESCRIPTION

fmlgrep searches *file* for a pattern and prints all lines that contain that pattern. The fmlgrep function uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with ed(1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful when using FMLI special characters (for example, $, `, ´, ") in *limited_regular_expression*. It is safest to enclose the entire *limited_regular_expression* in single quotes ´ . . . ´ .

If *file* is not specified, fmlgrep assumes standard input. Normally, each line matched is copied to standard output. The file name is printed before each line matched if there is more than one input file.

Command line options are:

- -b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
- -c Print only a count of the lines that contain the pattern.
- -i Ignore upper/lower case distinction during comparisons.
- -l Print only the names of files with matching lines, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
- -n Precede each line by its line number in the file (first line is 1).
- -s Suppress error messages about nonexistent or unreadable files.
- -v Print all lines except those that contain the pattern.

DIAGNOSTICS

fmlgrep returns the following exit values:

0 if the pattern is found (that is, TRUE)
1 if the pattern is not found (that is, FALSE)
2 if an invalid expression was used or *file* is inaccessible

NOTES

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in /usr/include/stdio.h.

If there is a line with embedded nulls, fmlgrep will only match up to the first null; if it matches, it will print the entire line.

SEE ALSO

fmlcut(1F)
ed(1), egrep(1), fgrep(1), grep(1).

NAME

fmli - invoke FMLI

SYNOPSIS

fmli [-a *alias_file*] [-c *command_file*] [-i *initialization_file*] *file* ...

DESCRIPTION

The fmli command invokes the Form and Menu Language Interpreter and opens the frame(s) specified by the *file* argument. The *file* argument is the pathname of the initial frame definition file(s), and must follow the naming convention Menu.*xxx*, Form.*xxx* or Text.*xxx* for a menu, form or text frame respectively, where *xxx* is any string that conforms to UNIX system file naming conventions. The FMLI descriptor lifetime will be ignored for all frames opened by argument to fmli. These frames have a lifetime of immortal by default.

The available options are as follows:

-a If -a is specified, *alias_file* is the name of a file which contains lines of the form *alias=pathname*. Thereafter, $*alias* can be used in definition files to simplify references to objects or devices with lengthy pathnames, or to define a search path (similar to $PATH in the UNIX system shell).

-c If -c is specified, *command_file* is the name of a file in which default FMLI commands can be disabled, and new application-specific commands can be defined. The contents of *command_file* are reflected in the FMLI Command Menu.

-i If -i is specified, *initialization_file* is the name of a file in which the following characteristics of the application as a whole can be specified:

- A transient introductory frame displaying product information
- A banner, its position, and other elements of the banner line
- Color attributes for all elements of the screen
- Screen Labeled Keys (SLKs) and their layout on the screen.

Environment Variables

LOADPFK

Leaving this environment variable unset, on certain terminals, tells FMLI to download its equivalent character sequences for using function keys into the terminal's programmable function keys, wiping out any settings the user may already have set in the function keys. Setting LOADPFK=NO in the environment will prevent this downloading.

COLUMNS

Can be used to override the width of the logical screen defined for the terminal set in TERM. For terminals with a 132-column mode, for example, invoking FMLI with the line

COLUMNS=132 fmli *frame-file*

will allow this wider screen width to be used.

LINES

Can be used to override the length of the logical screen defined for the terminal set in TERM.

EXAMPLES

To invoke `fmli`:

```
fmli Menu.start
```

where `Menu.start` is an example of *file* named according to the file name conventions for menu definition files explained above.

To invoke `fmli` and name an initialization file:

```
fmli -i init.myapp Menu.start
```

where `init.myapp` is an example of *initialization_file*.

DIAGNOSTICS

If *file* is not supplied to the `fmli` command, `fmli` returns the message:

```
Initial object must be specified.
```

If *file* does not exist or is not readable, `fmli` returns an error message and exits. The example command line above returns the following message and exits:

```
Can't open object "Menu.start"
```

If *file* exists, but does not start with one of the three correct object names (`Menu.`, `Form.`, or `Text.`) or if it is named correctly but does not contain the proper data, `fmli` starts to build the screen by putting out the screen labels for function keys, after which it flashes the message:

```
I do not recognize that kind of object
```

and then exits.

FILES

`/usr/bin/fmli`

SEE ALSO

`vsig`(1F).

NAME

fmt - simple text formatters

SYNOPSIS

fmt [-cs] [-w *width*] [*file* . . .]

DESCRIPTION

fmt is a simple text formatter that fills and joins lines to produce output lines of (up to) the number of characters specified in the -w *width* option. The default *width* is 72. fmt concatenates the *inputfile*s listed as arguments. If none are given, fmt formats text from the standard input.

Blank lines are preserved in the output, as is the spacing between words. fmt does not fill lines beginning with a ".'' (dot), for compatibility with nroff(1). Nor does it fill lines starting with "From:".

Indentation is preserved in the output, and input lines with differing indentation are not joined (unless -c is used).

fmt can also be used as an in-line text filter for vi(1); the vi command:

```
!}fmt
```

reformats the text between the cursor location and the end of the paragraph.

OPTIONS

-c Crown margin mode. Preserve the indentation of the first two lines within a paragraph, and align the left margin of each subsequent line with that of the second line. This is useful for tagged paragraphs.

-s Split lines only. Do not join short lines to form longer ones. This prevents sample lines of code, and other such formatted text, from being unduly combined.

-w *width* Fill output lines to up to *width* columns.

SEE ALSO

nroff(1), vi(1)

NOTES

The -w *width* option is acceptable for BSD compatibility, but it may go away in future releases.

NAME

fmthard - populate VTOC on hard disks

SYNOPSIS

/usr/sbin/fmthard [-d *data*] [-i] [-s *datafile*]
[-n *volume_name*] /dev/rdsk/*prefix*_c*X*d*Y*s7

DESCRIPTION

The fmthard command creates (or updates) the VTOC (Volume Table of Contents) on hard disks; use the dinit command to actually format a hard disk. The /dev/rdsk/*prefix*_c*X*d*Y*s7 file must be slice seven of the character special file of the device where the new VTOC is to be installed. *prefix* is the device type, c*X* is the controller number of that device, d*Y* is the device number for the controller, and s*Z* is the slice number. See intro(7) for complete lists of controllers, devices, and slices.

OPTIONS

The following options apply to fmthard:

-d *data*
: The *data* argument of this option is a string representing the information for a particular slice in the current VTOC. The string must be of the format *part:tag:flag:start:size* where *part* is the slice number, *tag* is the ID tag of the slice, *flag* is the set of permission flags, *start* is the starting sector number of the slice, and *size* is the number of sectors in the slice. See the description of the *datafile* below for more information on these fields.

-i
: Lets the command create the desired VTOC table, but prints the information to standard output instead of modifying the VTOC on the disk.

-n *volume_name*
: Allows the disk to be given a *volume_name* up to 8 characters long.

-s *datafile*
: The VTOC is populated according to a *datafile* created by the user. The *datafile* format is described below. This option causes all of the disk slice timestamp fields to be set to zero.

Every VTOC generated by fmthard will also have slice 7 (the whole disk). Slice 7 is the only slice that can overlap others.

The *datafile* contains one specification line for each slice, starting with slice 0. Each line is delimited by a new-line character (\n). If the first character of a line is an asterisk (*), the line is treated as a comment. Each line is composed of entries that are position-dependent, separated by white space and having the following format:

slice tag flag starting_sector size_in_sectors

where the entries have the following values:

slice
: The slice number: 0-15 decimal or 0x0-0xf hexadecimal.

tag
: The slice tag: a two-digit hex number. The following are reserved codes: 0x01 (V_BOOT), 0x02 (V_ROOT), 0x03 (V_SWAP), 0x04 (V_USR), 0x05 (V_VAR), 0x06 (V_STAND), 0x07 (V_BACKUP) and 0x08(V_HOME).

flag The flag allows a slice to be flagged as unmountable or read only, the masks being: V_UNMNT 0x01, and V_RONLY 0x10. For mountable slices use 0x00.

starting sector The sector number (decimal) on which the slice starts.

size in sectors The number (decimal) of sectors occupied by the slice.

Note that you can save the output of a `prtvtoc` command to a file, edit the file, and use it as the *datafile* argument to the `-s` option.

SEE ALSO

`dinit`(1M), `prtvtoc`(1M)

NOTES

Special care should be exercised when overwriting an existing VTOC, as incorrect entries could result in current data being inaccessible. As a precaution, save the old VTOC.

A drive is unbootable until the `dinit` command is executed with the `-b` option to place a bootstrap on the disk.

NAME

fmtmsg - display a message on stderr or system console

SYNOPSIS

fmtmsg [-c *class*] [-u *subclass*] [-l *label*] [-s *severity*] [-t *tag*] [-a *action*] *text*

DESCRIPTION

Based on a message's classification component, fmtmsg either writes a formatted message to stderr or writes a formatted message to the console.

A formatted message consists of up to five standard components as defined below. The classification and subclass components are not displayed as part of the standard message, but rather define the source of the message and direct the display of the formatted message. The valid options are:

-c *class* Describes the source of the message. Valid keywords are:

- hard — The source of the condition is hardware.
- soft — The source of the condition is software.
- firm — The source of the condition is firmware.

-u *subclass* A list of keywords (separated by commas) that further defines the message and directs the display of the message. Valid keywords are:

- appl — The condition originated in an application. This keyword should not be used in combination with either util or opsys.
- util — The condition originated in a utility. This keyword should not be used in combination with either appl or opsys.
- opsys — The message originated in the kernel. This keyword should not be used in combination with either appl or util.
- recov — The application will recover from the condition. This keyword should not be used in combination with nrecov.
- nrecov — The application will not recover from the condition. This keyword should not be used in combination with recov.
- print — Print the message to the standard error stream stderr.
- console — Write the message to the system console. print, console, or both may be used.

-l *label* Identifies the source of the message.

-s *severity* Indicates the seriousness of the error. The keywords and definitions of the standard levels of *severity* are:

- halt — The application has encountered a severe fault and is halting.
- error — The application has detected a fault.
- warn — The application has detected a condition that is out of the ordinary and might be a problem.

`info` The application is providing information about a condition that is not in error.

`-t` *tag* The string containing an identifier for the message.

`-a` *action* A text string describing the first step in the error recovery process. This string must be written so that the entire *action* argument is interpreted as a single argument. `fmtmsg` precedes each action string with the `TO FIX:` prefix.

text A text string describing the condition. Must be written so that the entire *text* argument is interpreted as a single argument.

The environment variables `MSGVERB` and `SEV_LEVEL` control the behavior of `fmtmsg`. `MSGVERB` is set by the administrator in the `/etc/profile` for the system. Users can override the value of `MSGVERB` set by the system by resetting `MSGVERB` in their own `.profile` files or by changing the value in their current shell session. `SEV_LEVEL` can be used in shell scripts.

`MSGVERB` tells `fmtmsg` which message components to select when writing messages to `stderr`. The value of `MSGVERB` is a colon separated list of optional keywords. `MSGVERB` can be set as follows:

```
MSGVERB=[keyword[:keyword[:...]]]
export MSGVERB
```

Valid *keywords* are: `label`, `severity`, `text`, `action`, and `tag`. If `MSGVERB` contains a keyword for a component and the component's value is not the component's null value, `fmtmsg` includes that component in the message when writing the message to `stderr`. If `MSGVERB` does not include a keyword for a message component, that component is not included in the display of the message. The keywords may appear in any order. If `MSGVERB` is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, `fmtmsg` selects all components.

`MSGVERB` affects only which message components are selected for display. All message components are included in console messages.

`SEV_LEVEL` defines severity levels and associates print strings with them for use by `fmtmsg`. The standard severity levels shown below cannot be modified. Additional severity levels can be defined, redefined, and removed.

0 (no severity is used)
1 HALT
2 ERROR
3 WARNING
4 INFO

SEV_LEVEL is set as follows:

```
SEV_LEVEL=[description[:description[:...]]]
export SEV_LEVEL
```

description is a comma-separated list containing three fields:

description=severity_keyword, level, printstring

severity_keyword is a character string used as the keyword with the `-s` *severity* option to `fmtmsg`.

level is a character string that evaluates to a positive integer (other than `0`, `1`, `2`, `3`, or `4`, which are reserved for the standard severity levels). If the keyword *severity_keyword* is used, *level* is the severity value passed on to `fmtmsg`(3C).

printstring is the character string used by `fmtmsg` in the standard message format whenever the severity value *level* is used.

If `SEV_LEVEL` is not defined, or if its value is null, no severity levels other than the defaults are available. If a *description* in the colon separated list is not a comma separated list containing three fields, or if the second field of a comma separated list does not evaluate to a positive integer, that *description* in the colon separated list is ignored.

DIAGNOSTICS

The exit codes for `fmtmsg` are the following:

`0`	All the requested functions were executed successfully.
`1`	The command contains a syntax error, an invalid option, or an invalid argument to an option.
`2`	The function executed with partial success, however the message was not displayed on `stderr`.
`4`	The function executed with partial success, however the message was not displayed on the system console.
`32`	No requested functions were executed successfully.

EXAMPLES

Example 1: The following example of `fmtmsg` produces a complete message in the standard message format and displays it to the standard error stream:

```
fmtmsg -c soft -u recov,print,appl -l UX:cat -s error -t
UX:cat:001 -a "refer to manual" "invalid syntax"
```

produces:

```
UX:cat: ERROR: invalid syntax
TO FIX: refer to manual   UX:cat:001
```

Example 2: When the environment variable `MSGVERB` is set as follows:

```
MSGVERB=severity:text:action
```

and Example 1 is used, `fmtmsg` produces:

```
ERROR: invalid syntax
TO FIX: refer to manual
```

Example 3: When the environment variable `SEV_LEVEL` is set as follows:

```
SEV_LEVEL=note,5,NOTE
```

the following `fmtmsg` command:

```
fmtmsg -c soft -u print -l UX:cat -s note -a "refer to
manual" "invalid syntax"
```

produces:

```
UX:cat: NOTE: invalid syntax
TO FIX: refer to manual
```

and displays the message on `stderr`.

SEE ALSO

`addseverity`(3C), `fmtmsg`(3C)

FUTURE DIRECTIONS

A slightly different standard error message format and a new developer interface, `pfmt`, is being introduced as the replacement for `fmtmsg`. A similar interface, `lfmt`, is also being introduced for producing a standard format message and forwarding messages to the console and/or to the system message logging and monitoring facilities. `fmtmsg` will be removed at a future time.

NAME

`fold` - fold long lines

SYNOPSIS

`fold` [`-w` *width* | `-`*width*] [*filename* . . .]

DESCRIPTION

Fold the contents of the specified *filename*s, 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.

SEE ALSO

`pr`(1)

NOTES

Folding may not work correctly if underlining is present.

The `-w` *width* option is provided as a transition tool only. It will be removed in future releases.

NAME

`frec` - recover files from a backup tape

SYNOPSIS

`/usr/sbin/frec` [`-p` *path*] [`-f` *reqfile*] *raw_tape i_number*`:`*name* . . .

DESCRIPTION

`frec` recovers files from the specified *raw_tape* backup tape written by `volcopy` or `finc`, given their *i_numbers*. The data for each recovery request will be written into the file given by *name* .

The `-p` option allows you to specify a default prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified (i.e. that do not begin with `/` or `./`). If any directories are missing in the paths of recovery *names*, they will be created.

`-p` *path* — Specifies a prefixing *path* to be used to fully qualify any names that do not start with `/` or `./`.

`-f` *reqfile* — Specifies a file which contains recovery requests. The format is *i_number:newname*, one per line.

EXAMPLES

To recover a file, *i_number* `1216` when backed-up, into a file named `junk` in your current working directory:

```
frec  /dev/rmt/ctape1  1216:junk
```

To recover files with *i_numbers* `14156`, `1232`, and `3141` into files `/usr/src/cmd/a`, `/usr/src/cmd/b` and `/usr/joe/a.c`:

```
frec -p /usr/src/cmd /dev/rmt/ctape1 14156:a 1232:b \
        3141:/usr/joe/a.c
```

SEE ALSO

`cpio`(1), `ff`(1M), `finc`(1M), `labelit`(1M), `volcopy`(1M)

NOTES

While paving a path (i.e., creating the intermediate directories contained in a pathname) `frec` can only recover inode fields for those directories contained on the tape and requested for recovery.

NAME

`fromsmtp` - receive RFC822 mail from SMTP

SYNOPSIS

`fromsmtp` [`-d`] [`-h` *host*] [`-s` *sender*] *to* . . .

DESCRIPTION

`fromsmtp` reads an RFC822 message from its standard input, does some conversion of the message to make it acceptable to UNIX System mail, and pipes the result to `rmail`. The *to* arguments are passed as arguments to `rmail`. `fromsmtp` is normally invoked by `smtpd` to deliver incoming mail messages.

The `-d` option may be used for debugging `fromsmtp`. It will cause the command line for `rmail` to be echoed to standard output, as well as the results of the message (after conversion). The message will not be given to `rmail` when this option is used.

The `-h` *host* option may be used to prepend a host or network name to the front of the sender path in the `From` line at the beginning of the message. This is useful if you need to identify which of several possible networks a message was received from (for possible use in replying).

The `-s` *sender* option is used to give a default sender name, in case `fromsmtp` cannot determine the name of the sender from the message it reads. If this option is not used, the default sender name `unknown` will be used.

FILES

`/usr/bin/rmail` where converted mail is piped to

SEE ALSO

`rmail`(1M), `smtpd`(1M)
RFC822 - Standard for the Format of ARPA Internet Text Messages

NAME

`fsba` - file system block analyzer

SYNOPSIS

`/usr/sbin/fsba` [`-b` *target_block_size*] *file-system1* [*file-system2* . . .]

DESCRIPTION

The `fsba` command determines the disk space required to store the data from an existing file system in a new file system with the specified logical block size. Each *file-system* listed on the command line refers to an existing file system and should be specified by device name (for example, `/dev/rdsk/*`, where the value of `*` is machine dependent).

The *target_block_size* specifies the logical block size in bytes of the new file system. Valid target block sizes are 512, 1024, and 2048. Default target block size is 1024. A block size of 2048 is supported only if the 2K file system package is installed.

The `fsba` command prints information about how many 512-byte disk sectors are allocated to store the data in the old (existing) file system and how many would be required to store the same data in a new file system with the specified logical block size. It also prints the number of allocated and free i-nodes for the existing file system.

If the number of free sectors listed for the new file system is negative, the data will not fit in the new file system unless the new file system is larger than the existing file system. The new file system must be made at least as large as the number of sectors listed by `fsba` as allocated for the new file system. The maximum size of the new file system is limited by the size of the disk slice used for the new file system.

Note that it is possible to specify a *target_block_size* that is smaller than the logical block size of the existing file system. In this case the new file system would require fewer sectors to store the data.

SEE ALSO

`mkfs`(1M), `prtvtoc`(1M)

NAME

fsck (generic) - check and repair file systems

SYNOPSIS

fsck [-F *FSType*] [-V] [-m] [*special* . . .]
fsck [-F *FSType*] [-V] [*current_options*] [-o *specific_options*] [*special* . . .]

DESCRIPTION

fsck audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that some corrective actions will result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond yes or no. If the user does not have write permission fsck defaults to a no action.

The file system should be unmounted when fsck is used. If this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards if the file system is a critical one, for example root.

current_options are options supported by the s5-specific module of fsck. Other *FSTypes* do not necessarily support these options. *specific_options* indicate suboptions specified in a comma-separated list of suboptions and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command. See fsck_*FSType*(1M) for details.

special represents a block or character special device (e.g., /dev/rdsk/*, where the value of * is machine dependent). It is preferable that a character special device be used. fsck will not work on a block device if it is mounted. If *special* is not supplied, fsck looks through /etc/vfstab and executes fsck for all character specials in the fsckdev field of /etc/vfstab for which there is a numeric entry in the fsckpass field.

The options are:

-F Specify the *FSType* on which to operate. The *FSType* should either be specified here or be determinable from /etc/vfstab by matching the *special* with an entry in the table.

-V Echo the complete command line, but do not execute the command. The command line is generated by using the options and arguments provided by the user and adding to them information derived from /etc/vfstab. This option should be used to verify and validate the command line.

-m Check but don't repair. This option checks that the file system is suitable for mounting.

-o Specify *FSType*-specific options.

NOTE

This command may not be supported for all *FSTypes*.

FILES

`/etc/vfstab` list of default parameters for each file system

SEE ALSO

`checkfsys`(1M), `fsck_bfs`(1M), `fsck_s5`(1M), `fsck_ufs`(1M), `mkfs`(1M), `vfstab`(4).

NAME

`fsck` (bfs) - check and repair `bfs` file systems

SYNOPSIS

```
fsck [-F bfs] [generic_options] [special . . .]
fsck [-F bfs] [generic_optionsi] [-y | -n] [special . . .]
```

DESCRIPTION

generic_options are options supported by the generic `fsck` command.

`fsck` checks to see if compaction was in process but was not completed, perhaps as a result of a system crash. If it was, `fsck` completes the compaction of the file [see `fs_bfs`(4)].

The options are:

`-y` Assume a yes response to all questions asked by `fsck`.

`-n` Assume a no response to all questions asked by `fsck`.

SEE ALSO

`checkfsys`(1M), generic `fsck`(1M), `mkfs`(1M), `fs_bfs`(4).

NAME

`fsck` (s5) - check and repair `s5` file systems

SYNOPSIS

`fsck` [`-F s5`] [*generic_options*] [*special...*]
`fsck` [`-F s5`] [*generic_options*] [`-y`] [`-n`] [`-p`] [`-s`*X*] [`-S`*X*] [`-t`*file*] [`-l`] [`-q`] [`-D`] [`-f`] [*special...*]

DESCRIPTION

generic_options are options supported by the generic `fsck` command.

The options are:

`-F s5` Specifies the `s5`-FSType.

`-y` Assume a `yes` response to all questions asked by `fsck`.

`-n` Assume a `no` response to all questions asked by `fsck`; do not open the file system for writing.

`-p` Correct inconsistencies that can be fixed automatically, that is, inconsistencies that are deemed harmless and can be fixed without confirmation by the administrator. Examples of such inconsistencies are unreferenced i-nodes, incorrect counts in the superblocks, and missing blocks in the free list.

`-s`*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The `-s`*X* suboption allows for creating an optimal free-list organization.

If *X* is not given, the values used when the file system was created are used. The format of *X* is *cylinder size:gap size.*

`-S`*X* Conditionally reconstruct the free list. This suboption is like `-s`*X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using `S` will force a `no` response to all questions asked by `fsck`. This suboption is useful for forcing free list reorganization on uncontaminated file systems.

`-t`*file* If `fsck` cannot obtain enough memory to keep its tables, it uses a scratch file. If the `t` option is specified, the *file* named is used as the scratch file, if needed. Without the `t` option, `fsck` will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when `fsck` completes.

`-l` Identify damaged files by their logical names.

`-q` Quiet `fsck`. Unreferenced `fifos` will silently be removed. If `fsck` requires it, counts in the superblock will be automatically fixed and the free list salvaged.

`-D` Directories are checked for bad blocks. Useful after system crashes.

`-f` Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   I-node number out of range.
8. Super Block checks:
   More than 65536 i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the `lost+found` directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. Empty files or directories are removed, as long as the `n` suboption is not specified. `fsck` will force the reconnection of nonempty directories. The name assigned is the i-node number.

NOTE

Checking the raw device is almost always faster.

I-node numbers for `.` and `..` in each directory are not checked for validity.

SEE ALSO

`checkfsys`(1M), `crash`(1M), generic `fsck`(1M), `mkfs`(1M), `ncheck`(1M), `fs`(4)

NAME

fsck (ufs) - file system consistency check and interactive repair

SYNOPSIS

fsck [-F ufs] [*generic_options*] [*special* . . .]
fsck [-F ufs] [*generic_options*] [(-y|-Y)|(-n|-N)] [-o p,b=#,w] [*special* . . .]

DESCRIPTION

generic_options are options supported by the generic fsck command. *current_options* are options supported by the s5-specific module of the fsck command.

fsck audits and interactively repairs inconsistent conditions on file systems. In this case, it asks for confirmation before attempting any corrections. Inconsistencies other than those mentioned above can often result in some loss of data. The amount and severity of data lost can be determined from the diagnostic output.

fsck corrects innocuous inconsistencies such as: unreferenced inodes, too-large link counts in inodes, missing blocks in the free list, blocks appearing in the free list and also in files, or incorrect counts in the super block, automatically. It displays a message for each inconsistency corrected that identifies the nature of, and file system on which, the correction is to take place. After successfully correcting a file system, fsck prints the number of files on that file system, the number of used and free blocks, and the percentage of fragmentation.

The default action for each correction is to wait for the operator to respond either yes or no. If the operator does not have write permission on the file system, fsck will default to a -n (no corrections) action.

Inconsistencies checked are as follows:

Blocks claimed by more than one inode or the free list.

Blocks claimed by an inode or the free list outside the range of the file system.

Incorrect link counts.

Incorrect directory sizes.

Bad inode format.

Blocks not accounted for anywhere.

Directory checks, file pointing to unallocated inode, inode number out of range, absence of '.' and '..' as the first two entries in each directory.

Super Block checks: more blocks for inodes than there are in the file system.

Bad free block list format.

Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the lost+found directory. The name assigned is the inode number. If the lost+found directory does not exist, it is created. If there is insufficient space its size is increased.

A file system may be specified by giving the name of the block or character special device on which it resides, or by giving the name of its mount point.

The options are:

`-F ufs` Specifies the `ufs`-FSType.

`-y | -Y` Assume a yes response to all questions asked by `fsck`.

`-n | -N` Assume a no response to all questions asked by `fsck`; do not open the file system for writing.

`-o` Specify `ufs` file system specific suboptions. These suboptions can be any combination of the following:

- `p` Check the filesystem non-interactively. Exit if there is a problem requiring intervention.
- `b=#` Use the block specified as the super block for the file system. Block 32 is always an alternate super block.

NOTES

Checking the character special device is almost always faster.

SEE ALSO

`checkfsys`(1M), `crash`(1M), generic `fsck`(1M), `mkfs`(1M), `ufs`(4)

NAME

fscoll - Filesystem image generator - Pass 1

SYNOPSIS

fscoll *[-Vv] fileprefix*

SUMMARY

fscoll is designed to be used as a first pass on **.cnf** files when preparing filesystem images with *fsgen*. The images built using *fscoll* and *fsgen*. are intended to be used as master images for CD-ROM production, but other uses, such as UNIX filesystem re-organization, are possible. The usual sequence of operations in preparing a CD-ROM image is to first prepare a **.cnf** file holding configuration information for the CD-ROM image. The *fscoll* utility is then run on the **.cnf** file to produce a complete file and directory list which is automatically saved in a **.lst** file. The **.lst** file is used by the *fsgen* utility to create filesystem images, usually to a tape device.

DESCRIPTION

fscoll requires one file name prefix as an argument. It attempts to open a configuration file named **prefix.cnf**, which it will use as input. If the file does not exist as specified, the program aborts with an error message.

fscoll accepts just two command-line options. The **-v** (verbose) option causes the generation of extra progress information to the **stderr** file, and the **-V** option simply causes *fscoll* to print a version number and exit. The latter option is intended as a means of checking on which version of *fscoll* is running.

Two operations are performed on the input configuration file. First, all configuration statements are checked for acceptable content. Any missing or invalid statements cause *fscoll* to abort with a suitable error message. Secondly, for each input slice, *fscoll* uses the **Root** configuration statement, which must be present at least once for all slice types, to generate a complete list of directories and files for inclusion on the final image.

CONFIGURATION FILE FORMAT

The user supplied configuration file is expected to have a hierarchical structure based on the desired slicing of the final CD-ROM image. Up to 15 slice statements are permitted. Each creates a separate slice on the final image, most of which will map directly to a disk slice as defined by Motorola's disk slicing scheme.

Between each **Slice** statement and the next are the various configuration statements that apply to that slice. The type and number of the configuration statements depend on the slice type code specified on the **Slice** statement. Some configuration entries are required, some are optional. In general, where it is possible to supply a reasonable default value for a configuration item, that statement is optional and the default value is assumed.

Usually the configuration statements that apply to a slice are active for the entire slice. In the case of the **ISOFS** slice, multiple volumes are permitted within the slice definition, and each Volume has its own set of configuration statements.

At least one occurrence of the **Root** configuration entry is required for all slices that generate filesystems (**ISOFS** & **UNIXS5**). The **Root** statement indicates a base directory to be used when collecting the file and directory list for the final filesystem. All files and directories found in and below this ´root´ directory are copied into the final image. Although not required by the program, the **Root** statement is, by convention, the final configuration entry in each slice or ISO9660 volume.

The usual format of the **.cnf** file may be represented as:

```
Slice Statement
  Configuration statements for Slice 0
Slice Statement
  Configuration statements for Slice 1
      .
      .
Slice Statement
  Configuration statements for Slice n
```

However, the format for ISOFS slices is more accurately represented by:

```
Slice Statement
  Volume Statement
    Configuration statements for Volume 0
  Volume Statement
    Configuration statements for Volume 1
        .
        .
  Volume Statement
    Configuration statements for Volume n
```

Comments are permitted in the configuration file and are introduced with '**#**' signs. Keywords are *not* case sensitive and may be typed in any combination of upper- and lowercase. Whitespace is considered a field delimited for all strings unless they are quoted.

For example, the following is a simple configuration file that generates a CD-ROM image with a Motorola Volume Identification (VID) header followed by an ISO9660 filesystem (ISOFS), followed by a UNIX Filesystem (UNIXS5).

```
#
Slice MOTVID
BootLoader ../boot/romboot
#
# ISO9660 Filesystem
#
Slice ISOFS
Volume ONE
Publisher MOTOROLA INC.
Preparer MOTOROLA INC.
Root /usr/catman
#
# UNIX Filesystem
#
Slice Unixfs
Volume CDVOL1
block 2048
root /user/tmp/cdromdir
```

.fi

CONFIGURATION STATEMENT SYNTAX

Except for comment lines, which are skipped, the only recognized configuration statement *fscoll* initially expects is the slice statement. The slice statement takes the form of a single text line having the keyword **Slice** as the first word on the line, followed by a second keyword that identifies the desired slice type. Currently supported slice type keywords are:

Type String	Meaning
MOTVID	Create a standard 16-block Motorola VID slice with an optional bootloader program. The VID will automatically be configured with parameters suitable for a CD-ROM drive.
UNIXS5	Create a standard UNIX System-V file system slice. By default, all files found in the supplied '**root**' directory are recreated exactly as they were found in the final UNIXS5 file system slice.
ISOFS	Create an ISO9660 compliant CD-ROM Filesystem. Multiple 'Volumes' may be created in an ISOFS Filesystem, each Volume has its own set of configuration parameters.
IMAGE	Create a slice that is an image copy from another device or file.

Once a **Slice** statement is recognized, all statements following that **Slice** statement up to the next recognized **Slice** statement are assumed to apply to that slice. In the case of the **ISOFS** slice, the next statement is expected to be a **Volume** statement, and all statements up to the next **Volume** or **Slice** statement, are considered to apply to that Volume. Configuration statements consist of a keyword, followed by an argument. The argument may be an integer numeric value or a string value. Permitted configuration statements are shown in alphabetical order in the following table.

Keyword	Argument Format	Slice Type(s)	Effect
AbstractFile	*String*	**ISOFS**	Supplies a UNIX file name to be recorded in the ISO9660 Volume Descriptor as the file containing an abstract statement for this volume set. The name must exist as a UNIX file in the supplied Root directory. The name will be translated as required before being recorded in the V.D. By default, there is no AbstractFile.
Application	*String*	**ISOFS**	Supplies a string of up to 128 ISO a-characters to identify the intended application use of the data on the CD-ROM. There is no default value, but secondary Volumes will use the same value as the Primary Volume unless a new string is supplied.
ApplicationFile	*String*	**ISOFS**	Supplies a UNIX file name to be recorded in the ISO9660 Volume Descriptor as the Application Identifier in place of any string supplied by the Application statement above. The name must exist as a UNIX file in the supplied Root directory. The name will be translated as required before being recorded in the V.D. with an underscore prefix as specified in the ISO9660 standard. By default, there is no ApplicationFile.
BibliographicFile	*String*	**ISOFS**	Supplies a UNIX file name to be recorded in the ISO9660 Volume Descriptor as the file containing Bibliographic records. The name must exist as a UNIX file in the supplied Root directory. The name will be translated as required before being recorded in the V.D. with an underscore prefix as specified in the ISO9660 standard. By default, there is no BibliographicFile.

Keyword	Argument Format	Slice Type(s)	Effect
Block	*Integer*	**UNIXS5** **IMAGE** **ISOFS**	Specifies a logical block-size for the slice. Or in the case of IMAGE slices, specifies the block size of the source image. In general, block sizes must be a power of 2 greater than, or equal to 2048. For ISOFS slices, the block size must be 512,1024 or 2048. Default values are chosen depending on slice type.
BootLoader	*String*	**MOTVID**	Supplies a path to the desired bootloader program. The bootloader is included in the MOTVID slice. There is no default BootLoader.
CopyrightFile	*String*	**ISOFS**	Supplies a UNIX file name to be recorded in the ISO9660 Volume Descriptor as the file containing Copyright information. The name must exist as a UNIX file in the supplied Root directory. The name will be translated as required before being recorded in the V.D. with an underscore prefix as specified in the ISO9660 standard. By default, there is no CopyrightFile.
DataType	*String*	**IMAGE**	An optional keyword that specifies the type of data held in an IMAGE source file. The argument is one of three strings, **RAW,ISO** or **BFS**. (Note: The **BFS** keyword is only available on SVR4 versions of these tools). Normally, this keyword is automatically supplied by *fscoll*; it is needed by *fsgen* to allow slice table adjustments for ISO-9660 format image files, and to make internal adjustments to the MOTVID slice.

Keyword	Argument Format	Slice Type(s)	Effect
Description	*String*	**MOTVID**	Supplies a description string for MOTVID slices. This is a string of up to 20 characters, which is recorded in the first block of the image. The default string is ´fsgen CD-ROM image´.
Extensions	*String(s)*	**ISOFS**	Supplies a list of extensions that will be applied to the ISO9660 standard filesystem. At present the only supported argument is XAR. The XAR argument causes extended attribute records to be recorded on the ISO9660 filesystem. By default, no extensions are used but secondary Volumes will use the same extensions set in previous volumes unless changed. To stop the use of any extensions use ´Extensions None´.
FollowList	*String*	**UNIXS5**	Specifies a file containing a list of symbolic links to ´follow´. Symbolic links are normally recreated exactly as found on the output image. A follow list allows the user to specify which symbolic links are followed when creating the **first** file.
FreeInodes	*Integer*	**UNIXS5**	Specifies a MINIMUM number of free inodes to record in a UNIXS5 file system. This statement is usually used in with the **Size** statement. (Not generally useful for CD-ROM file systems).
Packname	*String*	**UNIXS5**	Specifies a UNIX file system pack name. This name is limited to six characters.
Preparer	*String*	**ISOFS**	Supplies a string of up to 128 ISO a-characters to identify the data preparer of this ISO Volume. The default string is ´FSGEN VERSION N.N´. Secondary Volumes will use the same string as the primary volume unless it is specifically changed.

Keyword	Argument Format	Slice Type(s)	Effect
PreparerFile	*String*	**ISOFS**	Supplies a UNIX file name to be recorded in the ISO9660 Volume Descriptor as the Preparer Identifier in place of any string supplied by the Preparer statement above. The name must exist as a UNIX file in the supplied Root directory. The name will be translated as required before being recorded in the V.D. with an underscore prefix as specified in the ISO9660 standard. There is no default PreparerFile.
Publisher	*String*	**ISOFS**	Supplies a string of up to 128 ISO a-characters to identify the data publisher of this ISO Volume. There is no default string, but secondary Volumes will inherit a default value from any previous Volume.
PublisherFile	*String*	**ISOFS**	Supplies a UNIX file name to be recorded in the ISO9660 Volume Descriptor as the Publisher Identifier in place of any string supplied by the Publisher statement above. The name must exist as a UNIX file in the supplied Root directory. The name will be translated as required before being recorded in the V.D. with an underscore prefix as specified in the ISO9660 standard. There is no default PublisherFile.
Root	*String*	**UNIXS5** **ISOFS**	Supplies a base directory location. All files below the base directory are copied into the target filesystem. In the case of an ISOFS slice, certain file types may be ignored as not applicable to the ISO9660 format.
Size	*Integer*	**IMAGE** **UNIXS5**	Specifies a fixed size for an IMAGE or UNIXS5 slice. The size is the number of **Block** sized blocks to record in this slice. In the case of IMAGE slices, the source file may be truncated or zero padded to fit. In the case of UNIXS5 slices, all files must fit or an error is generated by *fsgen* when it is run. Padding space in UNIXS5 slices is recorded as free space in the file system.

Keyword	Argument Format	Slice Type(s)	Effect
Skip	*Integer*	**IMAGE**	Specifies a number of BLOCK sized input blocks to skip when recording this IMAGE slice. The specified number of blocks from the source file is not copied to the image.
Source	*String*	**IMAGE**	Supplies a source path for an IMAGE slice. The source path can be any file or device. It is used as the source of the image for the IMAGE slice type.
System	*String*	**ISOFS**	Supplies an identifier for the system that can recognize and act upon the first 16 blocks of the ISO9660 FileSystem. The default value for the Primary Volume is 'MOTOROLA'. Secondary Volumes will inherit a default value from any previous Volume.
VolCreateDate	*Date*	**ISOFS**	Supplies a date to be used in the ISO Creation Date field of the Volume Descriptor for this Volume. The default value will be the current date and time. Dates are specified using 'MM/DD/CCYY HH:MM:SS' format, but see LIMITATIONS.
VolEffectiveDate	*Date*	**ISOFS**	Supplies a date to be used in the ISO Effective Date field of the Volume Descriptor for this Volume. The default value will be the current date and time.
VolExpireDate	*Date*	**ISOFS**	Supplies a date to be used in the ISO Expiration Date field of the Volume Descriptor for this Volume. By default no expiration date will be set.
VolModDate	*Date*	**ISOFS**	Supplies a date to be used in the ISO Modification Date field of the Volume Descriptor for this Volume. By default the current date and time will be used.

Keyword	Argument Format	Slice Type(s)	Effect
Volume	*String*	**UNIXS5 ISOFS**	Specifies a volume name. The string size limit depends on slice type. For UNIX it is six characters. For ISOFS slices it is 32 ISO d-characters. For ISOFS slices all configuration file lines between one Volume statement and the next are considered to apply to that Volume. Multiple Volume statements are permitted only in ISOFS slices.
VolSet	*String*	**ISOFS**	Supplies an ISO9660 string to be used as the Volume Set identifier in the Volume Descriptor for this Volume. The string may consist of up to 128 ISO d-characters.

.PP Integer arguments should consist only of the digits 0 through 9. The *fscoll* program collects string arguments by scanning any characters that remain on the line after first skipping any white space after the keyword. Since whitespace is regarded as a delimiter, quote strings that contain whitespace with double or single quotes, in which case any quotes within the string, and any backslash characters must also be escaped.

For example, if the desired string is:

```
abcdefg\\ hij''klmn
```

Enter it as:

"abcdefg\ hij`'klmn"

The use of the backslash to introduce a three-digit octal character code is not supported.

OUTPUT FILE FORMAT

The *fscoll* utility generates an output file with a name constructed from the same prefix as the input file. For example, if the input file was named **demo.cnf**, the prefix supplied to the *fscoll* program is **demo** and the output file is named **demo.lst**
.

The **.lst** file contains all configuration information from the original **.cnf** file and all defaulted values. In addition, after each slice's configuration statements, the complete directory and file lists are appended. Each directory or file entry consists of a single character identifier: D(irectory), F(ile), B(lock device), C(haracter device), S(ymbolic link) or P(ipe/FIFO), followed by a space and 13 fields separated by commas. The values in the 13 fields (skipping the prefix character) are arranged as in the following table.

Field	Type	Content
0	*Decimal Integer*	Directory or File index number (identifier)
1	*String*	Filename to be used in the final CD-ROM image. In the case of ISOFS Filesystems, this name will be an ISO conforming translation of the original UNIX file name.
2	*String*	UNIX Pathname to be used as the source file.
3	*Decimal Integer*	Parent index number. Indicates the parent directories index.
4	*Decimal Integer*	Link Index. If a file can be recreated as a link, this field holds the decimal file index of the target file plus one.
5	*Decimal Integer*	Size. This field holds the file size of the source file.
6	*Octal Integer*	Mode. This field holds the mode bits of the original file. (see *chmod*(2)).
7	*Decimal Integer*	Userid. This field holds the original User I.D. of the source file.
8	*Decimal Integer*	Groupid. This field holds the original Group I.D. of the source file.
9	*Decimal Integer*	Dev. If the source file was a device node, this field holds the *mknod*(2) device I.D.
10	*Decimal Integer*	Access Time. The last access time of the file, represented as the number of seconds since 00:00 1/1/1970.
11	*Decimal Integer*	Modification Time. The last modification time of the file, represented as the number of seconds since 00:00 1/1/1970.
12	*Decimal Integer*	Create Time. The creation time of the file, represented as the number of seconds since 00:00 1/1/1970.

NOTE

Where file or path names contain backslashes, quotes (") or commas, they are automatically escaped with extra '\'s.

The user supplied comments in the original **.cnf** file are *not* copied to the **.lst** file. Certain comments are automatically supplied by *fscoll* in the **.lst** file to help break up the output and identify the individual slices.

ISO9660 NAME TRANSLATION

One of the functions of *fscoll* when processing **ISOFS** slices, is to perform automatic name translation from UNIX file names to ISO9660 file names. Because the ISO9660 standard is far more restrictive than UNIX on file-name length and content, the name translation process involves making names unique by adding integer extensions onto the file name.

For example, if we have two unix files named **unixfileone** and **unixfiletwo** then *fscoll* will initially translate both names into ISO9660 format by changing them both into **UNIXFILE.;1.** Obviously, two files with the same name cannot be allowed in the same directory, so *fscoll* will then 'increment' the name of the second file it finds, turning it into **UNIXFIL1.;1.** Subsequent name clashes will generate **UNIXFIL2.;1** then **UNIXFIL3.;1** and so on down the series.

The only way to avoid name translation altogether, is to make sure the initial name is a valid ISO9660 file name. Provided the name does not clash with one of the already translated names collected into the same directory, *fscoll* will not perform any further translation. Remember, directory names must not use ISO9660 extensions, but file names must have both the extension and the version number.

For more information on ISO9660 file names, consult the ISO9660 standard.

SYMBOLIC LINKS IN UNIXS5 SLICES

The default procedure when building a **.lst** file is to recreate all symbolic links as found in the original ´**root**´ directory. In some cases, it may be desirable to follow a symbolic link. In other words, to create it as a ´real´ file (or directory) on the resulting image. This allows, for example, the use of symbolic links to collect certain files and directories into the '**root**' directory without having to physically copy them.

The **FollowList** statement in a UNIXS5 slice configuration supplies the name of a file that holds a simple list of symbolic links that the user wants to be followed rather than recreated on the target image. The file can contain comments prefixed with ´**#**´ and paths listed should be relative to the supplied '**root**' directory with no preceding ´/´. For example, if the following command is typed while in the **/user/tmp/cdromdir** directory and the user wants all the files in **/usr/bin** to be recreated in the final image, the user must supply a **FollowList** specifying to follow **mysymlink**.

 # ln -s /usr/bin mysymlink

First, add an entry to the **.cnf** file:

```
#
Slice MOTVID
BootLoader ../boot/romboot
#
# ISO9660 Filesystem
#
Slice ISOFS
Volume ONE
Publisher MOTOROLA INC.
Preparer MOTOROLA INC.
Root /usr/catman
#
# UNIX Filesystem
#
Slice Unixfs
Volume CDVOL1
block 2048
root /user/tmp/cdromdir
FollowList CDVOL1.flw
```

Then edit the **CDVOL1.flw** file to contain:

```
#
# Symlink follow list for CDVOL1
#
mysymlink
```

Note that the path to the **FollowList** file is specified relative to the directory in which *fscoll* is run. It is not necessarily in the same directory as the **.cnf** file. As each entry in a **FollowList** file is read, the *fscoll* program checks that it exists and that it is a symbolic link. If either of these tests fail, the program exits with an error.

WARNINGS AND DIAGNOSTICS.

Most error and warning diagnostics are intended to be self-explanatory. It is not possible to list here all the illegal and out of range configuration values because they depend on slice type. In general, configuration error checking in *fscoll* is extensive and diagnostics should give both the problem, the *errno* number (where appropriate), and the line number location in the input file. After *fscoll* has run, a log file is created, named *fileprefix*.**log**. This file contains all warning and error messages displayed during the execution of *fscoll*. Always examine this file for possible problems.

fscoll exits with a value of 0 unless a fatal error was encountered. In the case of an error, exit values are picked from the following table, depending on the class of problem. This set of exit codes is common to all the CD-ROM generation tools.

Error Class	Exit Value
Incorrect user input (**.cnf** file problem)	10 (BADCONF)
Out of memory, or corrupted memory detected	11 (BADMEM)
File access problems (including missing files)	12 (BADFILE)
I/O operation errors	13 (BADIO)
Illegal input data format	14 (BADFMT)
Internal program errors detected	15 (INTERR)

LIMITATIONS

On Motorola machines, there is currently no way to mount secondary volumes from an ISO filesystem. If an ISO9660 Filesystem is being generated for use Motorola machine, then only one Primary Volume should be used, with no secondary **Volume** statements.

Fscoll and *fsgen* currently only support interchange level 1 as described in the ISO9660 document. This means that all file names recorded in an ISOFS slice will have a maximum of 8 characters, with a maximum extension of 3 characters.

CD-ROM Volume Sets, or the recording of a file tree over several physical CD-ROM volumes are not supported. Each ISO9660 Volume recorded by *fsgen* will be logically complete.

Due to way dates are stored internally, no expiration date later then January 18th 2038 can be specified. Keep all dates earlier than this.

SEE ALSO

fsgen(1), loadimg(1).
CD-ROM Image Generation Tools User's Guide

NAME

fsdb (generic) - file system debugger

SYNOPSIS

fsdb [-F *FSType*] [-V] [*current_options*] [-o *specific_options*] *special*

DESCRIPTION

fsdb is a file system debugger which allows for the manual repair of a file system after a crash. *special* is a special device used to indicate the file system to be debugged. fsdb is intended for experienced users only. *FSType* is the file system type to be debugged. Since different *FSTypes* have different structures and hence different debugging capabilities the manual pages for the *FSType*-specific fsdb should be consulted for a more detailed description of the debugging capabilities.

current_options are options supported by the s5-specific module of fsdb. Other *FSTypes* do not necessarily support these options. *specific_options* indicate suboptions specified in a comma-separated list of suboptions and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command. See fsdb_*FSType*(1M) for details.

The options are:

-F Specify the *FSType* on which to operate. The *FSType* should either be specified here or be determinable from /etc/vfstab by matching the *special* with an entry in the table.

-V Echo the complete command line, but do not execute the command. The command line is generated by using the options and arguments provided by the user and adding to them information derived from /etc/vfstab. This option should be used to verify and validate the command line.

-o Specify *FSType*-specific options.

NOTE

This command may not be supported for all *FSTypes*.

FILES

/etc/vfstab list of default parameters for each file system

SEE ALSO

fsdb_s5(1M), fsdb_ufs(1M), mkfs(1M), vfstab(4).

NAME

`fsdb` (s5) - `s5` file system debugger

SYNOPSIS

`fsdb` [`-F s5`] [*generic_options*] [`-z` *i-number*] *special* [-]

DESCRIPTION

generic_options are options supported by the generic `fsdb` command.

`fsdb` can be used to patch up a damaged `s5` file system after a crash. *special* is a special device used to indicate the file system to be debugged. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

`fsdb` contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking `fsdb` with the optional - argument or by the use of the `O` symbol. (`fsdb` reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

The options are:

`-F s5`	Specifies the `s5`-FSType.
`-z` *i-number*	Clear the i-node identified by *i-number*. Non-interactive.

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

`fsdb` reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by `fsdb` are:

`#`	absolute address
`i`	convert from i-number to i-node address
`b`	convert to block address
`d`	directory slot offset
`+,-`	address arithmetic
`q`	quit
`>,<`	save, restore an address
`=`	numerical assignment
`=+`	incremental assignment
`=-`	decremental assignment
`="`	character string assignment
`O`	error checking flip flop
`p`	general print facilities
`f`	file print facility
`B`	byte mode
`W`	word mode

D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the p symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The f symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the f symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A .B or .D is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a #	data block numbers (0 - 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.

`ln=4`	changes the link count for the working i-node to 4.
`ln=+1`	increments the link count by 1.
`fc`	prints, in ASCII, block zero of the file associated with the working i-node.
`2i.fd`	prints the first 32 directory entries for the root i-node of this file system.
`d5i.fc`	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
`512B.p0o`	prints the superblock of this file system in octal.
`2i.a0b.d7=3`	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
`d7.nm="`*name*`"`	changes the name field in the directory slot to the given string. Quotes are optional when used with `nm` if the first character is alphabetic.
`a2b.p0d`	prints the third block of the current i-node as directory entries.

INTERNATIONAL FUNCTIONS

`fsdb` can process characters from supplementary code sets.

Use of the `f` or `="` symbols do not display multibyte characters correctly.

SEE ALSO

`fsck`(1M), generic `fsdb`(1M), `dir`(4), `fs`(4)

NAME

`fsdb` (ufs) - `ufs` file system debugger

SYNOPSIS

`fsdb` [`-F ufs`] [*generic_options*] `-z` *i-number special*

DESCRIPTION

generic_options are options supported by the generic `fsdb` command.

The options are:

`-F ufs`
: Specifies the `ufs`-FSType.

`-z` *i-number*
: Clear the i-node identified by *i-number*. Non-interactive.

SEE ALSO

`fsck`(1M), `fsdb`(1M), `dir`(4), `fs`(4).

NAME

fsgen - Filesystem image generator - Pass 2

SYNOPSIS

fsgen *[-Vcvwa] [-f outfile] fileprefix*

SUMMARY

fsgen reads a **.lst** file as produced by *fscoll*(1) and prepares one or more filesystem images from the information contained in the file. The images are written sequentially to any specified file or device. The *fsgen* program also writes a **.log** file, that contains all warning and error messages, as well as final image sizing information.

Fsgen is capable of creating both UNIX and ISO9660 format filesystems, as well as simple image slices and Motorola style VID blocks.

DESCRIPTION

fsgen expects to be given a file name prefix as an argument. The prefix is used to construct an input file name by adding the string **.lst** to the prefix supplied. If the constructed file name cannot be opened, *fsgen* aborts with an error.

The user may supply any of the following command-line options:

-V causes *fsgen* to print its version number and exit.

-v causes *fsgen* to generate brief messages about the slices currently being processed and some sizing information, including the total size of the final image. The total image size is also always written to the **.log** file.

-w causes *fsgen* to treat the output image as if it is intended for a writable device (not a CD-ROM). Currently the only effect of this flag is to turn off the V_RONLY flag in the SVR4 VTOC. It thus has no effect unless the image is mounted under SVR4.

-a causes *fsgen* to run only the first allocation pass on the input **.lst** file. The output device is not opened, and no image is created. The allocation and sizing information for the image is recorded in the **.log** file in the usual way. Because the allocation pass is fast, it is a good idea to use the **-a** option for the initial run because it identifies many problems that cause a full image write to fail. Also use the **-a** option to check that an image is not too big to fit on the final device. For example the CD-ROM size limit is usually around 600MB, but varies between different manufacturers.

-f device
allows the user to supply an output device or file to which the image will be written. If no **-f** option is supplied, the default device **/dev/rmt/ctape** is always used.

-c option (continue), causes *fsgen* to try and ignore all file open and read errors during the image creation process. Any failed attempts to open a file or read a file that made up part of a UNIXS5 or ISOFS slice causes that file to be written either partially or completely empty (all zeros). A suitable warning message is generated in all cases.

If the output file is a tape device, and *fsgen* finds that it cannot fit the entire image on one tape, *fsgen* prompts the user to insert a new tape at the correct time. The correct operation of multi-tape images relies on using the correct, BCS compliant, tape device node.

DIAGNOSTICS

Since *fsgen* expects that the *fscoll* utility has already checked the information in the input **.lst** file, its error checking is not as thorough as in *fscoll.* However, all error messages detail the problem encountered, the input file line number (where relevant), and the UNIX *errno* (if applicable).

All error and warning messages printed by *fsgen* during its execution are also copied to a file named **xxx.log** (where 'xxx' is replaced by *fileprefix*). This **.log** file should always be carefully examined after *fsgen* has been run. Any fatal errors mean that the image is unusable. Any warnings usually mean that the image is complete, but not correct in data content.

fsgen always has a zero exit value unless an error was encountered. The exit value on encountering an error is picked from the following table based on the class of error. This same set of exit values is used by all the CD-ROM generation tools.

Error Class	Exit Value
Incorrect user input (**.cnf** file problem)	10 (BADCONF)
Out of memory, or corrupted memory detected	11 (BADMEM)
File access problems (including missing files)	12 (BADFILE)
I/O operation errors	13 (BADIO)
Illegal input data format	14 (BADFMT)
Internal program errors detected	15 (INTERR)

LIMITATIONS

For ISO9660 filesystems (ISOFS) *fscoll* and *fsgen* currently only support interchange level 1 as described in the ISO9660 document. This means that all filenames recorded in an ISOFS slice will have a maximum of 8 characters, with a maximum extension of 3 characters.

CD-ROM Volume Sets, or the recording of a file tree over several physical CD-ROM volumes are not supported. Each ISO9660 Volume recorded by *fsgen* will be logically complete.

BUGS

The multi-tape support may not always work correctly on the MVME350 tape controller.

SEE ALSO

fscoll(1), loadimg(1).

NAME

fsirand - install random inode generation numbers

SYNOPSIS

/usr/ucb/fsirand [-p] *special*

DESCRIPTION

fsirand installs random inode generation numbers on all the inodes on device *special*, and also installs a filesystem ID in the superblock. This helps increase the security of filesystems exported by NFS.

fsirand must be used only on an unmounted filesystem that has been checked with fsck(1M). The only exception is that it can be used on the root filesystem in single-user mode, if the system is immediately re-booted afterwords.

The -p option prints out the generation numbers for all the inodes, but does not change the generation numbers.

SEE ALSO

fsck(1M).

NAME

fstyp (generic) - determine file system type

SYNOPSIS

fstyp [-v] *special*

DESCRIPTION

fstyp allows the user to determine the file system type of unmounted file systems using heuristic programs.

An fstyp module for each file system type to be checked is executed; each of these modules applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; if none of the modules succeed, the error message unknown_fstyp (no matches) is returned and the exit status is 1. If more than one module succeeds the error message unknown_fstyp (multiple matches) is returned and the exit status is 2.

The options are:

-v Produce verbose output. This is usually information about the file systems superblock and varies across different *FSTypes*.

NOTES

The use of heuristics implies that the result of fstyp is not guaranteed to be accurate.

NAME

ftp - file transfer program

SYNOPSIS

ftp [-dgintv] [*hostname*]

DESCRIPTION

The ftp command provides the user interface to the ARPANET standard File Transfer Protocol (FTP). The ftp command transfers files to and from a remote network site.

The host with which ftp is to communicate may be specified on the command line. If this is done, ftp will immediately attempt to establish a connection to an FTP server on that host; otherwise, ftp will enter its command interpreter and then await instructions from the user. When ftp is awaiting commands from the user, it will display the prompt ftp>.

OPTIONS

The following options may be specified at the command line, or to the command interpreter:

-d Enable debugging.

-g Disable filename "globbing".

-i Turn off interactive prompting during multiple file transfers.

-n Do not attempt auto-login upon initial connection. If auto-login is not disabled, ftp will check the .netrc file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, ftp will prompt for the login name of the account on the remote machine (the default is the login name on the local machine); if necessary, ftp will prompt for an account for the login and for a password.

-t Enable packet tracing (currently not implemented).

-v Show all responses from the remote server, as well as provide a report on data transfer statistics. This is turned on by default if ftp is running interactively with its input coming from the user's terminal.

COMMANDS

The following commands can be specified to the command interpreter:

! [*command* [*args*]]
Run command as a shell command on the local machine. If there are any arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

$ *macro-name* [*args*]
Execute the macro *macro-name* that was defined with the macdef command. Any arguments will be passed to the macro in "unglobbed" format.

account [*passwd*]
Supply a supplemental password required by a remote system for access to its resources once a login has been completed successfully. If no argument is included, the user will be prompted for an account password in a "non-echoing" input mode.

`append` *local-file* [*remote-file*]
Append the *local-file* to a file on the remote machine. If *remote-file* is not specified, the local file name will be used, subject to alteration by any `ntrans` or `nmap` setting. The file transfer will use the current settings for the "representation type", the "format", the "file structure", and the "transfer mode".

`ascii` Set the "representation type" (or "file transfer type") to "network ASCII".

`bell` Sound a bell after completing each file transfer command.

`binary`
Set the "representation type" (or "file transfer type") to support binary image transfers (the default value).

`bye` Terminate the FTP session with the remote server and exit `ftp`. An EOF condition will also terminate the session and exit.

`case` Toggle the remote computer file name case mapping during `mget` commands. When `case` is on (the default is off), the remote computer file names with all letters in "upper case" will be written in the local directory with the letters mapped to "lower case".

`cd` *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

`cdup` Change the remote machine working directory to the parent of the current remote machine working directory.

`close` Terminate the FTP session with the remote server and return to the command interpreter. Any defined macros will be erased.

`cr` Toggle the "Carriage Return" stripping during "network ASCII" type file retrieval. Each end-of-record is denoted by a CARRIAGE RETURN/LINEFEED sequence during a "network ASCII" type file transfer. When `cr` is on (the default setting), the CARRIAGE RETURN characters are stripped from this sequence for consistency with the UNIX Operating System's single LINEFEED record delimiter. Records on non-UNIX-system remote hosts may contain single LINEFEED characters; during "network ASCII" type file transfers these LINEFEED characters may be distinguished from a record delimiter only when `cr` is off.

`delete` *remote-file*
Delete the file *remote-file* on the remote machine.

`debug` Toggle the debugging mode. When debugging is on, `ftp` will print each command sent to the remote machine, preceded by the string `-->`.

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory *remote-directory*; optionally, place the output into *local-file*. If no directory is specified, the current working directory on the remote machine will be used. If no local file is specified (or if *local-file* is -) the output will be sent to the terminal.

`disconnect`
A synonym for `close`.

`form` [*format-name*]
Set the carriage control format subtype of the "representation type" to *format-name*. The only valid *format-name* is `non-print`, which corresponds to the default "non-print" subtype.

`get` *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name as that on the remote machine, subject to alteration by the current `case`, `ntrans`, and `nmap` settings. The current settings for "representation type", "file structure", and "transfer mode" will be used while transferring the file.

`glob` Toggle the filename expansion ("globbing") mechanism for `mdelete`, `mget`, and `mput`. If "globbing" is turned off, the filenames are taken literally.

"Globbing" for `mput` is done as in `sh`(1). For `mdelete` and `mget`, each remote file name is expanded separately on the remote machine, but the lists are not merged.

Expansion of a directory name is likely to be radically different from an expansion of the name of an ordinary file: the exact result depends on the remote operating system and on the FTP server; this action can be previewed by doing the following: `mls` *remote-files* -

`hash` Toggle the hash-sign (`#`) which will be printed for each data block transferred. The size of a data block is 8192 bytes.

`help` [*command*]
Print an informative message about the meaning of `command`. If no argument is given, `ftp` will print a list of the known `ftp` commands.

`lcd` [*directory*]
Change the working directory on the local machine; if no *directory* is specified, the user's home directory will be used.

`ls` [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of *remote-directory* on the remote machine. The listing will include any system-dependent information which the server chooses to include: for example, most UNIX Operating Systems will produce output from the command "`ls -l`. (See `nlist`) If the *remote-directory* argument is not specified, the current working directory will be used. If interactive prompting is on, `ftp` will prompt the user to verify that the last argument is indeed the target *local-file* for receiving the `ls` output. If *local-file* is not specified (or if *local-file* is `-`) the output will be sent to the terminal.

`macdef` *macro-name*
Define a macro; all subsequent lines are stored as the macro *macro-name* until a null line (consisting of consecutive NEWLINE characters in a file or consecutive CARRIAGE_RETURN characters from the terminal) terminates the macro input mode. There is a limit of 16 macros, as well as a maximum of 4096 characters per macro definition. All macros will remain defined until a `close` command is executed.

The macro processor interprets $ and \ as special characters. A $ followed by a number (or numbers) will be replaced by the corresponding argument on the macro invocation command line. A $ followed by an i signals to the macro processor that the executing macro is to be looped. On the first pass $i will be replaced by the first argument on the macro invocation command line; on the second pass it will be replaced by the second argument, and so on. A \ followed by any character is replaced by that character. Use the \ to prevent special treatment of the $ symbol.

`mdelete` [*remote-file*]
Delete the *remote-file* on the remote machine.

`mdir` *remote-file local-file*
Like `dir`, but multiple remote files may be specified. If interactive prompting is on, `ftp` will prompt the user to verify that the last argument is indeed the target local file for receiving `mdir` output.

`mget` *remote-files*
Expand the *remote-files* on the remote machine and do a `get` for each file name thus produced. [See `glob` for details on the filename expansion.] The resulting file names will then be processed according to the prevailing `case`, `ntrans`, and `nmap` settings. The files will be transferred into the local working directory, which can be changed with `lcd` *directory*; new local directories can be created with `! mkdir` *new_directory*.

`mkdir` *directory-name*
Make a directory *directory-name* on the remote machine.

`mls` *remote-files local-file*
Similar to `ls`(1), but multiple remote files may be specified. If interactive prompting is on, `ftp` will prompt the user to verify that the last argument is indeed the target local file for receiving the `mls` output.

`mode` [*mode-name*]
Set the "transfer mode" to *mode-name*. The only valid *mode-name* is `stream`, which corresponds to the default "stream mode". This implementation only supports `stream` and requires that it be specified.

`mput` *local-files*
Expand "wild cards" in the list of *local-files* given as arguments and do a `put` for each file in the resulting list. [See `glob` for details of the filename expansion mechanism.] The resulting file names will then be processed according to the prevailing `ntrans` and `nmap` settings.

`nmap` [*inpattern outpattern*]
Enable (or disable) the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If any arguments are specified, the remote filenames will be mapped during the `mput` commands, as well as for `put` commands issued without a specified remote target filename. If any arguments are specified, the local filenames will be mapped during `mget` commands, as well as for `get` commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. Here *inpattern* is a template for incoming filenames (which may have already been processed according to the `ntrans` and `case` settings). Variable templating is accomplished by including the sequences `$1`, `$2`, . . . , `$9` in *inpattern*. Use "`\`" to prevent this special treatment of the `$` character. All other characters will be treated literally and will be used to determine the variable values which correspond to `nmap` *inpattern*.

For example, given the *inpattern* `$1.$2` and the remote file name `mydata.data`, then `$1` would have the value `mydata` and `$2` would have the value `data`.

Similarly, the *outpattern* determines the resulting mapped filename. The sequence `$1`, `$2`, . . . , `$9` will be replaced by any value resulting from the *inpattern* template. The sequence `$0` will be replaced by the original filename. In addition, the sequence "`[seq1,seq2]`" will be replaced by `seq1` if `seq1` is not a null string; otherwise it will be replaced by `seq2`.

For example, assume that the command `nmap $1.$2.$3 [$1,$2].[$2,file]` is given. Hence, for input filenames `myfile.data` and `myfile.data.old` this would yield the output filename `myfile.data`. Similarly, the input filename `myfile` would generate output `myfile.file`; and finally, an input filename `myfile` would produce output `myfile.myfile`.

The *outpattern* may include embedded SPACE characters, as in the example `nmap $1 | sed "s/ *$//" > $1`. (Use the `\` symbol to prevent special treatment of the "`$`", "`[`", "`]`", and "`,`" symbols.)

`ntrans` [*inchars* [*outchars*]]
Enable or disable the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is disabled. If arguments are specified, the characters in remote filenames are translated during `mput` commands, as well as for `put` commands issued without a specified remote target filename; characters in local filenames are translated during `mget` commands, as well as for `get` commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX-system remote host with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character will be deleted from the file name.

`open` *host* [*port*]
Establish a connection to the specified *host* FTP server. If an optional port number is supplied, `ftp` will attempt to contact an FTP server at that port. If the `auto-login` option is on (the default setting), `ftp` will also attempt to automatically log the user into the FTP server. (See below)

`prompt`
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to retrieve or store files selectively. By default, prompting is turned on. If prompting is turned off, any `mget` or `mput` command will transfer all files; any `mdelete` will delete all files.

`proxy` *ftp-command*
Execute an FTP command on a secondary control connection. This command allows the simultaneous connection to two remote FTP servers for transferring files between the two servers. The first `proxy` command should be an `open` which should establish the secondary control connection. Then enter the command `proxy ?` to see other FTP commands executable on the secondary connection.

`put` *local-file* [*remote-file*]
Store a *local-file* on the remote machine. If *remote-file* is not specified, the *local-file* name will be used after processing according to any prevailing `ntrans` or `nmap` settings for naming the *remote-file*. The file transfer uses the current settings for "representation type", "file format", "file structure", and "transfer mode".

`pwd` Print the name of the current working directory on the remote machine.

`quit` A synonym for `bye`.

`quote` *arg1 arg2 ...*
Send the specified arguments "verbatim" to the remote FTP server. A single FTP reply code is expected in return. (The `rhelp` command will display a list of valid arguments.)

`recv` *remote-file*1 [*local-file*]
A synonym for `get`.

`rhelp` [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is also supplied to the server.

`rstatus` [*file-name*]
If no argument is given, show the status of the remote machine. If *file-name* is specified, show the status of *file-name* on the remote machine.

`rename` [*from-name*] [*to-name*]
Rename the file *from-name* on the remote machine to the name *to-name*.

`reset` Clear the reply queue: this command re-synchronizes the command/reply sequences with the remote FTP server. Re-synchronization may be necessary as a result of detecting a violation of the FTP protocol by the remote server.

`rmdir` *directory-name*
Delete *directory-name* on the remote machine.

`runique`
Toggle the storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a `get` or `mget` command, then a `.1` will be appended to this name. If the resulting name matches another existing file, a `.2` will be appended to the original name. If this process continues up to `.99`, an error message will be printed and the file transfer will not occur. The generated unique filename will be reported.

`send` *local-file* [*remote-file*]
A synonym for `put`.

`sendport`
Toggle the use of the `PORT` commands. By default, `ftp` will attempt to use a `PORT` command when establishing a connection for each data transfer. The use of the `PORT` commands can prevent delays when performing multiple file transfers. If the `PORT` command fails, `ftp` will use the default data port. When the use of `PORT` commands is disabled, no attempt will be made to use `PORT` commands for each data transfer.

This is useful when connected to certain FTP implementations that ignore `PORT` commands, but incorrectly indicate they have been accepted.

`size` *file-name*
Return the size of *file-name* on the remote machine.

`status`
Show the current status of `ftp`.

`struct` [*struct-name*]
Set the file structure parameter to *struct-name*. The only valid *struct-name* is `stream`, which corresponds to the default "file" structure. The implementation only supports `file` and requires that it be specified.

`sunique`
Toggle the storing of files on a remote machine under unique file names. The remote FTP server must support the `ftp STOU` command for successful completion. The remote server will report the unique name. The default value for `sunique` is off.

`system`
Show the type of Operating System running on the remote machine.

`tenex` Set the "representation type" for talking to TENEX machines.

`trace` Toggle packet tracing (currently not unimplemented).

`type` [*type-name*]
Set the "file transfer type" to *type-name*. If *type-name* is not specified, the current type will be printed. The valid *type-names* are

ascii for "network ASCII" (the default value),
binary or image for "image", and
tenex for "local byte size" with a byte size of 8 (used for talking to TENEX machines).

`user` *user-name* [*password*] [*account*]
: Identify yourself to the remote FTP server. If the password is not specified, but the server requires it, `ftp` will prompt the user for it (after disabling local echo). If an account field is not specified and the FTP server requires it, the system will prompt the user for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless `ftp` is invoked with `auto-login` disabled, this process is done automatically on initial connection to the FTP server.

`verbose`
: Toggle the verbose mode: In verbose mode, all responses from the FTP server will be displayed to the user. In addition, if the verbose mode is enabled when a file transfer completes, statistics about the efficiency of the transfer will be reported. By default, the verbose mode will be enabled if the `ftp`'s commands are coming from a terminal and disabled otherwise.

`?` [*command*]
: A synonym for `help`.

THE `.netrc` FILE

The `.netrc` file contains login and initialization information for use by the `auto-login` process; this file should reside in the user's home directory. The following tokens will be recognized and may be separated by spaces, tabs, or new-lines:

`machine` *name*
: Identify the name of a remote machine: The `auto-login` process will search the `.netrc` file for a `machine` token that matches the remote machine as specified in the `ftp` command line or as an argument to an `open` command. Once a match has occurred, the subsequent `.netrc` tokens are processed until EOF is found or until another `machine` token is encountered.

`default`
: This is the same as `machine` *name*, except that `default` matches any name. There can be only one `default` token and it must occur after all other `machine` tokens. Normally, this is used as:

```
default login anonymous password user@site
```

This command line would give a user "automatic" anonymous `ftp` login privileges to machines not specified in `.netrc`. The `-n` flag can be used to override this capability by disabling `auto-login`.

`login` *login-name*
: Identify a user on the remote machine. If this token is present, the `auto-login` process will initiate a login procedure using the specified *login-name*.

`password` *password-string*
: Supply a password for this login. If this token is present, the `auto-login` process will supply the specified *password-string* if the remote server requires a password as part of the login procedure.

`account` *account-string*
Supply an additional account password. If this token is present, the `auto-login` process will supply the specified *account-string* if the remote server requires an additional account password; otherwise the `auto-login` process will initiate an ACCT command to solicit this input.

macdef-name
Define a macro. This token functions like the `ftp macdef` command functions. A macro will be defined for the specified *macdef-name*; its contents will begin with the next `.netrc` line and continue until a NULL line (consisting of consecutive new-line characters) is found. If a macro named `init` is defined, it will be executed automatically as the last step of the `auto-login` process.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually ^C). Sending transfers will be halted immediately; receiving transfers will be halted by sending an FTP protocol ABOR command to the remote server and then discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an `ftp>` prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when `ftp` has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from some unexpected behavior by the remote server, including any violations of the FTP protocol. If the delay results from unexpected remote server behavior, the local `ftp` program must be killed by hand.

FILE NAMING CONVENTIONS

Local files specified as arguments to `ftp` commands will be processed according to the following rules.

1) If the file name - is specified, the standard input (for reading) or standard output (for writing) will be used.

2) If the first character of the file name is `|`, the remainder of the argument will be interpreted as a shell command. `ftp` then will fork a shell, using `popen`(3S) with the argument supplied and read (write) from the standard output (standard input) of that shell. If the shell command includes any embedded space characters, the argument must be quoted; for example, `"| ls -lt"`. A particularly useful example of this mechanism is: `"dir | more"`.

3) Failing the above checks and if "globbing" is enabled, local file names will be expanded according to the rules used in the `sh`(1); see the `glob` command. If the `ftp` command expects a single local file (for example, `put`), only the first filename generated by the "globbing" operation will be used.

4) For `mget` commands and `get` commands with unspecified local file names, the local filename will be the remote filename, which may be altered by the prevailing `case`, `ntrans`, or `nmap` settings. The resulting filename may be altered even further if `runique` is enabled.

5) For mput commands and put commands with unspecified remote file names, the remote filename will be the local filename, which may be altered by a prevailing ntrans or nmap setting. The resulting filename may be altered even further by the remote server if sunique is enabled.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. For example, the "representation type" may be one of the following:

"network ASCII",
"EBCDIC",
"image", or
"local byte size" (normally with a specified byte size for PDP-10 and PDP-20 systems).

The "network ASCII" and "EBCDIC" types have a further subtype which specifies whether vertical format control data (i.e., NEWLINE characters, form feeds, etc.) is to be passed through (the "non-print" option), provides the TELNET format (i.e., "TELNET format controls"), or provides the ASA (FORTRAN) ["carriage control (ASA)"] format.

ftp supports the "network ASCII" [subtype "non-print" only) and "image" types, as well as "local byte size" with a byte size of 8 for communicating with TENEX machines.

The "file structure" may be one of "file" (i.e., no record structure), "record", or "page", but ftp supports only the "file" option (the default value).

The "transfer mode" may be one of "stream", "block", or "compressed", but ftp supports only the "stream" option (the default value).

SEE ALSO

ls(1), sh(1), tar(1), ftpd(1M), rcp(1N), popen(3S), netrc(4N).

USER CONSIDERATIONS

The correct execution of many commands will depend on the proper behavior by the remote server.

An error in the treatment of CARRIAGE RETURN symbols in the 4.2 BSD code handling transfers with a "representation type" of "network ASCII" has been corrected. However, this correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using a "representation type" of "network ASCII". Therefore avoid this problem by using the "image" type.

NOTES

The mget and mput command options are not meant to transfer entire directory subtrees of files. Instead, you can do this by transferring a tar(1) archive of the sub-

tree (using a "representation type" of "image" as set by the `binary` command).

The following commands behave differently when preceded by `proxy`:

- `open` will not define new macros during the `auto-login` process,
- `close` will not erase existing macro definitions,
- `get` and `mget` will transfer files from the host on the primary control connection to the host on the secondary control connection
- `put`, `mputd`, and `append` will transfer files from the host on the secondary control connection to the host on the primary control connection.

Third-party file transfers will depend on the support of the `PASV` command by the FTP server on the secondary control connection.

`quote` should be utilized only by experienced users familiar with the FTP protocol.

The `runique` capability will not affect local files generated from a shell command (see below). The default value for `runique` is off.

Any command argument containing embedded spaces should be surrounded by quote (") marks.

If any non-optional command argument is not specified, `ftp` will prompt for that argument.

If the `password` token is present in the `.netrc` file, `ftp` will abort this login procedure if the `.netrc` file is readable by anyone other than the user.

NAME

ftpd - DARPA Internet File Transfer Protocol server

SYNOPSIS

in.ftpd [-d] [-l] [-t *timeout*] [-T *maxtimeout*] *host.socket*

DESCRIPTION

The ftpd command is the Internet File Transfer Protocol (FTP) server process. The server is invoked by the Internet daemon inetd(1M) whenever a connection to the (FTP) service [see services(4)] is made: the connection will be available as descriptor 0; the host and socket from which the connection originated (in hexadecimal and in decimal respectively) will appear as an argument.

The FTP server will time out an inactive connection after 15 minutes.

OPTIONS

The following options are available:

-d Write debugging information to the syslog file.

-l Log each FTP session in the syslog.

-t *timeout* If the -t option is specified, the inactivity timeout period will be set to *timeout* seconds, with a default value of 15 minutes.

-T *maxtimeout*
If the -T option is specified, the maximum inactivity timeout period will be set to *maxtimeout* seconds, with a default value of 2 hours.

FTP REQUESTS

The FTP server currently supports the following FTP requests (without distinguishing between upper and lower case):

Request	*Description*
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -lg")
MKD	make a directory
MDTM	show last modification time of this file
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing

PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port
PWD	print the current working directory
QUIT	terminate session
REST	restart incomplete transfer
RETR	retrieve a file
RMD	remove a directory
RNFR	specify file name *rename-from*
RNTO	specify file name *rename-to*
SITE	non-standard commands (see next section)
SIZE	return the size of the file
STAT	return the status of the file
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer *structure*
SYST	show the Operating System type of the server system
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory (deprecated)
XCWD	change working directory (deprecated)
XMKD	make a directory (deprecated)
XPWD	print the current working directory (deprecated)
XRMD	remove a directory (deprecated)

The following non-standard UNIX-specific commands are supported by the SITE request:

Request	*Description*
UMASK	change umask (e.g., SITE UMASK 002)
IDLE	set idle-timer (e.g., SITE IDLE 60)
CHMOD	change mode of a file (e.g., SITE CHMOD 755 *filename*)
HELP	give help information (e.g., SITE HELP)

The remaining FTP requests specified in RFC 959 are recognized, but not implemented. MDTM and SIZE are not specified in RFC 959, but are expected to appear in the next FTP RFC.

The FTP server will abort an active file transfer only when the ABOR command is preceded by a TELNET "Interrupt Process" (IP) signal and by a TELNET "Synch" signal in the command stream (as described in RFC 959).

ftpd will interpret the file names according to the "globbing" conventions used by sh(1). This will allow users to utilize the following metacharacters: "*", "?", "[", "]", "{", "}", and "~".

ftpd will authenticate users according to the following rules:

1) The user name must be in the password data base, /etc/passwd, and not have a null password. Otherwise, a password must be provided by the client before any file operations may be performed.

2) If the user name appears in the file /etc/ftpusers, FTP access will be denied.

3) ftp access will be denied unless the user has a standard shell returned by getusershell(3), or the user's shell (from /etc/passwd) is listed in the file /etc/shells, or the user's shell is one of the following:
```
/bin/sh
/bin/ksh
/bin/csh
/usr/bin/sh
/usr/bin/ksh
/usr/bin/csh
```

4) If the user name is "anonymous" or "ftp", an anonymous FTP account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this will be given as the user's electronic mail address).

In the last case, ftpd will take special measures to restrict the client's access privileges. The server will perform a chroot(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with special care: the following rules are recommended.

home_directory
Make the home directory owned by "ftp" and unwritable by anyone else.

home_directory/bin
Make this directory owned by the super-user and unwritable by anyone. The program ls(1) must be present to support the list commands. This program should have mode 111.

NOTE: Since ftpd does a chroot(2) for security purposes, it will not have access to the system's dynamic libraries. This means that only statically linked versions of binaries should be placed in this directory. The ls program must be copied from /bin/ls and not from /usr/bin/ls.

If you have installed the OCS add-on package, your version of /bin/ls will have been replaced. In that case, use the saved version which can be found in /usr/add-on/OCS/bin/ls.

home_directory/etc
Make this directory owned by the super-user and unwritable by anyone else. The password field in /etc/passwd will not be used and should not contain any real encrypted password. Copies of the files passwd(5), group(5), and netconfig must be present for the ls command to work properly. These files should be mode "444".

home_directory/pub
Make this directory mode "777" and owned by "ftp". Users should then place files which are to be accessible via the anonymous account into this directory.

home_directory/dev
Make this directory owned by the super-user and unwritable by anyone else. Change directories to this directory and do the following:

```
FTP="'grep ^ftp: /etc/passwd | cut -d: -f6'"
MAJORMINOR="'ls -l /dev/tcp | nawk '{ gsub(/,/, ""); print $5, $6}''"
mknod $FTP/dev/tcp c $MAJORMINOR
chmod 666 $FTP/dev/tcp
```

SEE ALSO

ftp(1), chroot(2), getusershell(3), getsockopt(3N), passwd(4), services(4), group(4), syslogd(1M).
RFC 959.

USER CONSIDERATIONS

An "anonymous" FTP account is inherently dangerous and should be avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user ID of the logged-in user, but will revert to super-user status only when binding addresses to sockets.

The possible security holes have been scrutinized extensively, but are possibly incomplete.

/etc/ftpusers contains a list of users who cannot access the system; the format of this file is one user name per line.

NAME

`fumount` - forced unmount of advertised resources

SYNOPSIS

`fumount` [-w *sec*] *resource* [[-w *sec*] *resource*]. . .

DESCRIPTION

`fumount` unadvertises each *resource* and disconnects remote access to the *resource.* The `-w` *sec* causes a delay of *sec* seconds prior to the disconnect from the *resource* specified immediately after the `-w`.

When the forced unmount occurs, an administrative shell script is started on each remote computer that has the resource mounted (`/usr/bin/rfuadmin`). If a grace period of several seconds is specified with `-w`, `rfuadmin` is started with the `fuwarn` option. When the actual forced unmount is ready to occur, `rfuadmin` is started with the `fumount` option. See the `rfuadmin`(1M) manual page for information on the action taken in response to the forced unmount.

This command is restricted to the super-user.

ERRORS

If *resource* (1) does not physically reside on the local machine, (2) is an invalid resource name, (3) is not currently advertised and is not remotely mounted, or (4) the command is not run with super-user privileges, an error message will be sent to standard error.

SEE ALSO

`adv`(1M), `mount`(1M), `rfuadmin`(1M), `rfudaemon`(1M), `rmount`(1M), `unadv`(1M)

NAME

`fusage` - disk access profiler

SYNOPSIS

`fusage` [[*mount_point*] | [*advertised_resource*] | [*block_special_device*] [. . .]]

DESCRIPTION

When used with no options, `fusage` reports block I/O transfers, in kilobytes, to and from all locally mounted file systems and advertised Remote File Sharing resources on a per client basis. The count data are cumulative since the time of the mount. When used with an option, `fusage` reports on the named file system, advertised resource, or block special device.

The report includes one section for each file system and advertised resource and has one entry for each machine that has the directory remotely mounted, ordered by decreasing usage. Sections are ordered by device name; advertised resources that are not complete file systems will immediately follow the sections for the file systems they are in.

SEE ALSO

`adv`(1M), `mount`(1M), `df`(1M), `crash`(1M)

NAME

fuser - identify processes using a file or file structure

SYNOPSIS

/usr/sbin/fuser [-[c|f]ku] *files* | *resources* [[-] [-[c|f]ku] *files* | *resources*] . . .

DESCRIPTION

fuser outputs the process IDs of the processes that are using the *files* or remote *resources* specified as arguments. Each process ID is followed by one of these letter codes, which identify how the process is using the file:

c as its current directory.

r as its root directory, which was set up by the chroot(1M) command.

o as an open file.

t as its text file.

a as its trace file located in the /proc directory.

For block special devices with mounted file systems, processes using any file on that device are listed. For remote resource names, processes using any file associated with that remote resource (Remote File Sharing) are reported. For all other types of files (text files, executables, directories, devices, etc.) only the processes using that file are reported.

The following options may be used with fuser:

-c may be used with files that are mount points for file systems. With that option the report is for use of the mount point and any files within that mounted file system.

-f when this is used, the report is only for the named file, not for files within a mounted file system.

-u the user login name, in parentheses, also follows the process ID.

-k the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see kill(2)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

Any user with permission to read /dev/kmem and /dev/mem can use fuser. Only the super-user can terminate another user's process

EXAMPLES

fuser -ku /dev/dsk/1s?

if typed by a user with appropriate privileges, terminates all processes that are preventing disk drive one from being unmounted, listing the process ID and login name of each as it is killed.

`fuser -u /etc/passwd`
lists process IDs and login names of processes that have the password file open.

`fuser -ku /dev/dsk/1s? -u /etc/passwd`
executes both of the above examples in a single command line.

`fuser -cu /home`
if the `/dev/dsk/c1d0s9` device is mounted on /home, lists process ID's and login names of processes that are using `/dev/dsk/c1d0s9`.

FILES

`/stand/unix`	for system namelist
`/dev/kmem`	for system image
`/dev/mem`	also for system image

NOTE

If an RFS resource from a pre System V Release 4 server is mounted, `fuser` can only report on use of the whole file system, not on individual files within it.

Because `fuser` works with a snapshot of the system image, it may miss processes that begin using a file while `fuser` is running. Also, processes reported as using a file may have stopped using it while `fuser` was running. These factors should discourage the use of the `-k` option.

`fuser` does not report all possible usages of a file (for example, a mapped file).

SEE ALSO

`chroot`(1M) `mount`(1M), `ps`(1) `kill`(2), `signal`(2), `proc(4)`

NAME

fwtmp, wtmpfix - manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [-ic]
/usr/lib/acct/wtmpfix [*files*]

DESCRIPTION

fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in /var/adm/wtmp to formatted ASCII records. The ASCII version is useful when it is necessary to edit bad records.

The argument -ic is used to denote that input is in ASCII form, and output is to be written in binary form.

wtmpfix examines the standard input or named files in utmp.h format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, acctcon will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to /var/adm/wtmp. The first record is the old date denoted by the string "old time" placed in the line field and the flag OLD_TIME placed in the type field of the utmp structure. The second record specifies the new date and is denoted by the string new time placed in the line field and the flag NEW_TIME placed in the type field. wtmpfix uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, wtmpfix will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to INVALID and write a diagnostic to the standard error. In this way, wtmpfix reduces the chance that acctcon will fail when processing connect accounting records.

FILES

/var/adm/wtmp
/usr/include/utmp.h

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), ed(1), runacct(1M), acct(2), acct(4), utmp(4).

NAME

gcore - get core images of running processes

SYNOPSIS

gcore [-o *filename*] *process-id* . . .

DESCRIPTION

gcore creates a core image of each specified process. Such an image may be used with debuggers such as tbx. The name of the core image file for the process whose process ID is *process-id* will be core.*process-id*.

The -o option substitutes *filename* in place of core as the first part of the name of the core image files.

FILES

core.*process-id* core images

SEE ALSO

kill(1), csh(1), tbx(1), ptrace(2).

NAME

gencat - generate a formatted message catalogue

SYNOPSIS

gencat [-m] *catfile msgfile* ...

DESCRIPTION

The gencat utility merges the message text source file(s) msgfile into a formatted message database *catfile*. The database *catfile* will be created if it does not already exist. If *catfile* does exist its messages will be included in the new *catfile*. If set and message numbers collide, the new message-text defined in *msgfile* will replace the old message text currently contained in *catfile*. The message text source file (or set of files) input to gencat can contain either set and message numbers or simply message numbers, in which case the set NL_SETD [see nl_types(5)] is assumed.

The format of a message text source file is defined as follows. Note that the fields of a message text source line are separated by a single ASCII space or tab character. Any other ASCII spaces or tabs are considered as being part of the subsequent field.

$set n comment

Where *n* specifies the set identifier of the following messages until the next $set, $delset or end-of-file appears. *n* must be a number in the range (1-{NL_SETMAX}). Set identifiers within a single source file need not be contiguous. Any string following the set identifier is treated as a comment. If no $set directive is specified in a message text source file, all messages will be located in the default message set NL_SETD.

$delset n comment

Deletes message set *n* from an existing message catalogue. Any string following the set number is treated as a comment.

(Note: if *n* is not a valid set it is ignored.)

$ comment

A line beginning with a dollar symbol $ followed by an ASCII space or tab character is treated as a comment.

m message-text

The *m* denotes the message identifier, which is a number in the range (1-{NL_MSGMAX}). The message-text is stored in the message catalogue with the set identifier specified by the last $set directive, and with message identifier *m*. If the message-text is empty, and an ASCII space or tab field separator is present, an empty string is stored in the message catalogue. If a message source line has a message number, but neither a field separator nor message-text , the existing message with that number (if any) is deleted from the catalogue. Message identifiers need not be contiguous. The length of message-text must be in the range (0-{NL_TEXTMAX}).

$quote c

This line specifies an optional quote character *c*, which can be used to surround message-text so that trailing spaces or null (empty) messages are visible in a message source line. By default, or if an empty $quote directive is supplied, no quoting of message-text will be recognized.

Empty lines in a message text source file are ignored.

Text strings can contain the special characters and escape sequences defined in the following table:

Description	Symbol	Sequence
newline	NL(LF)	\n
horizontal tab	HT	\t
vertical tab	VT	\v
backspace	BS	\b
carriage return	CR	\r
form feed	FF	\f
backslash	\	\\
bit pattern	ddd	\ddd

The escape sequence `\ddd` consists of backslash followed by 1, 2 or 3 octal digits, which are taken to specify the value of the desired character. If the character following a backslash is not one of those specified, the backslash is ignored.

Backslash followed by an ASCII newline character is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

```
1 This line continues \
to the next line
```

which is equivalent to:

```
1 This line continues to the next line
```

NOTES

This version of gencat is built upon the `mkmsgs` utility. The gencat database comprises of two files *catfile*`.m` *which is an* `mkmsgs` format catalogue and the file *catfile* which contains the information required to translate an set and message number into a simple message number which can be used in a call to `gettxt`.

Using `gettxt` constrains the catalogues to be located in a subdirectory under `/usr/lib/locale`. This restriction is lifted by placing only a symbolic link to the catalogue in the directory `/usr/lib/locale/Xopen/LC_MESSAGES` when the catalogue is opened. It is this link that `gettxt` uses when attempting to access the catalogue. The link is removed when the catalogue is closed but occasionally as applications exit abnormally without closing catlogues redundant symbolic links will be left in the directory.

For compatibility with previous version of `gencat` released in a number of specialized internationalization products, the `-m` option is supplied. This option will cause gencat to build a single file *catfile* which is compatible with the format catalogues produced by the earlier versions. The retrieval routines detect the type of catalogue they are using and will act appropriately.

SEE ALSO

`mkmsgs`(1), `catopen`(3C), `catgets`(3C), `catclose`(3C), `gettxt`(3C), `nl_types`(5).

NAME

get - get a version of an SCCS file

SYNOPSIS

get [-a*seq-no.*] [-c*cutoff*] [-i*list*] [-r*SID*] [-w*string*] [-x*list*] [-l[p]] [-b] [-e] [-g] [-k] [-m] [-n] [-p] [-s] [-t] *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 may 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.

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 the FILES section 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 apply independently to each named file.

-r*SID* The SCCS identification 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 that 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 may separate the two-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form:

```
-c"77/2/2 9:22:25".
```

-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.

-x*list* A *list* of deltas to be excluded in the creation of the generated file. See the -i keyletter for the *list* format.

`-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`s 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.) A branch `delta` may always be created from a non-leaf `delta`. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

`-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 IDENTIFICATION KEYWORDS for detailed information on the l-file.

`-p` Causes the text retrieved from the SCCS file to be written on the standard output. No g-file is created. All output that normally goes to the standard output goes to file descriptor 2 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 delta in a given release (for example, `-r1`), or release and level (for example, `-r1.2`).

`-w` *string* Substitute *string* for all occurrences of `%W%` when getting the file. Substitution occurs prior to keyword expansion.

`-a`*seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved. This keyletter is used by the `comb` command; it is not a generally useful keyletter. If both the `-r` and `-a` keyletters are specified, only 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 (that is, 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 four-character string @(#) recognizable by the what command.
%W%	A shorthand notation for constructing what strings for UNIX System program files. %W% = %Z%%M%<tab>%I%
%A%	Another shorthand notation for constructing what strings for non-UNIX System program files: %A% = %Z%%Y% %M% %I%%Z%

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), or "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 (that is, 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 with mode 444.

FILES

g-file	Created by the execution of get.
p-file	[see delta(1)]
q-file	[see delta(1)]
z-file	[see delta(1)]
bdiff	Program to compute differences between the "gotten" file and the g-file.

SEE ALSO

admin(1), bdiff(1), delta(1), help(1), prs(1), what(1).

DIAGNOSTICS

Use help(1) for explanations.

NOTES

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.

NAME

getdev - lists devices based on criteria

SYNOPSIS

getdev [-ae] [*criteria* [. . .]] [*device* [. . .]]

DESCRIPTION

getdev generates a list of devices that match certain criteria. The criteria includes a list of attributes (given in expressions) and a list of devices. If no criteria is given, all devices are included in the list.

Devices must satisfy at least one of the criteria in the list unless the -a option is used. Then, only those devices which match all of the criteria in a list will be included.

Devices which are defined on the command line and which match the criteria are included in the generated list. However, if the -e flag is used, the list becomes a set of devices to be *excluded* from the list.

Criteria Expression Types

There are four possible expression types which the criteria specified in the *criteria* argument may follow:

attribute=*value*	Selects all devices whose attribute *attribute* is defined and is equal to *value*.
attribute!=*value*	Selects all devices whose attribute *attribute* is defined and does not equal *value*.
attribute:*	Selects all devices which have the attribute *attribute* defined.
attribute!:*	Selects all devices which do not have the attribute *attribute* defined.

See the putdev(1M) manual page for a complete listing and description of available attributes.

Options and Arguments

The options and arguments for this command are:

-a	Specifies that a device must match all criteria to be included in the list generated by this command. The flag has no effect if no criteria are defined.
-e	Specifies that the list of devices which follows on the command line should be *excluded* from the list generated by this command. (Without the -e the named devices are *included* in the generated list.) The flag has no effect if no devices are defined.
criteria	Defines criteria that a device must match to be included in the generated list. Should be given in expressions.
device	Defines devices which should be included in the generated list. Can be the pathname of the device or the device alias.

ERRORS

The command will exit with one of the following values:

0 = Successful completion of the task.

1 = Command syntax incorrect, invalid option used, or internal error occurred.

2 = Device table could not be opened for reading.

FILES

`/etc/device.tab`

SEE ALSO

`devattr`(1), `getdgrp`(1), `putdev`(1), `putdgrp`(1), `getdev`(3X).

NAME

getdgrp - lists device groups which contain devices that match criteria

SYNOPSIS

getdgrp [-ael] [*criteria* [. . .]] [*dgroup* [. . .]]

DESCRIPTION

getdgrp generates a list of device groups that contain devices matching the given criteria. The criteria is given in the form of expressions.

criteria can be one expression or a list of expressions which a device must meet for its group to be included in the list generated by getdgrp. If no criteria is given, all device groups are included in the list.

Devices must satisfy at least one of the criteria in the list. However, the -a flag can be used to define that a "logical and" operation should be performed. Then, only those groups containing devices which match all of the criteria in a list will be included.

dgroup defines a set of device groups to be included in the list. Device groups that are defined and which contain devices matching the criteria are included. However, if the -e flag is used, this list defines a set of device groups to be excluded. When the -e option is used and criteria is also defined, the generated list will include device groups containing devices which match the criteria and are not in the command line list.

Criteria Expression Types

There are four possible expressions types:

attribute=*value* — Selects all device groups with a member whose attribute *attribute* is defined and is equal to *value*.

attribute!=*value* — Selects all device groups with a member whose attribute *attribute* is defined and does not equal *value*.

attribute:* — Selects all device groups with a member which has the attribute *attribute* defined.

attribute!:* — Selects all device groups with a member which does not have the attribute *attribute* defined.

See the putdev(1M) manual page for a complete listing and description of available attributes.

Options and Arguments

The options and arguments for this command are:

-a — Specifies that a device must match all criteria before a device group to which it belongs can be included in the list generated by this command. The flag has no effect if no criteria are defined.

-e — Specifies that the list of device groups on the command line should be excluded from the list generated by this command. (Without the -e the named device groups are the only ones which can be included in the generated list.) The flag has no effect if no device groups are defined.

`-l`	Specifies that all device groups (subject to the `-e` option and the *dgroup* list) should be listed even if they contain no valid device members. This option has no affect if *criteria* is specified on the command line.
criteria	Defines criteria that a device must match before a device group to which it belongs can be included in the generated list.
dgroup	Defines device groups which should be included in or excluded from the generated list.

ERRORS

The command will exit with one of the following values:

`0` = successful completion of the task.

`1` = command syntax incorrect, invalid option used, or internal error occurred.

`2` = device table or device group table could not be opened for reading.

FILES

`/etc/device.tab`
`/etc/dgroup.tab`

SEE ALSO

`devattr`(1), `getdev`(1), `putdev`(1), `putdgrp`(1), `getdgrp`(3X)

NAME

`getfrm` - returns the current frameID number

SYNOPSIS

`getfrm`

DESCRIPTION

`getfrm` returns the current frameID number. The frameID number is a number assigned to the frame by FMLI and displayed flush left in the frame's title bar. If a frame is closed its frameID number may be reused when a new frame is opened. `getfrm` takes no arguments.

EXAMPLES

If a menu whose frameID is `3` defines an item to have this `action` descriptor:

```
action=open text stdtext `getfrm`
```

the text frame defined in the definition file `stdtext` would be passed the argument `3` when it is opened.

NOTES

It is not a good idea to use `getfrm` in a backquoted expression coded on a line by itself. Stand-alone backquoted expressions are evaluated before any descriptors are parsed, thus the frame is not yet fully current, and may not have been assigned a frameID number.

NAME

`getid` - program to retrieve the "system" MIB variables from an SNMP entity.

SYNOPSIS

`getid` *entity_addr community string*

DESCRIPTION

`getid` is an SNMP application to retrieve the variables `sysDescr.0`, `sysObjectID.0`, and `sysUpTime.0` from an SNMP entity. The arguments are the entity's address and the community string needed for access to the SNMP entity. The primary purpose of this application is to illustrate the use of the SNMP library routines.

SEE ALSO

`getmany`(1M), `getone`(1M), `getnext`(1M)
RFC 1155, RFC 1156, RFC 1157

NAME

`getitems` - return a list of currently marked menu items

SYNOPSIS

`getitems` [*delimiter_string*]

DESCRIPTION

The `getitems` function returns the value of `lininfo` if defined, else it returns the value of the `name` descriptor, for all currently marked menu items. Each value in the list is delimited by *delimiter_string*. The default value of *delimiter_string* is newline.

EXAMPLE

The `done` descriptor in the following menu definition file executes `getitems` when the user presses ENTER (note that the menu is multiselect):

```
Menu="Example"
multiselect=TRUE
done=`getitems ":" | message`

name="Item 1"
action=`message "You selected item 1"`

name="Item 2"
lininfo="This is item 2"
action=`message "You selected item 2"`

name="Item 3"
action=`message "You selected item 3"`
```

If a user marked all three items in this menu, pressing ENTER would cause the following string to be displayed on the message line:

```
Item 1:This is item 2:Item 3
```

Note that because `lininfo` is defined for the second menu item, its value is displayed instead of the value of the `name` descriptor.

NAME

`getmajor` - print major number(s) of hardware and software drivers

SYNOPSIS

`/usr/sbin/getmajor` [`-m` *master_dir*] *name*

DESCRIPTION

The `getmajor` command prints the major number for the requested driver found in the corresponding `master.d` file. *Name* is the boot hame of the driver. `Getmajor` will convert *name* to lower case before searching for the master file.

`Getmajor` searches the directory `/etc/master.d` for the master file. The [`-m`] option can be used to specify an alternate master directory.

DIAGNOSTICS

If successful, a zero is returned. If *name* or *ID_code* is not found, a blank line is printed and the return code is nonzero.

SEE ALSO

`ed`(1), `prtconf`(1M)

NAME

`getmany` - program to retrieve classes of variables from an SNMP entity

SYNOPSIS

`getmany` *entity_addr community_string variable class name [variable class name] ...*

DESCRIPTION

`getmany` is an SNMP application to retrieve classes of variables from an SNMP entity. The arguments are the entity's address, the community string access to the SNMP entity, and the variable class name(s) is expressed as object identifiers in either dot-notation or as the mib-variables as they appear in the MIB document. `getmany` retrieves the variable class by calling the SNMP entity with the variable class name to get the first variable in the class, and then calling the entity again using the variable name returned in the previous call to retrieve the next variable in the class, utilizing the get-next aspect of the variable retrevial system. For instance, running the following:

```
getmany suzzy public ipRouteDest
```

will traverse the network entity's ipRouteDest variable class (the next node traveling to in the route for the given net-number, which makes up the rest of the variable name) The traversing of the variable space stops when all of the classes being polled return a variable of a class different than what was requested. Note that a network entity's entire variable tree can be traversed with a call of

```
getmany suzzy public iso
```

SEE ALSO

`getone`(1M), `getnext`(1M)
RFC 1155, RFC 1156, RFC 1157

NAME

getnext - program to retrieve variables from an SNMP entity

SYNOPSIS

getnext *entity_addr community string variable name [variable name] ...*

DESCRIPTION

getnext is an SNMP application to retrieve a set of individual variables from an SNMP entity using a GET-NEXT request. The arguments are the entity's address, the community string for access to the SNMP entity, and the variable name(s) expressed as either dot-notation or the variable name as it appears in the MIB document. It should be noted that since the function utilizes the powerful GET-NEXT operator, the variable returned will be the lexicographically greater fully qualified object identifier for what was entered. For example:

```
getnext suzzy public system interfaces
```

would return the variables sysDescr.0 and ifNumber.0.

SEE ALSO

getmany(1M), getone(1M)
RFC 1155, RFC 1156, RFC 1157

NAME

getone - program to retrieve variables from an SNMP entity

SYNOPSIS

getone *entity_addr community string variable name [variable name] ...*

DESCRIPTION

getone is an SNMP application to retrieve a set of individual variables from an SNMP entity using a GET request. The arguments are the entity's address, the community string for access to the SNMP entity, and the fully qualified variable name(s) expressed as either dot-notation or the variable name as it appears in the MIB document. It should be noted that since the function is GET , as opposed to a GET-NEXT , the variable **must** be fully qualified for the request to be successful. For example:

```
getone suzzy public sysDescr.0 ifNumber.0
```

would return the variables sysDescr.0 and ifNumber.0 .

But the call

```
getone suzzy public system
```

would return an error from the entity since it is not a fully qualified SNMP variable.

SEE ALSO

getmany(1M), getnext(1M)
RFC 1155, RFC 1156, RFC 1157

NAME

getopt - parse command options

SYNOPSIS

```
set -- `getopt optstring $*`
```

DESCRIPTION

The getopts command supercedes getopt. For more information, see the NOTES below.

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 special option -- is used to delimit the end of the options. If it is used explicitly, getopt recognizes it; otherwise, getopt generates it; in either case, getopt places it at the end of the options. 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.

EXAMPLE

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 accepts 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
```

SEE ALSO

getopts(1), sh(1), getopt(3C).

DIAGNOSTICS

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NOTES

`getopt` will not be supported in the next major release. For this release a conversion tool has been provided, `getoptcvt`. For more information about `getopts` and `getoptcvt`, see `getopts`(1).

Reset `optind` to 1 when rescanning the options.

`getopt` does not support the part of Rule 8 of the command syntax standard [see `intro`(1)] that permits groups of option-arguments following an option to be separated by white space and quoted. For example,

```
cmd -a -b -o "xxx z yy" file
```

is not handled correctly. To correct this deficiency, use the `getopts` command in place of `getopt`.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring* (referring to the earlier EXAMPLE section, but using the following command line: `cmd -o -a file`), `getopt` always treats `-a` as an option-argument to `-o`; it never recognizes `-a` as an option. For this case, the `for` loop in the example shifts past the *file* argument.

NAME

getopts, getoptcvt - parse command options

SYNOPSIS

getopts *optstring name* [*arg* . . .]

/usr/lib/getoptcvt [-b] *file*

DESCRIPTION

getopts is used by shell procedures to parse positional parameters and to check for valid options. It supports all applicable rules of the command syntax standard (see Rules 3-10, intro(1)). It should be used in place of the getopt command. (See the NOTES section below.)

optstring must contain the option letters the command using getopts will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, getopts places the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable OPTIND. Whenever the shell or a shell procedure is invoked, OPTIND is initialized to 1. (OPTIND is not initialized to 1 when a shell function is called.)

When an option requires an option-argument, getopts places it in the shell variable OPTARG.

If an illegal option is encountered, ? will be placed in *name*.

When the end of options is encountered, getopts exits with a non-zero exit status. The special option -- may be used to delimit the end of the options.

By default, getopts parses the positional parameters. If extra arguments (*arg* . . .) are given on the getopts command line, getopts parses them instead.

/usr/lib/getoptcvt reads the shell script in *file*, converts it to use getopts instead of getopt, and writes the results on the standard output.

-b Make the converted script portable to earlier releases of the UNIX system. /usr/lib/getoptcvt modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke getopts or getopt.

So all new commands will adhere to the command syntax standard described in intro(1), they should use getopts or getopt to parse positional parameters and check for options that are valid for that command (see the NOTES section below).

EXAMPLE

The following fragment of a shell program 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 option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
```

```
done
shift `expr $OPTIND - 1`
```

This code accepts any of the following as equivalent:

```
cmd -a -b -o "xxx z yy" file
cmd -a -b -o "xxx z yy" -- file
cmd -ab -o xxx,z,yy file
cmd -ab -o "xxx z yy" file
cmd -o xxx,z,yy -b -a file
```

SEE ALSO

intro(1), sh(1), getopt(3C).

NOTES

Although the following command syntax rule [see intro(1)] relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the EXAMPLE section above, a and b are options, and the option o requires an option-argument. The following example violates Rule 5: options with option-arguments must not be grouped with other options:

```
cmd -aboxxx file
```

The following example violates Rule 6: there must be white space after an option that takes an option-argument:

```
cmd -ab -oxxx file
```

Changing the value of the shell variable OPTIND or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

getroute - a program to extract the routing information from an SNMP entity

SYNOPSIS

getroute *entity_addr community string*

DESCRIPTION

getroute is an SNMP application that retrieves routing information from an entity by traversing the ipRouteDest, ipRouteIfIndex, ipRouteMetric1, ipRouteNextHop, ipRouteType, and pRouteProto variable classes for each route found. It takes as arguments the address of the SNMP entity and a community string to provide access to that entity.

SEE ALSO

getmany(1M), getone(1M), getnext(1M)
RFC 1155, RFC 1156, RFC 1157

NAME

gettable - get DoD Internet format host table from a host

SYNOPSIS

gettable *host*

DESCRIPTION

gettable is a simple program used to obtain the DoD Internet host table from a hostname server. The indicated *host* is queried for the table. The table, if retrieved, is placed in the file hosts.txt.

gettable operates by opening a TCP connection to the port indicated in the service specification for hostname. A request is then made for all names and the resultant information is placed in the output file.

gettable is best used in conjunction with the htable(1M) program which converts the DoD Internet host table format to that used by the network library lookup routines.

SEE ALSO

htable(1M)

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *HOSTNAME Server*, RFC 953, Network Information Center, SRI International, Menlo Park, Calif., October 1985

NOTES

Should allow requests for only part of the database.

NAME

gettxt - retrieve a text string from a message data base

SYNOPSIS

gettxt *msgfile*:*msgnum* [*dflt_msg*]

DESCRIPTION

gettxt retrieves a text string from a message file in the directory /usr/lib/locale/*locale*/LC_MESSAGES. The directory name *locale* corresponds to the language in which the text strings are written; see setlocale(3C).

msgfile Name of the file from which to retrieve *msgnum*. The name can be up to 14 characters in length, but may not contain either \0 (null) or the characters / (slash) or : (colon).

msgnum Sequence number of the string to retrieve from *msgfile*. The strings in *msgfile* are numbered sequentially from 1 to *n*, where *n* is the number of strings in the file.

dflt_msg Default string to be displayed if gettxt fails to retrieve *msgnum* from *msgfile*. Nongraphic characters must be represented as alphabetic escape sequences.

The text string to be retrieved is in the file *msgfile*, created by the mkmsgs(1) utility and installed under the directory /usr/lib/locale/*locale*/LC_MESSAGES. You control which directory is searched by setting the environment variable LC_MESSAGES. If LC_MESSAGES is not set, the environment variable LANG will be used. If LANG is not set, the files containing the strings are under the directory /usr/lib/locale/C/LC_MESSAGES.

If gettxt fails to retrieve a message in the requested language, it will try to retrieve the same message from /usr/lib/locale/C/LC_MESSAGES/*msgfile*. If this also fails, and if *dflt_msg* is present and non-empty, then it will display the value of *dflt_msg*; if *dflt_msg* is not present or is empty, then it will display the string Message not found!!\n.

EXAMPLE

If the environment variables LANG or LC_MESSAGES have not been set to other than their default values,

```
gettxt UX:10 "hello world\n"
```

will try to retrieve the 10th message from /usr/lib/locale/C/LC_MESSAGES/UX. If the retrieval fails, the message "hello world," followed by a new-line, will be displayed.

FILES

/usr/lib/locale/C/LC_MESSAGES/*	default message files created by mkmsgs(1)
/usr/lib/locale/*locale*/LC_MESSAGES/*	message files for different languages created by mkmsgs(1)

SEE ALSO

exstr(1), mkmsgs(1), srchtxt(1), gettxt(3C), setlocale(3C).

NAME

getty - set terminal type, modes, speed, and line discipline

SYNOPSIS

getty [-h] [-t *timeout*] *line* [*speed* [*type* [*linedisc*]]]

getty -c *file*

DESCRIPTION

getty is included for compatibility with previous releases for the few applications that still call getty directly. getty can only be executed by the superuser, that is, by a process with the user ID root. Initially getty prints the login prompt, waits for the user's login name, and then invokes the login command. getty attempts to adapt the system to the terminal speed by using the options and arguments specified on the command line.

line The name of a TTY line in /dev to which getty is to attach itself. getty uses this string as the name of a file in the /dev directory to open for reading and writing.

-h If the -h flag is not set, a hangup will be forced by setting the speed to zero before setting the speed to the default or specified speed.

-t *timeout*
specifies that getty should exit if the open on the line succeeds and no one types anything in *timeout* seconds.

speed The *speed* argument is a label to a speed and TTY definition in the file /etc/ttydefs. This definition tells getty at what speed to run initially, what the initial TTY settings are, and what speed to try next, should the user indicate, by pressing the BREAK key, that the speed is inappropriate. The default *speed* is 1200 baud.

type and *linedisc*
These options are obsolete and will be ignored.

-c *file* The -c option is no longer supported. Instead use sttydefs -l to list the contents of the /etc/ttydefs file and perform a validity check on the file.

When given no optional arguments, getty specifies the following: The *speed* of the interface is set to 1200 baud, either parity is allowed, new-line characters are converted to carriage return-line feed, and tab expansion is performed on the standard output. getty types the login prompt before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pressing the BREAK key. This will cause getty to attempt the next *speed* in the series. The series that getty tries is determined by what it finds in /etc/ttydefs.

NOTES

Administrators and developers are encouraged to use ttymon(1M) as support for getty may be dropped in the future.

FILES

/etc/ttydefs

SEE ALSO

`sttydefs`(1M), `tty`(7), `ttymon`(1M), `ct`(1C), `login`(1), `ioctl`(2).

NAME

`getvol` - verifies device accessibility

SYNOPSIS

`getvol -n [-l` *label*`]` *device*
`getvol [-f | -F] [-wo] [-l` *label* `| -x` *label* `]` *device*

DESCRIPTION

`getvol` verifies that the specified device is accessible and that a volume of the appropriate medium has been inserted. The command is interactive and displays instructional prompts, describes errors, and shows required label information.

Options and arguments for this command are:

`-n` Runs the command in non-interactive mode. The volume is assumed to be inserted upon command invocation.

`-l` Specifies that the label *label* must exist on the inserted volume (can be overriden by the -o option).

`-f` Formats the volume after insertion, using the format command defined for this device in the device table.

`-F` Formats the volume after insertion and places a file system on the device. Also uses the format command defined for this device in the device table.

`-w` Allows administrator to write a new label on the device. User is prompted to supply the label text. This option is ineffective if the `-n` option is enabled.

`-o` Allows the administrator to override a label check.

`-x` Specifies that the label *label* must exist on the device. This option should be used in place of the `-l` option when the label can only be verified by visual means. Use of the option causes a message to be displayed asking the administrator to visually verify that the label is indeed *label*.

device Names the device which should be verified for accessibility.

ERRORS

The command will exit with one of the following values:

`0` = successful completion of the task.

`1` = command syntax incorrect, invalid option used, or internal error occurred.

`3` = device table could not be opened for reading.

NOTES

This command uses the device table to determine the characteristics of the device when performing the volume label checking.

FILES

`/etc/device.tab`

SEE ALSO

`getvol`(3X)

NAME

graph - draw a graph

SYNOPSIS

graph [*option*] ...

DESCRIPTION

graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the plot(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes, "...", in which case they may be empty or contain blanks and numbers; labels never contain new lines.

The following options are recognized, each as a separate argument.

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by -x).

-b Break (disconnect) the graph after each label in the input.

-c Character string given by next argument is default label for each point.

-g Next argument is grid style, 0 for no grid, 1 for a frame with ticks, 2 for a full grid (default).

-l Next argument is label for graph.

-m Next argument is mode (style) of connecting lines: 0 for disconnected, 1 for connected (default). Some devices give distinguishable line styles for other small integers.

-s Save screen, do not erase before plotting.

-x [l]
If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.

-y [l]
If l is present, y axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) y limits. Third argument, if present, is grid spacing on y axis. Normally these quantities are determined automatically.

-h Next argument is fraction of space for height.

-w Next argument is fraction of space for width.

-r Next argument is fraction of space to move right before plotting.

-u Next argument is fraction of space to move up before plotting.

-t Transpose horizontal and vertical axes. (Option -x now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the -s option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

`plot`(1G), `spline`(1G).

BUGS

`graph` stores all points internally and drops those for which there is no room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

grep - search a file for a pattern

SYNOPSIS

grep [*options*] limited regular expression [*file ...*]

DESCRIPTION

grep searches files for a pattern and prints all lines that contain that pattern. grep uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with ed(1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters $, *, [, ^, |, (,), and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes ´...´.

If no files are specified, grep assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

- -b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
- -c Print only a count of the lines that contain the pattern.
- -i Ignore upper/lower case distinction during comparisons (valid for single byte characters only).
- -h Prevents the name of the file containing the matching line from being appended to that line. Used when searching multiple files.
- -l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
- -n Precede each line by its line number in the file (first line is 1).
- -s Suppress error messages about nonexistent or unreadable files
- -v Print all lines except those that contain the pattern.

INTERNATIONAL FUNCTIONS

grep can process characters from supplementary code sets, as well as ASCII characters. Searches are performed on characters, not individual bytes.

SEE ALSO

ed(1), egrep(1), fgrep(1), sed(1), sh(1).

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).

BUGS

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in /usr/include/stdio.h.
If there is a line with embedded nulls, grep will only match up to the first null; if it matches, it will print the entire line.

NAME

groupadd - add (create) a new group definition on the system

SYNOPSIS

groupadd [-g *gid* [-o]] *group*

DESCRIPTION

The groupadd command creates a new group definition on the system by adding the appropriate entry to the /etc/group file.

The following options are available:

-g *gid* The group ID for the new group. This group ID must be a non-negative decimal integer below MAXUID as defined in the <param.h> header file. By default, a unique group ID is allocated in the valid range. Group IDs from 0-99 are reserved.

-o This option allows the *gid* to be duplicated (non-unique).

group A string of printable characters that specifies the name of the new group. It may not include a colon (:) or newline (\\n).

FILES

/etc/group

SEE ALSO

groupdel(1M), groupmod(1M), logins(1M), useradd(1M), userdel(1M), usermod(1M), users(1)

DIAGNOSTICS

The groupadd command exits with one of the following values:

0 Success.

2 Invalid command syntax; a usage message for the groupadd command is displayed.

3 An invalid argument was provided to an option.

4 *gid* is not unique (when the -o option is not used).

9 *group* is not unique.

10 Cannot update the /etc/group file.

NAME

groupdel - delete a group definition from the system

SYNOPSIS

groupdel *group*

DESCRIPTION

The groupdel command deletes a group definition from the system. It deletes the appropriate entry from the /etc/group file.

The following options are available:

group A string of printable characters that specifies the group to be deleted.

FILES

/etc/group

SEE ALSO

groupadd(1M), groupmod(1M), logins(1M), useradd(1M), userdel(1M), usermod(1M), users(1)

DIAGNOSTICS

The groupdel command exits with one of the following values:

0 Success.

2 Invalid command syntax. A usage message for the groupdel command is displayed.

6 group does not exist.

10 Cannot update the /etc/group file.

NAME

`groupmod` - modify a group definition on the system

SYNOPSIS

`groupmod` [`-g` *gid* [`-o`]] [`-n` *name*] `group`

DESCRIPTION

The `groupmod` command modifies the definition of the specified group by modifying the appropriate entry in the `/etc/group` file.

The following options are available:

-g *gid* The group id for the new group. This group id must be a non-negative decimal integer below MAXUID as defined in <`param.h`>. The group ID defaults to the next available (unique) number above 99. (Group IDs from 0-99 are reserved.)

-o This option allows the *gid* to be duplicated (non-unique).

-n *name*
A string of printable characters that specifies a new name for the group. It may not include a colon (:) or newline (`\\n`).

group The current name of the group to be modified.

FILES

`/etc/group`

SEE ALSO

`groupadd`(1M), `groupdel`(1M), `logins`(1M), `useradd`(1M), `userdel`(1M), `usermod`(1M), `users`(1).

DIAGNOSTICS

The `groupmod` command exits with one of the following values:

`0` Success.

`2` Invalid command syntax. A usage message for the `groupmod` command is displayed.

`3` An invalid argument was provided to an option.

`4` *gid* is not unique (when the `-o` option is not used).

`6` *group* does not exist.

`9` *name* already exists as a group name.

`10` Cannot update the `/etc/group` file.

NAME

`groups` - print group membership of user

SYNOPSIS

`groups` [*user*]

DESCRIPTION

The command `groups` prints on standard output the groups to which you or the optionally specified user belong. Each user belongs to a group specified in `/etc/passwd` and possibly to other groups as specified in `/etc/group`.

SEE ALSO

`setgroups`(2), `group`(4), `passwd`(4)

FILES

`/etc/passwd`
`/etc/group`

NAME

groups - display a user's group memberships

SYNOPSIS

/usr/ucb/groups [*user* . . .]

DESCRIPTION

With no arguments, groups displays the groups to which you belong; else it displays the groups to which the user belongs. Each user belongs to a group specified in the password file /etc/passwd and possibly to other groups as specified in the file /etc/group. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

FILES

/etc/passwd
/etc/group

SEE ALSO

getgroups(2).

NOTES

This command is obsolescent.

NAME

grpck - check group database entries

SYNOPSIS

/usr/ucb/grpck [*filename*]

DESCRIPTION

grpck checks that a file in group(4) does not contain any errors; it checks the /etc/group file by default.

This command differs from /usr/sbin/grpck in its ability to correctly parse YP entries in /etc/passwd.

FILES

/etc/group

SEE ALSO

group(4), passwd(4).

DIAGNOSTICS

Too many/few fields
: An entry in the group file does not have the proper number of fields.

No group name
: The group name field of an entry is empty.

Bad character(s) in group name
: The group name in an entry contains characters other than lower-case letters and digits.

Invalid GID
: The group ID field in an entry is not numeric or is greater than 65535.

Null login name
: A login name in the list of login names in an entry is null.

Login name not found in password file
: A login name in the list of login names in an entry is not in the password file.

NAME

halt - stop the processor

SYNOPSIS

/usr/ucb/halt [-nqy]

DESCRIPTION

halt writes out any information pending to the disks and then stops the processor.

halt normally logs the system shutdown to the system log daemon, syslogd(1M), and places a shutdown record in the login accounting file /var/adm/wtmp. These actions are inhibited if the -n or -q options are present.

The following options are available:

-n Prevent the *sync* before stopping.

-q Quick halt. No graceful shutdown is attempted.

-y Halt the system, even from a dialup terminal.

FILES

/var/adm/wtmp login accounting file

SEE ALSO

reboot(1M), syslogd(1M), shutdown(1M), init(1M).

NOTES

This command is equivalent to init 0.

NAME

hd - display files in hexadecimal format

SYNOPSIS

hd [*-format* [-s *offset*] [-n *count*] [*file*]

DESCRIPTION

The hd command displays the contents of files in hexadecimal octal, decimal and character formats. Control over the specification of ranges of characters is also available. The default behavior is with the following flags set: "-abx -A". This says that addresses (file offsets) and bytes are printed in hexadecimal and that characters are also printed. If no *file* argument is given, the standard input is read.

Options include:

-s *offset* Specify the beginning offset in the file where printing is to begin. If no 'file' argument is given, or if a seek fails because the input is a pipe, 'offset' bytes are read from the input and discarded. Otherwise, a seek error will terminate processing of the current file.

The *offset* may be given in decimal, hexadecimal (preceded by '0x'), or octal (preceded by a '0'). It is optionally followed by one of the following multipliers: w, l, b, or k; for words (2 bytes), long words (4 bytes), blocks (512 bytes), or K bytes (1024 bytes). Note that this is the one case where "b" does not stand for bytes. Since specifying a hexadecimal offset in blocks would result in an ambiguous trailing 'b', any offset and multiplier may be separated by an asterisk (*).

-n *count* Specify the number of bytes to process. The *count* is in the same format as *offset*, above.

Format Flags

Format flags may specify addresses, characters, bytes, words (2 bytes), or longs (4 bytes) to be printed in hexadecimal, decimal, or octal. Two special formats may also be indicated: test or ASCII. Format and base specifiers amy be freely combined and repeated as desired in order to specify different bases (hexadecimal, decimal or octal) for different output formats (addresses, characters, etc.). All format flags appearing in a single argument are applied as appropriate to all other flags in that argument.

acbwlA Output format specifiers for address, characters, bytes, words, longs and ASCII, respectively. Only one base specifier will be used for addresses; the address will appear on the first line of output that begins each new offset in the input.

The character format prints printable characters unchanged, special C escapes as defined in the language, and remaining values in the specified base.

The ASCII format prints all printable characters unchanged, and all others as a period (.). This format appears to the right of the first of other specified output formats. A base specifier has no meaning with the ASCII format. If no other output format (other than addresses) is given, bx is assumed. If no base specifier is given, all of xdo are used.

`xdo` Output base specifiers for hexadecimal, decimal and octal. If no format specifier is given, all of `acbwl` are used.

`t` Print a test file, each line preceded by the address in the file. Normally, lines should be terminated by a `\n` character; but long lines will be broken up. Control characters in the range 0x00 to 0x1f are rpinted as `'^@'` to `'^_'`. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special characters (^,~,\) are preceded by a backslash (\) to escape their special meaning. As special cases, two values are represented numerically as '\177' and '\377'. This flag will override all output format specifiers except addresses.

NAME

`head` - display first few lines of files

SYNOPSIS

`head` [*-n*] [*file* . . .]

DESCRIPTION

`head` copies the first *n* lines of each *file* to the standard output. If no *file* is given, `head` copies lines from the standard input. The default value of *n* is 10 lines.

When more than one file is specified, the start of each file will look like:

==>*file*<==

Thus, a common way to display a set of short files, identifying each one, is:

`head -9999` *file1 file2* . . .

SEE ALSO

`cat`(1), `more`(1), `pg`(1), `tail`(1)

NAME

help - ask for help with message numbers or SCCS commands

SYNOPSIS

help [*args*]

DESCRIPTION

help finds information to explain a message from a command or explain the use of a SCCS 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 within the parentheses following a message) or SCCS command names.

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".

```
help get
```

prints information about the SCCS get command.

FILES

LIBDIR/help	directory containing files of message text.
LIBDIR/help/helploc	file containing locations of help files not in *LIBDIR*/help.
LIBDIR	usually /usr/ccs/lib

NAME

hostid - print the numeric identifier of the current host

SYNOPSIS

/usr/ucb/hostid

DESCRIPTION

The hostid command prints the identifier of the current host in hexadecimal. This numeric value is likely to differ when hostid is run on a different machine.

SEE ALSO

gethostid(2), sysinfo(2).

NAME

hostname - set or print name of current host system

SYNOPSIS

/usr/ucb/hostname [*name-of-host*]

DESCRIPTION

The hostname command prints the name of the current host, as given before the login prompt. The super-user can set the hostname by giving an argument.

SEE ALSO

uname(1).

NAME

htable - convert DoD Internet format host table

SYNOPSIS

htable *filename*

DESCRIPTION

htable converts a host table in the format specified by RFC 952 to the format used by the network library routines. Three files are created as a result of running htable: hosts, networks, and gateways. The hosts file is used by the gethostent(3N) routines in mapping host names to addresses. The networks file is used by the getnetent(3N) routines in mapping network names to numbers. The gateways file is used by the routing daemon in identifying passive Internet gateways; see routed(1M) for an explanation.

If any of the files localhosts, localnetworks, or localgateways are present in the current directory, the file's contents are prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

htable is best used in conjunction with the gettable(1M) program which retrieves the DoD Internet host table from a host.

FILES

localhosts
localnetworks
localgateways

SEE ALSO

gethostent(3N), getnetent(3N), gettable(1M), routed(1M)

Harrenstien, Ken, Mary Stahl, and Elizabeth Feinler, *DoD Internet Host Table Specification*, RFC 952, Network Information Center, SRI International, Menlo Park, Calif., October 1985

NOTES

Does not properly calculate the gateways file.

NAME

icdpatch - patch in-core disk into kernel

SYNOPSIS

icdpatch [-s] [-u unixname] filesystem

DESCRIPTION

icdpatch will determine the size in blocks of the filesystem specified and print the size to its standard output. If the -u flag is given, icdpatch will copy the entire filesystem into the location given unixname starting at the location in unixname specified by the the symbol "icdbase." Unixname must be a COFF or ELF binary that has a data symbol "icdbase." Unixname should be a bootable kernel and icdbase should represent a buffer in data space large enough to hold the filesystem. Filesystem may be a regular file or a special file as long as its contents is an s5 filesystem.

The -u flag also suppresses the printout of the filesystem size. The -s flag gives a more verbose output of the filesystem size.

Icdpatch is intended to be called by cunix(1m) to patch the In-Core Disk (ICD) image into a bootable kernel.

SEE ALSO

cunix(1m), system(4)

NAME

`iconv` - code set conversion utility

SYNOPSIS

`iconv -f` *fromcode* `-t` *tocode* `[-m` *mode*`]` `[`*file*`]`

DESCRIPTION

`iconv` converts the encoding of characters in *file* from one code set to another and writes the results to standard output.

The required arguments, *fromcode* and *tocode*, identify the input and output code sets, respectively. The optional argument *mode* provides a further distinction between multiple code set maps for the same *fromcode* and *tocode*. If no *file* argument is specified on the command line, `iconv` reads the standard input.

`iconv` uses a database with four required fields (*fromcode*, *tocode*, *table*, *file*) and one optional field (*mode*). The order of the database fields is as named above. The database fields are separated by spaces or tabs, and the database rows are separated by newlines. The optional field *mode* does not have to be uniformly included or excluded from the database, that is, it may be included in some rows and not in others.

`iconv` matches the required arguments *fromcode* and *tocode* and the optional argument *mode* to the corresponding fields in the database. If the *mode* argument is not included in the `iconv` command line, `iconv` will match the first row found that contains the correct *fromcode* and *tocode* fields, ignoring any *mode* fields.

The naming conventions in the database are left entirely up to the user. However, absolute pathnames are required for the *file* fields not located in `/usr/lib/kbd`.

The following table lists the supported conversions.

Code Set Conversions Supported

initial code set	*fromcode* symbol	target code set	*tocode* symbol	*modes*	comment
ISO 6937	6937	ISO 8859-1	88591	d	Teletext
ISO 8859-1	88591	ISO 6937	6937	d	Teletext
ISO 646	646	ISO 8859-1	88591	d	US ASCII
ISO 646DE	646DE	ISO 8859-1	88591	d	German
ISO 646DK	646DK	ISO 8859-1	88591	d	Danish
ISO 646GB	646GB	ISO 8859-1	88591	d	English ASCII
ISO 646ES	646ES	ISO 8859-1	88591	d	Spanish
ISO 646FR	646FR	ISO 8859-1	88591	d	French
ISO 646IT	646IT	ISO 8859-1	88591	d	Italian
ISO 646NO	646NO	ISO 8859-1	88591	d	Norwegian
ISO 646SE	646SE	ISO 8859-1	88591	d	Swedish
ISO 646PT	646PT	ISO 8859-1	88591	d	Portuguese
ISO 646YU	646YU	ISO 8859-2	88592	d	Serbo Croatian
ISO 8859-1	88591	ISO 646	646	d b e p	7 bit ASCII

Code Set Conversions Supported

initial code set	*fromcode* symbol	target code set	*tocode* symbol	*modes*	comment
ISO 8859-1	88591	ISO 646DE	646DE	d b e p	German
ISO 8859-1	88591	ISO 646DK	646DK	d b e p	Danish
ISO 8859-1	88591	ISO 646GB	646GB	d b e p	English ASCII
ISO 8859-1	88591	ISO 646ES	646ES	d b e p	Spanish
ISO 8859-1	88591	ISO 646FR	646FR	d b e p	French
ISO 8859-1	88591	ISO 646IT	646IT	d b e p	Italian
ISO 8859-1	88591	ISO 646NO	646NO	d b e p	Norwegian
ISO 8859-1	88591	ISO 646SE	646SE	d b e p	Swedish
ISO 8859-1	88591	ISO 646PT	646PT	d b e p	Portuguese
ISO 8859-2	88592	ISO 646	646	d b e	
ISO 8859-2	88592	ISO 646YU	646YU	d b e	Serbo Croatian
PC 437	PC437	ISO 8859-1	88591	d b e p	United States
ISO 8859-1	88591	PC 437	PC437	d b e p	
PC 850	PC850	ISO 8859-1	88591	d b e p	Multilingual
ISO 8859-1	88591	PC 850	PC850	d b e p	
PC 860	PC860	ISO 8859-1	88591	d b e	Portugal
ISO 8859-1	88591	PC 860	PC860	d b e p	
PC 863	PC863	ISO 8859-1	88591	d b e	Canada - French
ISO 8859-1	88591	PC 863	PC863	d b e p	
PC 865	PC865	ISO 8859-1	88591	d b e	Norway
ISO 8859-1	88591	PC 865	PC865	d b e p	
PC 437	PC437	ISO 646	646	d b e	United States
PC 850	PC850	ISO 646	646	d b e	Multilingual
PC 860	PC860	ISO 646	646	d b e	Portugal
PC 863	PC863	ISO 646	646	d b e	Canada - French
PC 865	PC865	ISO 646	646	d b e	Norway
PC 437	PC437		EBCDIC	d	
	EBCDIC	PC 437	PC437	d	
PC 437	PC437		IBM_EBCDIC	d	
ISO 8859-1	88591		ROMAN8	d	HP LaserJet II
ISO 8859-1	88591		VT220	d b e	
	VT220	ISO 8859-1	88591	d b e	
ISO 8859-3	88593	ISO 646	646	d b e	Latin 3
ISO 8859-4	88594	ISO 646	646	d b e	Latin 4
ISO 8859-5	88595	ISO 646	646	d b e	Latin/Cyrillic
ISO 8859-7	88597	ISO 646	646	d b e	Latin/Greek
ISO 646	646	ISO 8859-1	8859		US ASCII
ISO 646de	646de	ISO 8859-1	8859		German
ISO 646da	646da	ISO 8859-1	8859		Danish
ISO 646en	646en	ISO 8859-1	8859		English ASCII

Code Set Conversions Supported					
initial code set	*fromcode* symbol	target code set	*tocode* symbol	*modes*	comment
ISO 646es	646es	ISO 8859-1	8859		Spanish
ISO 646fr	646fr	ISO 8859-1	8859		French
ISO 646it	646it	ISO 8859-1	8859		Italian
ISO 646sv	646sv	ISO 8859-1	8859		Swedish
ISO 8859-1	8859	ISO 646	646		7 bit ASCII
ISO 8859-1	8859	ISO 646de	646de		German
ISO 8859-1	8859	ISO 646da	646da		Danish
ISO 8859-1	8859	ISO 646en	646en		English ASCII
ISO 8859-1	8859	ISO 646es	646es		Spanish
ISO 8859-1	8859	ISO 646fr	646fr		French
ISO 8859-1	8859	ISO 646it	646it		Italian
ISO 8859-1	8859	ISO 646sv	646sv		Swedish

ISO 646 is equivalent to U.S. 7-bit ASCII.

The optional *modes* (`d`, `b`, `e`, and `p`) have the following meaning:

`d` default — Any character that cannot be represented is mapped to the "ultimate fall back character," which in the tables supplied is the underscore character '_'.

`b` best fit with no expansion
Where possible, characters are mapped to the closest approximation of that character but always without expansion. That is, all the character mappings are one-to-one. This will be important, for example, when using `curses`-based applications where any expansion of a character representation would affect the screen management. (If such code set mappings are performed by the STREAMS module in the tty subsystem, then such mappings will be transparent and the application will have no knowledge that these mappings take place.)

`e` best fit with expansion
Where possible, characters of the source code set are mapped to the closest approximation of that character in the target code set. Where necessary the character in the source code set is expanded to a sequence of characters in the target code set.

`p` printer mode — with overstriking
If there is a non-destructive backspace, as exists on many printers, then characters that are not available can be displayed by overstriking. In this way many accented characters can be displayed.

EXAMPLES

An example of a database for `iconv` is shown below, with the following fields:

fromcode *tocode* *table* *file* *mode*
(the above field names are not included in the database).

88591	646FR	pubtable	88591.646FR.d	d
88591	646FR	88591.646FR.b	pubfile	b
646	646DE	togerman	/mydir/togerman	

Using the above database, the following command converts the contents of file `mail1` from the ISO 8859-1 code set to ISO 646FR using the `d` mode and stores the results in file `mail.local`.

```
iconv -f 88591 -t 646FR -m d mail1 > mail.local
```

The following command will accomplish the same result as above, because the `d` mode conversion from `88591` to `646FR` is the first row found in the database containing the correct code set conversion.

```
iconv -f 88591 -t 646FR mail1 > mail.local
```

FILES

`/usr/lib/kbd/iconv_data`	default database of supported conversions
`/usr/lib/kbd/*.[tbdep]`	conversion tables

NOTES

The table titled "Code Set Conversions Supported" assumes that the full BOS is available. If the European Language Support package of the BOS is not installed, only those conversions shown in the table with a blank *mode* field are available.

SEE ALSO

`iconv`(5)

DIAGNOSTICS

`iconv` returns 0 upon successful completion, 1 otherwise.

NAME

iconv - code set conversion tables

DESCRIPTION

The following code set conversions are supported:

Code Set Conversions Supported				
Code	Symbol	Target Code	Symbol	comment
ISO 646	646	ISO 8859-1	8859	US Ascii
ISO 646de	646de	ISO 8859-1	8859	German
ISO 646da	646da	ISO 8859-1	8859	Danish
ISO 646en	646en	ISO 8859-1	8859	English Ascii
ISO 646es	646es	ISO 8859-1	8859	Spanish
ISO 646fr	646fr	ISO 8859-1	8859	French
ISO 646it	646it	ISO 8859-1	8859	Italian
ISO 646sv	646sv	ISO 8859-1	8859	Swedish
ISO 8859-1	8859	ISO 646	646	7 bit Ascii
ISO 8859-1	8859	ISO 646de	646de	German
ISO 8859-1	8859	ISO 646da	646da	Danish
ISO 8859-1	8859	ISO 646en	646en	English Ascii
ISO 8859-1	8859	ISO 646es	646es	Spanish
ISO 8859-1	8859	ISO 646fr	646fr	French
ISO 8859-1	8859	ISO 646it	646it	Italian
ISO 8859-1	8859	ISO 646sv	646sv	Swedish

The conversions are performed according to the tables following. All values in the tables are given in octal.

ISO 646 (US ASCII) to ISO 8859-1

For the conversion of ISO 646 to ISO 8859-1 all characters in ISO 646 can be mapped unchanged to ISO 8859-1

ISO 646de (GERMAN) to ISO 8859-1

For the conversion of ISO 646de to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646de	ISO 8859-1
100	247
133	304
134	326
135	334
173	344
174	366
175	374
176	337

ISO 646da (DANISH) to ISO 8859-1

For the conversion of ISO 646da to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646da	ISO 8859-1
133	306
134	330
135	305
173	346
174	370
175	345

ISO 646en (ENGLISH ASCII) to ISO 8859-1

For the conversion of ISO 646en to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646en	ISO 8859-1
043	243

ISO 646fr (FRENCH) to ISO 8859-1

For the conversion of ISO 646fr to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646fr	ISO 8859-1
043	243
100	340
133	260
134	347
135	247
173	351
174	371
175	350
176	250

ISO 646it (ITALIAN) to ISO 8859-1

For the conversion of ISO 646it to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646it	ISO 8859-1
043	243
100	247
133	260
134	347
135	351
140	371
173	340
174	362
175	350
176	354

ISO 646es (SPANISH) to ISO 8859-1

For the conversion of ISO 646es to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646es	ISO 8859-1
100	247
133	241
134	321
135	277
173	260
174	361
175	347

ISO 646sv (SWEDISH) to ISO 8859-1

For the conversion of ISO 646sv to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646sv	ISO 8859-1
100	311
133	304
134	326
135	305
136	334
140	351
173	344
174	366

175	345
176	374

ISO 8859-1 to ISO 646 (ASCII)

For the conversion of ISO 8859-1 to ISO 646 all characters not in the following table are mapped unchanged.

Converted to Underscore '_' (137)
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 371 372 373 374 375 376 377

ISO 8859-1 to ISO 646de (GERMAN)

For the conversion of ISO 8859-1 to ISO 646de all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646de
247	100
304	133
326	134
334	135
337	176
344	173
366	174
374	175

Converted to Underscore '_' (137)
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 327
330 331 332 333 335 336 337
340 341 342 343 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 367
370 371 372 373 375 376 377

ISO 8859-1 to ISO 646da (DANISH)

For the conversion of ISO 8859-1 to ISO 646da all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646da
305	135
306	133
330	134
345	175
346	173
370	174

Converted to Underscore '_' (137)
133 134 135 173 174 175
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
331 332 333 334 335 336 337
340 341 342 343 344 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
371 372 373 374 376 377

ISO 8859-1 to ISO 646en (ENGLISH ASCII)

For the conversion of ISO 8859-1 to ISO 646en all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646en
243	043

Converted to Underscore '_' (137)
043
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346 347
350 351 352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 371 372 373 374 375 376 377

ISO 8859-1 to ISO 646fr (FRENCH)

For the conversion of ISO 8859-1 to ISO 646fr all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646fr
243	043
247	135
250	176
260	133
340	100
347	134
350	175
351	173
371	174

Converted to Underscore '_' (137)
043
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 244 245 246
251 252 253 254 255 256 257
261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
341 342 343 344 345 346
352 353 354 355 356 357
360 361 362 363 364 365 366 367
370 372 373 374 375 376 377

ISO 8859-1 to ISO 646it (ITALIAN)

For the conversion of ISO 8859-1 to ISO 646it all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646it
243	043
247	100
260	133
340	173
347	134
350	175
351	135
354	176
362	174
371	140

Converted to Underscore '_' (137)
043
100 133 134 135 173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 244 245 246
250 251 252 253 254 255 256 257
261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 321 322 323 324 325 326 327
330 331 332 333 334 335 336 337
341 342 343 344 345 346
352 353 354 355 356 357
360 361 363 364 365 366 367
370 372 373 374 375 376 377

ISO 8859-1 to ISO 646es (SPANISH)

For the conversion of ISO 8859-1 to ISO 646es all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646es
241	133
247	100
260	173
277	135
321	134
347	175
361	174

Converted to Underscore '_' (137)
100 133 134 135 173 174 175
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 242 243 244 245 246
250 251 252 253 254 255 256 257
261 262 263 264 265 266 267
270 271 272 273 274 275 276
300 301 302 303 304 305 306 307
310 311 312 313 314 315 316 317
320 322 323 324 325 326 327
330 331 332 333 334 335 336 337
340 341 342 343 344 345 346
350 351 352 353 354 355 356 357
360 362 363 364 365 366 367
370 371 372 373 374 375 376 377

ISO 8859-1 to ISO 646sv (SWEDISH)

For the conversion of ISO 8859-1 to ISO 646sv all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646sv
304	133
305	135
311	100
326	134
334	136
344	173
345	175
351	140
366	174
374	176

Converted to Underscore '_' (137)
100 133 134 135 136 140
173 174 175 176
200 201 202 203 204 205 206 207
210 211 212 213 214 215 216 217
220 221 222 223 224 225 226 227
230 231 232 233 234 235 236 237
240 241 242 243 244 245 246 247
250 251 252 253 254 255 256 257
260 261 262 263 264 265 266 267
270 271 272 273 274 275 276 277
300 301 302 303 306 307
310 312 313 314 315 316 317
320 321 322 323 324 325 327
330 331 332 333 335 336 337
340 341 342 343 346 347
350 352 353 354 355 356 357
360 361 362 363 364 365 367
370 371 372 373 375 376 377

FILES

`/usr/lib/kbd/iconv_data`	lists the conversions supported
`/usr/lib/kbd/*.t`	conversion tables

SEE ALSO

`iconv`(1).

NAME

id - print the user name and ID, and group name and ID

SYNOPSIS

id [-a]

DESCRIPTION

id displays the calling process's ID and name. It also displays the group ID and name. If the real effective IDs do not match, both are printed.

The -a option reports all the groups to which the invoking process belongs. ID, and your username. If your real and effective IDs do not match, both are printed.

The -a option reports all the groups to which the invoking user belongs.

SEE ALSO

getuid(2).

NAME

idload - Remote File Sharing user and group mapping

SYNOPSIS

idload [-n] [-g *g_rules*] [-u *u_rules*] [directory]
idload -k

DESCRIPTION

idload is used on Remote File Sharing server machines to build translation tables for user and group ids. It takes your /etc/passwd and /etc/group files and produces translation tables for user and group ids from remote machines, according to the rules set down in the *u_rules* and *g_rules* files. If you are mapping by user and group name, you will need copies of remote /etc/passwd and /etc/group files. If no rules files are specified, remote user and group ids are mapped to MAXUID+1 (this is an id number that is one higher than the highest number you could assign on your system.)

By default, the remote password and group files are assumed to reside in /etc/rfs/auth.info/*domain*/*nodename*/[passwd | group]. The directory argument indicates that some directory structure other than /etc/rfs/auth.info contains the *domain*/*nodename* passwd and group files. (*nodename* is the name of the computer the files are from and *domain* is the domain that computer is a member of.)

You must run idload to put the mapping into place. Global mapping will take effect immediately for machines that have one of your resources currently mounted. Mapping for other specific machines will take effect when each machine mounts one of your resources.

-n This is used to do a trial run of the id mapping. No translation table will be produced, however, a display of the mapping is output to the terminal (*stdout*).

-k This is used to print the idmapping that is currently in use. (Specific mapping for remote machines will not be shown until that machine mounts one of your resources.)

-u *u_rules* The *u_rules* file contains the rules for user id translation. The default rules file is /etc/rfs/auth.info/uid.rules.

-g *g_rules* The *g_rules* file contains the rules for group id translation. The default rules file is /etc/rfs/auth.info/gid.rules.

This command is restricted to the super-user.

Rules

The rules files have two types of sections (both optional): global and host. There can be only one global section, though there can be one host section for each computer you want to map.

The global section describes the default conditions for translation for any machines that are not explicitly referenced in a host section. If the global section is missing, the default action is to map all remote user and group ids from

undefined computers to MAXUID+1. The syntax of the first line of the global section is:

```
global
```

A host section is used for each machine or group of machines that you want to map differently from the global definitions. The syntax of the first line of each host section is:

```
host name . . .
```

where *name* is replaced by the full name of a computer (*domain.nodename*).

The format of a rules file is described below. (All lines are optional, but must appear in the order shown.)

```
global
default local | transparent
exclude remote_id-remote_id | remote_id
map remote_id:local

host domain.nodename [domain.nodename. . .]
default local | transparent
exclude remote_id-remote_id | remote_id | remote_name
map remote:local | remote | all
```

Each of these instruction types is described below.

The line

```
default local | transparent
```

defines the mode of mapping for remote users that are not specifically mapped in instructions in other lines. transparent means that each remote user and group id will have the same numeric value locally unless it appears in the exclude instruction. *local* can be replaced by a local user name or id to map all users into a particular local name or id number. If the default line is omitted, all users that are not specifically mapped are mapped into a "special guest" login id.

The line

```
exclude remote_id-remote_id | remote_id | remote_name
```

defines remote ids that will be excluded from the default mapping. The exclude instruction must precede any map instructions in a block. You can use a range of id numbers, a single id number, or a single name. (*remote_name* cannot be used in a global block.)

The line

```
map remote:local | remote | all
```

defines the local ids and names that remote ids and names will be mapped into. *remote* is either a remote id number or remote name; *local* is either a local id number or local name. Placing a colon between a *remote* and a *local* will give the value on the left the permissions of the value on the right. A single *remote* name or id will assign the user or group permissions of the same local name or id. all is a predefined alias for the set of all user and group ids found in the local /etc/passwd and /etc/group files. (You cannot map by remote name in

`global` blocks.)

Note: `idload` will always output warning messages for `map all`, since password files always contain multiple administrative user names with the same id number. The first mapping attempt on the id number will succeed, each subsequent attempts will produce a warning.

Remote File Sharing doesn't need to be running to use `idload`.

EXIT STATUS

On successful completion, `idload` will produce one or more translation tables and return a successful exit status. If `idload` fails, the command will return an exit status of zero and not produce a translation table.

ERRORS

If (1) either rules file cannot be found or opened, (2) there are syntax errors in the rules file, (3) there are semantic errors in the rules file, (4) `host` password or group information could not be found, or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Partial failures will cause a warning message to appear, though the process will continue.

FILES

`/etc/passwd`
`/etc/group`
`/etc/rfs/auth.info/`*domain*`/`*nodename*`/`[`user` | `group`]`
`/etc/rfs/auth.info/uid.rules`
`/etc/rfs/auth.info/gid.rules`

SEE ALSO

`mount`(1M).

NAME

ifconfig - configure network interface parameters

SYNOPSIS

ifconfig *interface* [*address_family*] [*address* [*dest_address*]] [*parameters*]
[netmask *mask*] [broadcast *address*] [metric *n*]

ifconfig *interface* [*protocol_family*]

DESCRIPTION

The ifconfig command is used to assign an address to a network interface and/or to configure network interface parameters. The ifconfig command also must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or some other operating parameters. When used without options, ifconfig displays the current configuration for a network interface. If a protocol family is specified, ifconfig will report only the details specific to that protocol family.

The *interface* parameter is a string of the form "name_unit", for example, emd1. The interface name -a is reserved and will cause the remainder of the arguments to be applied to each address of each interface in a sequential manner.

Since an interface may receive transmissions using different protocols - each of which may require separate naming schemes - the parameters and addresses need to be interpreted according to the rules of a specified *address_family* parameter (which may change the interpretation of the remaining command line parameters). Currently, only the Internet *address_family* is supported: ether and inet.

If no *address_family* value is specified, inet will be assumed.

For the DARPA Internet family (inet), the *address* is either a host name present in the host name data base [see hosts(4)], or a DARPA Internet address expressed in the Internet standard "dot notation". Typically, an Internet address specified in dot notation will consist of your system's network number and the machine's unique host number. For example, a typical Internet address could be 192.9.200.44 where 192.9.200 is the "network number" and 44 is the machine's "host number".

For the ether address family, the address is an Ethernet address represented as

$x:x:x:x:x:x$ where x is a hexadecimal number between 0 and ff.

If the *dest_address* parameter is supplied in addition to the address parameter, this specifies the address of the correspondent on the other end of a point-to-point link.

OPTIONS

The following are valid values for *parameters*:

up — Mark an interface up. This may be used to enable an interface after an ifconfig down. It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

down — Mark an interface down. When an interface is marked down, the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well, but this action will not automatically disable routes using this interface.

`trailers`
(`inet` only) Enable the use of a `trailer` link-level encapsulation when sending (the default value). If a network interface supports the `trailers` option (i.e., trailer encapsulation), the system will - when possible - encapsulate outgoing messages in a manner which will minimize the number of memory-to-memory copy operations performed by the receiver.

On networks which support the Address Resolution Protocol (ARP) - see `arp`(7); currently, `arp` applies only to the 10 Mb/s Ethernet - this flag indicates that the system should request that other systems should use trailer encapsulation when sending to this host. Similarly, the trailer encapsulation technique will be used when sending to those other hosts which have made such requests.

`-trailers`
Disable the use of the `trailer` link-level encapsulation technique.

`arp` Enable the use of the Address Resolution Protocol (ARP) in mapping between network-level addresses and link-level addresses (the default value). Currently, this is implemented for mapping between ARPA Internet addresses and 10Mb/s Ethernet addresses.

`-arp` Disable the use of the Address Resolution Protocol.

`metric` *n*
Set the routing metric of the interface to *n*; the default value is 0. The routing metric is used by the routing protocol [`routed`(1M)]. Higher metrics have the effect of making a route less favorable: that is, the "metric" values are counted as additional hops to the destination network or host.

`netmask` *mask*
(`inet` only) Specify how much of the "address space" to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading `0x`, with a "dot-notation" Internet address, or with a pseudo-network name listed in the network table `networks` [see `networks`(4)]. The mask will contain `1` 's for the bit positions in the 32-bit "address space" to be used for the network and subnet portions, and `0` 's for the host portion. The mask should contain at least the standard network portion; the subnet field should be adjacent to the network portion. If a `+` (plus sign) is given for the `netmask` value, then the network number will looked up in the `/etc/netmasks` file.

`broadcast` *address*
(`inet` only) Specify the address for representing broadcasts to the network. The default broadcast address is the address with a host part of all `1` 's. A `+` (plus sign) given for the broadcast value causes the broadcast address to be reset to a default value appropriate for the (possibly new) Internet `address` and `netmask` parameters. Note that the arguments of `ifconfig` are interpreted left to right: therefore the following two command lines may yield different sets of values to be assigned to the broadcast addresses for this interface:

```
ifconfig -a netmask + broadcast +
```

and

```
ifconfig -a broadcast + netmask +
```

onepacket *packet_size threshold*
: Enable the "one-packet" mode of operation which is used for any interface which cannot handle "back-to-back" packets. The keyword onepacket must be followed by two numeric parameters which specify the *packet_size* and *threshold* values, respectively. If "small packet detection" is not desired, these two parameters should be set to zero. [See tcp(7) for an explanation of the "one-packet" mode of operation.]

-onepacket
: Disable the "one-packet" mode of operation.

EXAMPLES

If your workstation is not attached to an Ethernet, the emd1 interface should be marked down as follows:

```
ifconfig emd1 down
```

To print out the addressing information for each interface, use

```
ifconfig -a
```

To reset each interface's broadcast address after the netmasks have been set correctly, use

```
ifconfig -a broadcast +
```

FILES

/dev/nit

/etc/netmasks

/etc/slattach calls ifconfig to start the serial lines

SEE ALSO

arp(1M), netstat(1M), routed(1M), tcp(1M), arp(4), hosts(4), netmasks(4), networks(4), strcf(4), arp(7), tcp(7).

DIAGNOSTICS

There are diagnostic messages to indicate that the specified interface does not exist, that the requested address is unknown, or that the user is not privileged and tried to alter an interface's configuration.

NOTES

Only the super-user may modify the configuration of a network interface.

The super-user is also the only one who may use the ether *address_family*.

The trailers feature is machine-dependent and therefore its use is not recommended.

The arp option is not applicable in the STREAMS environment. The use of arp for an interface is specified in etc/strcf. The arp driver will be opened when the STREAMS stack is built.

NAME

igf - software management package-generation facility

SYNOPSIS

igf [-vb] [-f *prefix_cXdYsuffix*] { *table* | - }

DESCRIPTION

igf is a general-purpose utility for creating software installation tapes. It provides a flexible mechanism to combine identification information, multiple archives of software, and scripts needed to manipulate software packages (usually installation and removal scripts) on a single tape. Optionally, the tape can be made bootable to run standalone.

Information about what to write to the tape is read from *table* or standard input when the argument is a dash (-). This information includes the package description, name, version number, and entries that describe the number and contents of the archives. The archives are not limited to a specific format and may even be several different formats on a single tape.

If the -b option is used to create a bootable tape, *table* also contains information about a bootloader and other files necessary to run in a standalone environment.

The -f option is used to specify a tape device other than the default (/dev/rmt/ctape1n). Since the tape is created with multiple records, *prefix_cXdYsuffix* must be a no-rewind tape device (see WARNINGS). Device specific special files take the form *prefix_cXdYsuffix*, where *prefix* uniquely defines the type of device, *X* specifies the controller number of the stated device type, *Y* specifies the logical device number for the device attached to the stated controller, and *suffix* specifies device dependent information.

The -v option prints messages showing the progress and contents of the tape being made.

table is made up of a number of single- and/or multiple-line entries separated by any number of blank lines. A line is defined as all characters leading up to a newline character; line continuation is not recognized. Lines within a multiple line entry must not be separated by blank lines. A pound sign (#) in the first column of a line indicates a comment line, and the line is discarded. Each line of each entry consists of a keyword immediately followed by an equal sign (=) and the text for that line.

table contains several types of entries. Some of them are optional, while some are used only when creating a bootable tape. In the following descriptions, lines are referred to by their keyword.

DESC — Optional description of the software package on the tape. This description is limited to 88 characters.

VER — Optional package version number. This is a string of 12 characters and need not be a number. All printable characters, including spaces, are acceptable.

PACK — Optional package name. The name is usually the one-word name of the package, limited to 14 characters.

BL Pathlist to the tape bootloader. It must be a valid a.out file that is readable by the current process (only with -b option). Typically, this is /usr/lib/tapeboot.

OS Pathlist to the standalone operating system or boot file. It must be a valid a.out file and readable by the current process (only with -b option).

RAM Ramdisk image. This entry must be a pathlist to a valid file system image (only with -b option).

FILES Definition of an archive's contents. This entry begins a multiple-line entry. It has two arguments separated by white space: source and destination directory. igf changes to the source directory before the archive command is executed. This is useful for archives with relative pathnames. The destination directory is stored on the tape and used by the tape-extraction facility, ixf(1M). The other possible lines in this entry are A_DESC, ARC, O_OPT and I_OPT. The FILES line must be the first line in the entry and is the only required line.

A_DESC Archive description. This entry is a short description of the contents of this archive, limited to 25 characters. This line is always optional. If included, it is stored in the tape directory entry for this archive.

ARC Name of the archive utility. This is the archive utility that is used to write the archive specified by the current FILES entry. The name is not a full pathlist. The environment variable PATH is parsed to find the instance of the utility. The first match is used, as it is in the shell. The archiver name is also stored in the tape directory for extracting of this archive.

O_OPT Archiver output options. These options are used by the archiver to create the archive. If the string $DEVICE is in this entry, it is substituted with the current tape device specified by the -f option or by /dev/rmt/ctape1n when -f was not specified. This allows the use of different tape device nodes without having to change *table*.

I_OPT Archiver input options. These options are used by the archiver when files are being extracted from the tape. They are stored in the tape directory. If the string $DEVICE is in this entry, it is replaced with the current tape device specified on the command line of ixf(1M) when the tape is being read or extracted. This allows different device names to be used on the source and destination systems without changing the contents of the tape directory which would require remaking the tape.

SCRIPTS Scripts archive. This entry is similar to the FILES entry except it has only one argument, a source directory. SCRIPTS is a special archive that is used to contain scripts and all files necessary to manipulate a software package (used by sysadm softwaremgmt). A SCRIPTS entry is also a multiple line entry which can contain all the lines associated with the FILES entry (A_DESC, ARC, O_OPT, I_OPT).

All other lines in FILES1 and SCRIPTS entries are optional. If any one of the ARC, O_OPT or I_OPT lines are specified, all three lines are required to prevent a mismatch of archiver options and arguments with the archive utility. If all three lines are not in the entry, they take on the following values: ARC defaults to cpio,

O_OPT defaults to -ocB > $DEVICE, and I_OPT defaults to -icBdu < $DEVICE. Filenames in and below the source directory are supplied to cpio(1) by find(1).

The tape generated by igf is made up of multiple records. The number of records depends on whether or not the tape is bootable and how many archives are on it.

The first tape record, called volume id block, is a header. It contains information about the tape for the software package and for the tape-based bootloader. There is no line in *table* that creates this record but some information from the table is stored here.

If the tape is bootable, the next record is the bootloader. It is stored on the tape as an image of how it is run, not as an a.out file. The tape bootloader designed to work with igf is /usr/lib/tapeboot. This record is created from the BL line in *table*.

The next record is a tape directory. It contains the following information about all records after the tape directory (where applicable): record number starting at zero for the volume id block, archive number starting at 1 for the first archive, type of record, name of record or archive utility, description, destination directory, and input options. There is no line in *table* that causes this record to be created.

On a bootable tape, the next record is an operating system or stand-alone kernel. There may be as many of these records on the tape as needed. One record is created for each OS entry in *table*. They are stored on the tape in a.out format. The bootloader, 350ipl, is able to select any one of these files to boot from.

After the operating system(s), there may be a ramdisk image record on a bootable tape. This record is created from the RAM line in *table*. It is an image of a file system from disk.

Following this record are all of the archives. Archive records are created from FILES and SCRIPTS entries in *table*, one record per entry. If there is a SCRIPTS entry, it is always the first archive; otherwise, the archives are in the order in which they appear in *table*. These archives have no special headers and their format is determined by the archiving utility used.

EXAMPLES

Following is an example of a table used to create a bootable installation tape for an M88000 platform.

-- Example Table --

```
#IGF Table for creating m88k distribution tapes

DESC=System V/88 Base Operating System

PACK=BOS

VER=FH40.10

# Tape bootloader
BL=/root/stand/m88k/boots/350ipl

# kernel used for installation
OS=/root/stand/tapeboot

RAM=/dev/mbfs

#Archive entry for the bill-of-materials files
FILES=/root                /mnt

ARC=cpio
O_OPT=-ocB > $DEVICE </usr/src/build/tape/releaselist
I_OPT=-icBdu < $DEVICE

#Archive entry for the files in root and usr file systems
FILES=/usr/src/build/tape       /mnt
ARC=cpio
O_OPT=-ocB > $DEVICE <releasefile
I_OPT=-icBdu < $DEVICE
```

In this example, `tapeboot` is the operating system kernel that is booted by the tape-based bootloader `1350ipl`. The file system that is mounted as a memory-based file system by the bootloader is specified by the block-special device `/dev/dsk/m328_c0d0s0`.

There are two `cpio`(1) archive entries. The first archive is for the basic operating system files; the second is for the bill-of-materials file.

Since `cpio`(1) gets its file list from the standard input, standard input is redirected from the file `/usr/src/build/tape/releaselist`, which contains a list of path-names of basic operating systems files for the first archive and `/usr/src/build/tape/releaselist`.

The `$DEVICE` string is used so the tape device node name may be changed for the second archive. Notice that the output is being redirected to the tape.

SEE ALSO

`ixf`(1M), `sh`(1)

NOTES

Although 9-track drives have the capabilities needed by `igf`, only the MVME328 streaming tape drives are supported.

If *prefix*_c*X*d*Y*s*uffix* is not a no-rewind device, the program execution proceeds normally without a single error message, but the tape contains only one record, usually the last archive.

If the `-f` option is used, the *prefix*_c*X*d*Y*s*uffix* name must correspond to a no-rewind device. The name for the rewind device is derived by truncating the last character from the no-rewind device's name (for example, `/dev/rmt/ctape1n` for a no-rewind device and `/dev/rmt/ctape1` for the rewind device).

The option to have the *table* information read from standard input does not work, although it may appear to.

NAME

in.timed, timed - time server daemon

SYNOPSIS

/usr/sbin/in.timed [-t] [-M] [-n *network*] [-i *network*]

DESCRIPTION

The in.timed command is the time daemon server which supports the DARPA Time Server Protocol. Normally, in.timed will be invoked at boot time from a startup script (which is a link to /etc/init.d/timed) located in /etc/rc2.d. in.timed will synchronize the host's time with the time of other machines in a local area network running in.timed. These time servers will slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time. The average network time is computed from measurements of clock differences using the ICMP timestamp request message.

The service provided by in.timed is based on a master-slave scheme. When in.timed is started on a machine, it asks the master for the network time and sets the host's clock to that time. After that, it accepts synchronization messages sent periodically by the master and calls adjtime(2) to perform the needed corrections on the host's clock.

It also communicates with date(1) in order to set the date globally, as well as with timedc(1M), an in.timed control program. If the machine running the master crashes, the slaves will elect a new master from among the slaves running with the -M flag. An in.timed running without the -M flag will remain a slave. The -t flag enables in.timed to trace the messages it receives in the file /var/log/timed.log. Tracing can be turned on or off by the program timedc(1M).

Normally, in.timed checks for a master time server on each network to which it is connected, except as modified by the options described below. It will request synchronization service from the first master server located. If permitted by the -M flag, it will provide synchronization service on any attached networks on which no current master server was detected. Such a server propagates the time computed by the top-level master.

The -n flag, followed by the name of a network to which the host is connected [see networks(4)], will override the default choice of the network addresses made by the program. Whenever the -n flag appears, that network name will be added to a list of valid networks. All other networks will be ignored by the time daemon.

The -i flag, followed by the name of a network to which the host is connected [see networks(4)], will override the default choice of the network addresses made by the program. Whenever the -i flag appears, that network name will be added to a list of networks to ignore. All other networks will be used by the time daemon. **NOTE:** The -n and -i flags will be meaningless if used together.

Network interfaces specified in /etc/if.ignore [see if.ignore(4)] will also be ignored by in.timed.

FILES

/etc/timed.pid	
/etc/if.ignore	
/var/adm/timed.masterlog	log file for master in.timed
/var/log/timed.log	System V tracing file for in.timed

/usr/adm/timed.log BSD tracing file for in.timed

SEE ALSO

date(1), timedc(1M), adjtime(2), gettimeofday(3), if.ignore(4), timednet.conf(4), icmp(7)

NAME

`incfile` - create, restore an incremental filesystem archive

SYNOPSIS

`incfile -B` [`-dilmortvxAENSV`] *bkjobid ofsname ofsdev ofslab descript*

`incfile -T` *bkjobid tocfname descript*

`incfile -RC` [`-dilmortvxAENSV`] *ofsname ofsdev refsname redev rsjobid descript*

`incfile -RF` [`-dilmortvxAENSV`] *ofsname ofsdev descript rsjobid:uid:date:type:name* [:[`rename`]:[`inode`]] ...

DESCRIPTION

`incfile` is invoked as a child process by other shell commands. The command name, `incfile`, is read either from the `bkhist.tab` file or the `bkreg -m` command and option. The `-B`, `-T`, `-R`, `-F`, and `-C` options are passed to `incfile` by the shell commands `backup`, `restore`, and *urestore(1)* described below. The minus options are passed from the `bkhist.tab` file or the `bkreg -p` command and option. The arguments are sent to `incfile` from various locations in the backup service.

`incfile -B` is invoked as a child process by the `bkdaemon` command to perform an incremental backup of the filesystem *ofsname* (the originating filesystem). All files in *ofsname* that have been modified or have had an inode change since the last full backup are archived. The resulting backup is created in `cpio` file format. The backup is recorded in the backup history log, `/etc/bkup/bkhist.tab`.

bkjobid the job id assigned by `backup`. The method uses the *bkjobid* when it creates history log and table-of-contents entries.

ofsname the name of the filesystem that is to be backed up.

ofsdev the name of the UNIX block special device on which the filesystem resides.

ofslab the volume name on the filesystem [see `labelit`(1M)].

descript is a description for a destination device in the form:

dgroup:dname:dchar:dlabels

dgroup specifies a device group [see `devgroup.tab`(4)].
dname specifies a particular device name [see `device.tab`(4)].
dchars specifies characteristics associated with the device. If specified, *dchar* overrides the defaults for the specified device and group. [See `device.tab`(4) for a further description of device characteristics].
dlabels specifies the volume names for the media to be used for reading or writing the archive.

`incfile -T` is invoked as a child process by the `backup` to archive a table-of-contents on the volumes described by *descript.*

tocfname the name of the file containing the table-of-contents.

`incfile -RC` and `incfile -RF` are invoked as child processes by the `rsoper` command to extract files from an incremental filesystem archive created by `incfile -B`. The filesystem archive is assumed to be in `cpio` format.

If the -RC option is selected, all files recorded in the archive are restored.

refsname if non-null, the name of the filesystem to be restored to instead of *ofsname*.

redev if non-null, the slice to be restored to instead of *ofsdev*.

At least one of *refsname* and *redev* must be null.

If the -RF option is specified, only selected objects from the archive are restored. Each 7-tuple, composed of *rsjobid:uid:date:type:name:rename:inode*, specifies an object to be restored from the filesystem archive. The 7-tuple objects come to incfile from the rsstatus.tab file.

rsjobid the restore jobid assigned by restore or urestore.

uid the real uid of the user who requested the object to be restored. It must match the uid of the owner of the object at the time the archive was made, or it must be the superuser uid.

date the newest "last modification time" that is acceptable for a restorable object. The object is restored from the archive immediately older than this date. *date* is a hexadecimal representation of the date and time provided by the time system call.

type either F or D, indicating that the object is a file or a directory, respectively.

name the name the object had in the filesystem archive.

rename the name that the object should be restored to (it may differ from the name the object had in the filesystem archive). If omitted, the object is restored to *name.*

inode the inode number of the object as it was stored in the filesystem archive. *[inode]* is not used by incfile -R, and is provided only for command-line compatibility with other restoral methods.

Options

Some options are only significant during incfile -B invocations; they are accepted but ignored during incfile -R invocations because the command is invoked and options are specified automatically by restore. These options are flagged with an asterisk (*).

d* Inhibits the recording of the archive in the backup history log.

i* Excludes from the backup those files that have only had an inode change.

l* Creates a long form of the backup history log that includes a table of contents for the archive. This includes the data used to generate a listing of each file in the archive like that produced by the ls -l command.

m* Mounts the originating filesystem read-only before starting the backup and remounts it with its original permissions after completing the backup. Cannot be used with root or /usr filesystems.

o	Permits the user to override media insertion requests [see the getvol(1M), -o option].
r*	Includes remotely mounted resources in the archive.
t*	Creates a table of contents for the backup on additional media instead of in the backup history log.
v*	Validates the archive as it is written. A checksum is computed as the archive is being written; as each medium is completed, it is re-read and the checksum is recomputed to verify that each block is readable and correct. If either check fails, the medium is considered unreadable. If -A has been specified, the archiving operation fails; otherwise, the operator is prompted to replace the failed medium.
x*	Ignores the exception list; backs up all changed or modified files.
A	Establishes automated mode, (i.e., does not prompt the user to insert or remove media).
E*	Reports an estimate of media usage for the archive, then performs the backup.
N*	Reports an estimate of media usage for the archive, but does not perform the backup.
S	Displays a period (.) for every 100 (512 byte) blocks read-from or written-to the archive on the destination device.
V	Displays the name of each file written-to or extracted-from the archive on the destination device.

User Interactions

The connection between an archiving method and the backup command is more complex than a simple fork/exec or pipe. The backup command is responsible for all interactions with the user, either directly, or through the bkoper command. Therefore, incfile neither reads from standard-input nor writes to standard-output or standard-error. A method library must be used [see libbrmeth(3)] to communicate reports (estimates, filenames, periods, status, etc.) to the backup command.

DIAGNOSTICS

The exit codes for incfile are the following:

0 = successful completion of the task
1 = one or more parameters to incfile are invalid.
2 = an error has occurred which caused incfile to fail to complete all portions of its task.

FILES

/etc/bkup/bkexcept.tab	lists the files that are to be excluded from an incremental filesystem backup.
/etc/bkup/bkhist.tab	lists the labels of all volumes that have been used for backup operations.

`/etc/bkup/rsstatus.tab`	tracks the status of all restore requests from users.
`/etc/bkup/bklog`	lists errors generated by the backup methods and the `backup` command.
`/etc/bkup/rslog`	logs errors generated by the restore methods and the `restore` command.
$TMP/filelist$$	temporarily stores a table of contents for a backup archive.

SEE ALSO

`backup`(1M), `bkoper`(1M) `cpio`(1), `cpio`(4), `device.tab`(4), `fdp`(1), `ff`(1M), `ffile`(1), `fimage`(1), `getvol`(1M), `incfile`(1), `labelit`(1M), `libbrmeth`(3), `ls`(1), `restore`(1M), `rsoper`(1M), `time`(2)

NAME

`indicator` - display application specific alarms and/or the "working" indicator

SYNOPSIS

`indicator` [`-b` [*n*]] [`-c` *column*] [`-l` *length*] [`-o`] [`-w`] [*string* ...]

DESCRIPTION

The `indicator` function displays application specific alarms or the "working" indicator, or both, on the FMLI banner line. By default, `indicator` ???? The argument *string* is a string to be displayed on the banner line, and should always be the last argument given. Note that *string* is not automatically cleared from the banner line.

The following options are available:

`-b` *n* — The `-b` option rings the terminal bell *n* times, where *n* is an integer from 1 to 10. The default value is 1. If the terminal has no bell, the screen is flashed instead, if possible.

`-c` *column* — The `-c` option defines the column of the banner line at which to start the indicator string. The argument *column* must be an integer from `0` to `DISPLAYW-1`. If the `-c` option is not used, *column* defaults to `0`.

`-l` *length* — The `-l` option defines the maximum length of the string displayed. If *string* is longer than *length* characters, it will be truncated. The argument *length* must be an integer from `1` to `DISPLAYW`. If the `-l` option is not used, *length* defaults to `DISPLAYW`. NOTE: if *string* doesn't fit it will be truncated.

`-o` — The `-o` option causes `indicator` to duplicate its output to *stdout*.

`-w` — The `-w` option turns on the working indicator.

EXAMPLES

When the value entered in a form field is invalid, the following use of `indicator` will ring the bell three times and display the word WRONG starting at column 1 of the banner line.

```
invalidmsg=`indicator -b 3 -c 1 "WRONG"`
```

To clear the indicator after telling the user the entry is wrong:

```
invalidmsg=`indicator -b 9 -c 1 "WRONG"; sleep(3);
      indicator -c 1 "      "`
```

In this example the value of `invalidmsg` (in this case the default value `Input is not valid`), still appears on the FMLI message line.

NAME

indxbib - create an inverted index to a bibliographic database

SYNOPSIS

/usr/ucb/indxbib *database-file* . . .

DESCRIPTION

indxbib makes an inverted index to the named *database-file* (which must reside within the current directory), typically for use by lookbib and refer. A *database* contains bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '%', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '%' (see addbib).

indxbib is a shell script that calls two programs: mkey and inv. mkey truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed.

indxbib creates an entry file (with a .ia suffix), a posting file (.ib), and a tag file (.ic), in the working directory.

FILES

/usr/ucblib/reftools/mkey
/usr/ucblib/reftools/inv

*.ia	entry file
*.ib	posting file
*.ic	tag file
*.ig	reference file

SEE ALSO

addbib(1), lookbib(1), refer(1), roffbib(1), sortbib(1)

NOTES

All dates should probably be indexed, since many disciplines refer to literature written in the 1800s or earlier.

indxbib does not recognize pathnames.

NAME

inetd - Internet services daemon

SYNOPSIS

inetd [-d] [-s] [*configuration-file*]

DESCRIPTION

inetd, the Internet services daemon, is normally run at boot time by the Service Access Facility (SAF). When started, inetd reads its configuration information from *configuration-file*, the default being /etc/inetd.conf. See inetd.conf(4) for more information on the format of this file. It listens for connections on the Internet addresses of the services that its configuration file specifies. When a connection is found, it invokes the server daemon specified by that configuration file for the service requested. Once a server process exits, inetd continues to listen on the socket.

The -s option allows you to run inetd "stand-alone," outside the Service Access Facility (SAF).

Rather than having several daemon processes with sparsely distributed requests each running concurrently, inetd reduces the load on the system by invoking Internet servers only as they are needed.

inetd itself provides a number of simple TCP-based services. These include echo, discard, chargen (character generator), daytime (human readable time), and time (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). For details of these services, consult the appropriate RFC, as listed below, from the Network Information Center.

inetd rereads its configuration file whenever it receives a hangup signal, SIGHUP. New services can be activated, and existing services deleted or modified in between whenever the file is reread.

SEE ALSO

comsat(1M), ftpd(1M), rexecd(1M), rlogind(1M), rshd(1M), telnetd(1M), tftpd(1M), inetd.conf(4)

Postel, Jon, "Echo Protocol," RFC 862, Network Information Center, SRI International, Menlo Park, Calif., May 1983

Postel, Jon, "Discard Protocol," RFC 863, Network Information Center, SRI International, Menlo Park, Calif., May 1983

Postel, Jon, "Character Generater Protocol," RFC 864, Network Information Center, SRI International, Menlo Park, Calif., May 1983

Postel, Jon, "Daytime Protocol," RFC 867, Network Information Center, SRI International, Menlo Park, Calif., May 1983

Postel, Jon, and Ken Harrenstien, "Time Protocol," RFC 868, Network Information Center, SRI International, Menlo Park, Calif., May 1983

NAME

infocmp - compare or print out *terminfo* descriptions

SYNOPSIS

infocmp [-d] [-c] [-n] [-I] [-L] [-C] [-r] [-u] [-s d | i | l | c] [-v] [-V] [-1] [-w *width*] [-A *directory*] [-B *directory*] [*termname* . . .]

DESCRIPTION

infocmp can be used to compare a binary terminfo entry with other terminfo entries, rewrite a terminfo description to take advantage of the use= terminfo field, or print out a terminfo description from the binary file (term) in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the -I option will be assumed. If more than one *termname* is specified, the -d option will be assumed.

Comparison Options [-d] [-c] [-n]

infocmp compares the terminfo description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: F for boolean variables, -1 for integer variables, and NULL for string variables.

-d produces a list of each capability that is different between two entries. This option is useful to show the difference between two entries, created by different people, for the same or similar terminals.

-c produces a list of each capability that is common between two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the -u option is worth using.

-n produces a list of each capability that is in neither entry. If no *termnames* are given, the environment variable TERM will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of a description.

Source Listing Options [-I] [-L] [-C] [-r]

The -I, -L, and -C options will produce a source listing for each terminal named.

-I use the terminfo names

-L use the long C variable name listed in <term.h>

-C use the termcap names

-r when using -C, put out all capabilities in termcap form

If no *termnames* are given, the environment variable TERM will be used for the terminal name.

The source produced by the -C option may be used directly as a termcap entry, but not all of the parameterized strings may be changed to the termcap format. infocmp will attempt to convert most of the parameterized information, but anything not converted will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where termcap expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All termcap variables no longer supported by terminfo, but which are derivable from other terminfo variables, will be output. Not all terminfo capabilities will be translated; only those variables which were part of termcap will normally be output. Specifying the -r option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output. Mandatory padding is not supported. Because termcap strings are not as flexible, it is not always possible to convert a terminfo string capability into an equivalent termcap format. A subsequent conversion of the termcap file back into terminfo format will not necessarily reproduce the original terminfo source.

Some common terminfo parameter sequences, their termcap equivalents, and some terminal types which commonly have such sequences, are:

terminfo	termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [-u]

-u produces a terminfo source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with use= fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using infocmp will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler tic does a left-to-right scan of the capabilities, specifying two use= entries that contain differing entries for the same capabilities will produce different results

depending on the order that the entries are given in. infocmp will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a use= entry that contains that capability will cause the second specification to be ignored. Using infocmp to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra use= fields that are superfluous. infocmp will flag any other *termname* use= fields that were not needed.

Other Options [-s d | i | l | c] [-v] [-V] [-1] [-w *width*]

-s sorts the fields within each type according to the argument below:

- d leave fields in the order that they are stored in the *terminfo* database.
- i sort by *terminfo* name.
- l sort by the long C variable name.
- c sort by the *termcap* name.

If the -s option is not given, the fields printed out will be sorted alphabetically by the terminfo name within each type, except in the case of the -C or the -L options, which cause the sorting to be done by the termcap name or the long C variable name, respectively.

-v prints out tracing information on standard error as the program runs.

-V prints out the version of the program in use on standard error and exit.

-1 causes the fields to be printed out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

-w changes the output to *width* characters.

Changing Databases [-A *directory*] [-B *directory*]

The location of the compiled terminfo database is taken from the environment variable TERMINFO . If the variable is not defined, or the terminal is not found in that location, the system terminfo database, usually in /usr/share/lib/terminfo, will be used. The options -A and -B may be used to override this location. The -A option will set TERMINFO for the first *termname* and the -B option will set TERMINFO for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people.

FILES

/usr/share/lib/terminfo/?/* Compiled terminal description database.

SEE ALSO

curses(3X), captoinfo(1M), terminfo(4), tic(1M)

NAME

init, telinit - process control initialization

SYNOPSIS

/usr/sbin/init [0123456SsQqabc]

/usr/sbin/telinit [0123456SsQqabc]

DESCRIPTION

init

init is a general process spawner. Its primary role is to create processes from information stored in the file /etc/inittab [see inittab(4)].

At any given time, the system is in one of eight possible run levels. A run level is a software configuration of the system under which only a selected group of processes exist. The processes spawned by init for each of these run levels is defined in /etc/inittab. init can be in one of eight run levels, 0-6 and S or s (run levels S and s are identical). The run level changes when a privileged user runs /usr/sbin/init. This user-spawned init sends appropriate signals to the original init spawned by the operating system when the system was booted, telling it which run level to change to.

The following are the arguments to init.

- 0 — shut the machine down so it is safe to remove the power. Have the machine remove power if it can.
- 1 — put the system in system administrator mode. All file systems are mounted. Only a small set of essential kernel processes are left running. This mode is for administrative tasks such as installing optional utility packages. All files are accessible and no users are logged in on the system.
- 2 — put the system in multi-user mode. All multi-user environment terminal processes and daemons are spawned. This state is commonly referred to as the multi-user state.
- 3 — start the remote file sharing processes and daemons. Mount and advertise remote resources. Run level 3 extends multi-user mode and is known as the remote-file-sharing state.
- 4 — is available to be defined as an alternative multi-user environment configuration. It is not necessary for system operation and is usually not used.
- 5 — Stop the UNIX system and go to the firmware monitor.
- 6 — Stop the UNIX system and reboot to the state defined by the initdefault entry in /etc/inittab.
- a,b,c — process only those /etc/inittab entries having the a, b, or c run level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run level to change.
- Q,q — re-examine /etc/inittab.

S,s enter single-user mode. When this occurs, the terminal which executed this command becomes the system console. This is the only run level that doesn't require the existence of a properly formatted /etc/inittab file. If this file does not exist, then by default the only legal run level that init can enter is the single-user mode. When the system comes up to S or s, file systems for users' files are not mounted and only essential kernel processes are running. When the system comes down to S or s, all mounted file systems remain mounted, and all processes started by init that should only be running in multi-user mode are killed. In addition, any process that has a utmp entry will be killed. This last condition insures that all port monitors started by the SAC are killed and all services started by these port monitors, including ttymon login services, are killed. Other processes not started directly by init will remain running. For example, cron remains running.

When a UNIX system is booted, init is invoked and the following occurs. First, init looks in /etc/inittab for the initdefault entry [see inittab(4)]. If there is one, init will usually use the run level specified in that entry as the initial run level to enter. If there is no initdefault entry in /etc/inittab, init requests that the user enter a run level from the virtual system console. If an S or s is entered, init goes to the single-user state. In the single-user state the virtual console terminal is assigned to the user's terminal and is opened for reading and writing. The command /usr/sbin/su is invoked and a message is generated on the physical console saying where the virtual console has been relocated. Use either init or telinit, to signal init to change the run level of the system. Note that if the shell is terminated (via an end-of-file), init will only re-initialize to the single-user state if the /etc/inittab file does not exist.

If a 0 through 6 is entered, init enters the corresponding run level. Run levels 0, 5, and 6 are reserved states for shutting the system down. Run levels 2, 3, and 4 are available as multi-user operating states.

If this is the first time since power up that init has entered a run level other than single-user state, init first scans /etc/inittab for boot and bootwait entries [see inittab(4)]. These entries are performed before any other processing of /etc/inittab takes place, providing that the run level entered matches that of the entry. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. init then scans /etc/inittab and executes all other entries that are to be processed for that run level.

To spawn each process in /etc/inittab, init reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by /etc/inittab, init waits for one of its descendant processes to die, a powerfail signal, or a signal from another init or telinit process to change the system's run level. When one of these conditions occurs, init re-examines /etc/inittab. New entries can be added to /etc/inittab at any time; however, init still waits for one of the above three conditions to occur before re-examining /etc/inittab. To get around this, init Q or init q command wakes init to re-examine /etc/inittab immediately.

When init comes up at boot time and whenever the system changes from the single-user state to another run state, init sets the ioctl(2) states of the virtual console to those modes saved in the file /etc/ioctl.syscon. This file is written by init whenever the single-user state is entered.

When a run level change request is made init sends the warning signal (SIGTERM) to all processes that are undefined in the target run level. init waits five seconds before forcibly terminating these processes via the kill signal (SIGKILL).

When init receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in /var/adm/utmp and /var/adm/wtmp if it exists [see who(1)]. A history of the processes spawned is kept in /var/adm/wtmp.

If init receives a powerfail signal (SIGPWR) it scans /etc/inittab for special entries of the type powerfail and powerwait. These entries are invoked (if the run levels permit) before any further processing takes place. In this way init can perform various cleanup and recording functions during the powerdown of the operating system.

telinit

telinit, which is linked to /usr/sbin/init, is used to direct the actions of init. It takes a one-character argument and signals init to take the appropriate action.

FILES

/etc/inittab
/var/adm/utmp
/var/adm/wtmp
/etc/ioctl.syscon
/dev/console

SEE ALSO

ttymon(1M), shutdown(1M), inittab(4), utmp(4), utmpx(4), termio(7).
login(1), sh(1), stty(1), who(1), kill(2).

DIAGNOSTICS

If init finds that it is respawning an entry from /etc/inittab more than ten times in two minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either five minutes has elapsed or it receives a signal from a user-spawned init or telinit. This prevents init from eating up system resources when someone makes a typographical error in the inittab file or a program is removed that is referenced in /etc/inittab.

When attempting to boot the system, failure of init to prompt for a new run level may be because the virtual system console is linked to a device other than the physical system console.

NOTES

init and telinit can be run only by a privileged user.

The S or s state must not be used indiscriminately in the /etc/inittab file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the initdefault.

If a default state is not specified in the `initdefault` entry in `/etc/inittab`, state `6` is entered. Consequently, the system will loop, that is, it will go to firmware and reboot continuously.

If the `utmp` file cannot be created when booting the system, the system will boot to state "`s`" regardless of the state specified in the *initdefault* entry in `/etc/inittab`. This can happen if the `/var` filesystem is not accessible.

If the `utmp` file cannot be created when booting the system, the system will boot to state "`s`" regardless of the state specified in the *initdefault* entry in `/etc/inittab`. This can happen if the `/var` filesystem is not accessible.

NAME

`install` - install commands

SYNOPSIS

`/usr/sbin/install` [`-c` *dira*] [`-f` *dirb*] [`-i`] [`-n` *dirc*] [`-m` *mode*] [`-u` *user*] [`-g` *group*] [`-o`] [`-s`] *file* [*dirx* ...]

DESCRIPTION

The `install` command is most commonly used in "makefiles" [see `make`(1)] to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, `install` will search a set of default directories (`/usr/bin`, `/etc`, and `/usr/lib` in that order) for a file with the same name as *file*. When the first occurrence is found, `install` issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The -m, -u and -g options will only succeed for the user if the file's permissions allow the change or the user is the super-user. If any option fails, a warning message will be displayed for each file that fails to be changed.

The meanings of the options are:

`-c` *dira*	Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, `install` issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the `-s` option.
`-f` *dirb*	Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to `755` and `bin`, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the `-o` or `-s` options.
`-i`	Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options except `-c` and `-f`.
`-n` *dirc*	If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to `755` and `bin`, respectively. May be used alone or with any other options except `-c` and `-f`.
`-m` *mode*	The mode of the new file is set to *mode*.
`-u` *user*	The owner of the new file is set to *user*.
`-g` *group*	The group id of the new file is set to *group*.

`-o` If *file* is found, this option saves the "found" file by copying it to `OLD`*file* in the directory in which it was found. This option is useful when installing a frequently used file such as `/usr/bin/sh` or `/usr/lib/saf/ttymon`, where the existing file cannot be removed. May be used alone or with any other options except `-c`.

`-s` Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

`make`(1).

NAME

`install` - install files

SYNOPSIS

`/usr/ucb/install` [`-cs`] [`-g` *group*] [`-m` *mode*] [`-o` *owner*] *file1 file2*

`/usr/ucb/install` [`-cs`] [`-g` *group*] [`-m` *mode*] [`-o` *owner*] *file . . . directory*

`/usr/ucb/install -d` [`-g` *group*] [`-m` *mode*] [`-o` *owner*] *directory*

DESCRIPTION

Install is used within makefiles to copy new versions of files into a destination directory and to create the destination directory itself.

The first two forms are similar to the `cp`(1) command with the addition that executable files can be stripped during the copy and the owner, group, and mode of the installed file(s) can be given.

The third form can be used to create a destination directory with the required owner, group and permissions.

Note: `install` uses no special privileges to copy files from one place to another. The implications of this are:

You must have permission to read the files to be installed.

You must have permission to copy into the destination file or directory.

You must have permission to change the modes on the final copy of the file if you want to use the `-m` option to change modes.

You must be superuser if you want to specify the ownership of the installed file with `-o`. If you are not the super-user, or if `-o` is not in effect, the installed file will be owned by you, regardless of who owns the original.

OPTIONS

`-g` *group* Set the group ownership of the installed file or directory. (staff by default)

`-m` *mode* Set the mode for the installed file or directory. (0755 by default)

`-o` *owner* If run as root, set the ownership of the installed file to the user-ID of *owner*.

`-c` Copy files. In fact `install` *always* copies files, but the `-c` option is retained for backwards compatibility with old shell scripts that might otherwise break.

`-s` Strip executable files as they are copied.

`-d` Create a directory. Missing parent directories are created as required as in `mkdir -p`. If the directory already exists, the owner, group and mode will be set to the values given on the command line.

SEE ALSO

`chgrp`(1), `chmod`(1), `chown`(1), `cp`(1), `install`(1M), `mkdir`(1), `strip`(1)

NAME

ipcrm - remove a message queue, semaphore set, or shared memory ID

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

ipcrm removes one or more messages, semaphores, 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* Removes 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). Use the ipcs command to find the identifiers and keys.

SEE ALSO

ipcs(1)
msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

NAME

ipcs - report inter-process communication facilities status

SYNOPSIS

ipcs [*options*]

DESCRIPTION

ipcs prints 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 -q, -m, or -s are specified, information about only those indicated is printed. If none of these three are specified, information about all three is printed subject to these options:

-b Print information on biggest allowable size: 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, process ID of last process to receive a message on message queues, 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, time of last shmat and last shmdt on shared memory, time of last semop 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*
Use the file *namelist* in place of /stand/unix.

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 are 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 A process is waiting on a *msgrcv*. S A process is waiting on a *msgsnd*. D The associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it. C The associated shared memory segment is to be cleared when the first attach is executed. - The corresponding special flag is not set. The next nine 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 Read permission is granted. w Write permission is granted. a Alter permission is granted. - 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.

FILES

/stand/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

NOTES

If the user specifies either the -C or -N flag, the real and effective UID/GID is set to the real UID/GID of the user invoking ipcs.

Things can change while ipcs is running; the information it gives is guaranteed to be accurate only when it was retrieved.

SEE ALSO

msgop(2), semop(2), shmop(2)

NAME

ixf - software management package extraction facility

SYNOPSIS

ixf [-Vvptdncax] [- { l | m } *list*] [-s *dir*] *prefix*_cX*d*Y*suffix*

DESCRIPTION

ixf is the facility used to read tapes created by igf(1M). Depending on the options given, general information is printed about the contents of the tape, or archives are read. *tape_dev* is the tape device node. It must be a no-rewind type device.

Options

-V Print package version.

-v Verbose mode.

-p Print package name.

-t Print table of contents. This includes all of the archives on the media. In verbose mode, it also lists any operating system and ramdisk entries.

-d Print package description.

-n No rewind. Do not rewind tape before exiting.

-c Print creation date of package.

-a All print options. Combines the following options: -t, -V, -p, -d, -c.

-x If a SCRIPTS entry exists, exit with a special value (13 decimal) to indicate this.

-l *list*
Extract (or give a table of contents if -t option) only the archives in *list*. *list* is a comma-separated list of archive numbers. If the IXF_AR_OFFSET environment variable is set, it is added to each archive number in the list to determine which archives to extract (or list in the table of contents).

-m *list*
Select a list of archives to extract. Before any extraction occurs, the user has a chance to interactively change the input options and destination directory of the selected archives. *list* is a comma-separated list of archive numbers. If the IXF_AR_OFFSET environment variable is set, it is added to each archive number in the list to determine which archives to extract.

-s *dir*
Override the default destination directory for a SCRIPTS entry. Use *dir* instead.

ixf has two operation modes. The first mode prints information about the contents of the tape. There are two types of information, general and table of contents. The general information may be printed with any combination of the following options: -p, -d, -V, -c. A table of contents may be printed using the -t option. The -a option combines both types. The second mode is for extracting files from archives. A combination of these two modes is not allowed.

General information about the contents of the tape is always printed to the standard output path. This information may include the package name, description, version number, and media creation date. Each piece of information is printed on its own line. In verbose mode, each line is prepended with a description of that line's contents. For example, ixf -vp *prefix*_cX*d*Y*suffix*, may print "Package Name - xyz". Without the -v option, it may print "xyz". Device specific special files take

the form *prefix*_cX`d`Y*suffix*, where *prefix* uniquely defines the type of device, *X* specifies the controller number of the stated device type, *Y* specifies the logical device number for the device attached to the stated controller, and *suffix* specifies device dependent information.

The table of contents prints a list of the archives on the tape. If the verbose flag is set, the operating system and ramdisk entries are also printed.

The `Archive Number` (or `Arc Num`) is a unique file number assigned to each archive, starting with 1 for the first archive (which may be a `SCRIPTS` entry).

The `Tape File Number` is the actual record number of that file on the tape, starting at zero for the volume ID.

The `Name` is the original filename of the operating system or ramdisk when the tape was made. This is also the name that must be passed to the bootloader, `/usr/lib/tapeboot`, to boot from an operating system other than the first OS on the tape.

The `Destination Directory` is the directory pathlist that `ixf` changes to before executing the archive command.

The `Input Command/Description` field is the command that will be run to read the archive.

The optional `Description` field is a small description of the archive's contents enclosed in double quotes.

In extraction mode, `ixf` reads the tape directory, then goes to the first archive. Using the archiver, input options, and destination directory stored in the tape directory, it first changes to the destination directory and then executes the command with the input options. It assumes that if the archive command completes successfully, it will advance the tape one record. To do this, the process must open the no-rewind device, perform at least one read, and then close the device. If this is not done, successive commands may read the wrong archive. If the command completes without error, the next archive is read in same fashion until all of the archives are read. However, if the first archive is a `SCRIPTS` entry, `ixf` changes to the default destination directory `/tmp` (or the directory specified by the `-s` option) and then executes the archive command. In this case, no other archives are processed. Upon completion, the tape is rewound.

SEE ALSO

`igf`(1M)

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 ASCII collating sequence on the fields on which they are to be joined, normally the first in each line [see `sort`(1)].

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then 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 options below 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 with string *s*. The string *s* can contain characters from supplementary code sets.

`-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 includes 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. The separator *c* can be a character from the supplementary code sets.

EXAMPLE

The following command line will join 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 ASCII collating sequence on the group ID fields.

```
join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

INTERNATIONAL FUNCTIONS

`join` can process characters from supplementary code sets in the current locale, as well as ASCII characters.

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

NOTES

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 the `join`, `sort`, `comm`, `uniq`, and `awk` commands are wildly incongruous.

Filenames that are numeric may cause conflict when the `-o` option is used just before listing filenames.

NAME

jwin - print size of layer

SYNOPSIS

jwin

DESCRIPTION

jwin is functional only under layers(1) and is used to determine the size of the window associated with the current process. It prints the width and the height of the window in bytes (number of characters across and number of lines, respectively). For bit-mapped terminals only, it also prints the width and height of the window in bits.

EXIT STATUS

Returns 0 on successful completion, 1 otherwise.

DIAGNOSTICS

If layers(1) has not been invoked, an error message is printed:

```
jwin: not mpx
```

NOTE

The window whose size is printed is the one attached to standard input; that is, the window you are in when you type the jwin command.

SEE ALSO

layers(1)

EXAMPLE

```
jwin
bytes:  86 25
bits:   780 406
```

NAME

kbdcomp - compile kbd tables

SYNOPSIS

`kbdcomp [-vrR] [-o` *outfile*`] [`*infile*`]`

DESCRIPTION

`kbdcomp` compiles tables for use with the `kbd` *STREAMS* module, a programmable string-translation module. The module has two separate abilities, each of which may be used alone or in combination.

The first ability, *lookup*, is that of performing simple substitution of bytes in an input stream. This ability is based on a simple 256-entry lookup table (as there are 256 possible bit combinations for a byte). As input is received, each byte is looked up in the translation table, and the table value for that byte is substituted in place of the original byte. The process is quick, and can be performed on each *STREAMS* message with no message copying or duplication.

The second ability, *mapping* allows searching for occurrences of specified strings of bytes (or individual bytes) in an input stream, and substituting other strings (or bytes) for them as they are recognized. There are three kinds of mapping that are differentiated by the relationship between the number of bytes in the input and the number of bytes in the output. *One-many* mapping means that for a given byte in the input, *many* bytes are substituted. *Many-one* mapping means that for many bytes in the input *one* byte is substituted. *Many-many* mapping includes the other two types as a proper subset, but also includes substitution of *many* bytes in the input with *many* bytes of output. KBD can perform all three types of mapping. The *lookup* ability described in the previous paragraph (that is, what amounts to *one-one* mapping) is a common special case useful enough to be included separately. By using combinations of both *lookup* and *mapping*, a larger class of input translation and conversion problems can be solved than can be solved by the use of either alone.

During operation, processing occurs in two major passes: the lookup table pass *always* precedes string mapping. The string mapping procedure is non-recursive for a given table and there is no feedback mechanism (that is, input is scanned in order as received and output is not re-scanned for occurrences of recognizable input strings). As an example of mapping, suppose one wishes to translate all occurrences of the string `this` in an input stream into the string `there`. The module recognizes and buffers occurrences of the string `th` (as each byte is received); if the following character is `i`, it will also be buffered, but if `x` is then received, a mismatch is recognized and no translation occurs. Assuming `thi` has been buffered, if the next character seen is `s`, a match is recognized, the buffer containing `this` is discarded, and the string `there` replaces it.

It should be obvious that both input and output strings can be of any non-zero length (see however, the section below on limitations). Each string to be recognized and translated must be unique, and no complete input string may constitute the leading substring of any other (for example, one may not define `abc` and `ab` simultaneously, but may so define `abc`, `abd`, and `abxy`).

Given a filename (or standard input if no name is supplied), `kbdcomp` will compile tables into the output file specified by the `-o` option. If the `-o` option is not supplied, output is to the file `kbd.out`.

The `-v` option causes parsing and verification—no output file is produced; if no error messages are printed, then the input file is syntactically correct. The `-r` option causes the compiler to check for and report on byte values that cannot be generated in a table (see the description below). The option `-R` is equivalent to `-r` but it tries to print printable characters as themselves rather than in octal format.

Input Language

Source files for `kbdcomp` are a series of table declarations. Within each table declaration are a number of definitions and functions. A table declaration is one of the forms `map`, `link`, or `extern`:

```
map type ( name ) { expressions }

link ( string )
extern ( string )
```

The `link` and `extern` forms will be described later below. The *name* of a `map` must be a *simple token* not containing any colons, commas, quotes, or spaces. (For our purposes, a *simple token* is a sequence of alphabetic and/or numeric characters with no embedded punctuation, white space, or special symbols.) The *type* field is an optional field that may be either of the keywords `full` or `sparse`. If omitted, the type defaults to `sparse`. The effect of this field is described in more detail below. The expressions contained in the `map` declaration are one of the following forms. Reserved keywords are printed in constant-width font, variables in italics:

```
keylist ( string string )
define ( word value )
word ( extension result )
string ( word word )
strlist ( string string )
error ( string )
timed
```

The `keylist` form is for defining lookup table entries while the remaining forms are the separate string functions.

The definition form (`define`) allows a mnemonic word (the first argument) to be associated with a string (the second argument). It is useful for replacing complicated sequences (for example, those containing special symbols or control characters) with mnemonic words to facilitate the design and readability of tables.

Using the *word* form (where *word* must be a previously defined sequence) in a manner similar to a `C` function call results in the *value* of *word* being concatenated with *extension*; when the combination is recognized at runtime, it is mapped to *result*. The *value* may be a string of characters or a single byte. The following is an illustration (not intended to be complete):

```
map (some_accents) {
     define(acute '\047')
     define(grave '`' )
     acute(a '\341')        # same as string("\047a" "\341")
```

```
        grave(a '\340')
        # ...et cetera...
        keylist("zyZY" "yzYZ")
}
```

This map (above) defines the single quote and reverse quote keys as *dead-keys*, which when followed by *a* produce a character from the *ISO 8859-1* codeset. It is not necessary for the definition, extension, or result to be a single byte; they may be arbitrary strings.

Strings in definitions and arguments may generally be entered either without quotation or between double quotes. Byte constants may likewise be entered unquoted or between single quotes. The only time quotation is strictly required is when the string contains parentheses, spaces, tab characters, or other special symbols. The language makes no real distinction between byte constants and string constants: both are treated as null-terminated strings; the choice of whether to use a one-character string or a byte constant is thus a matter of taste. Most quoting conventions of `C` are recognized, except that *octal* constants must be exactly three digits long. Octal constants may be used in strings as well. In the example above, the arguments to `keylist` need not be quoted, as they contain no special symbols. The following example illustrates some situations where strings *must* be quoted:

```
string(abc "two words")      # literal space
keylist("[{}]" "(())")       # brackets/parentheses
define(esc_seq "\033\t(")    # tab and parenthesis
define(space ' ')            # literal space
string(abc "keylist")        # keyword used as argument
```

Comments in files (inside or outside of map declarations) may be entered in the same manner as for `sh`(1); that is, after a `#` at the end of a line, or on a line beginning with `#`, as shown in the above examples.

The `keylist` form allows single bytes to be mapped to other single bytes; it defines actions that are treated in the lookup table (that is, are performed before mapping). Any byte value that is not explicitly changed by being included in a `keylist` form will, of course, be left unchanged; if no `keylist` forms appear in a map definition, then `kbdcomp` does not generate a lookup table for the map, and the lookup phase is skipped during module operation. Each byte in the first string argument to `keylist` is mapped to the byte at *the same position* in the second string argument. That is, given two strings X and Y as arguments: X_i maps to Y_i, X_j maps to Y_j and so forth. The two arguments must, after evaluation, be found to contain the same number of bytes.

The `string` form has a function similar to mnemonic forms defined with `define` and may be used for any type of many-many mapping. The first argument to `string` is mapped to the second argument (see the comment in the sample map above).

Mappings using both `keylist` and `string` or any `define` forms may be combined: if *i* is mapped to *a* with a `keylist` form, and *a* is used in the sequence `a*, then when the user types `i*, the sequence `a* is seen by the string mapping process (because *lookup* is done first) and translated accordingly.

The `keylist` form is intended mainly for use in simple keyboard re-arrangement and case-conversion applications; `string` is for one-many mapping or for isolated instances of many-many mapping; the `define` form and words defined with it are intended for more general use in groups of related sequences. In some situations while a one-one mapping with `keylist` may be an obvious choice, the same effect may be achieved with `string` forms to avoid having a contradictory mapping. For example, suppose one desires, simultaneously, to translate *x* into *y* and *y* into *abc*. If *x* is mapped to *y* via a `keylist` form and *y* is mapped to *abc* via a `string` form, then it may be impossible to obtain *y* itself (unless defined in another sequence), even though that was not the intention—the intention was to obtain *y* whenever the user enters *x*. This is a *contradictory mapping*:

```
keylist(x y)
string(y abc)     # "y" itself cannot be generated
```

There are cases where the intention is that *y* not be generated, but most often the intention is to generate it. This problem (a relatively common one in codeset mapping) can be "solved" by using a `string` form to map *x* to *y* initially rather than using a `keylist` form. This allows both *y* and *abc* to be generated:

```
string(x y)
string(y abc)
```

Entering a large number of one-one mappings with `string` can be somewhat tedious. To make things easier, the `strlist` form is provided. The two string arguments to `strlist` are interpreted in the same manner as arguments to `keylist`, (that is, they are one-one mappings) except that they are not done by the lookup table, but are processed as string mappings. In the following example, the first three `string` definitions can be reduced to the `strlist` form which follows:

```
string(a b)
string(c d)
string(e f)

strlist(ace bdf)
```

It is important to recognize the difference between `string` and `strlist`: with `string`, the two arguments are a *single* mapping definition (which may be of any type) whereas with `strlist`, one or more *one-one* string mappings are defined simultaneously. A set of mappings defined with a combination of `string` and `strlist` do *not* exhibit the same type of incompatibility described above between `keylist` and `string`.

Some further aspects of module processing can now be presented. When a partial match in an input sequence is detected during string processing, it is buffered; if at some point the match no longer succeeds, the *first byte* of the matched buffer is normally sent to the neighboring module. The rest of the input is left in the buffer and scanned again to see if it matches the beginning of another sequence. The `error` entry allows one to send a string (or byte) constant (called a *fallback character*) instead of the byte that began the previous sequence; this is particularly useful in codeset mapping and conversion applications where the character which failed to be translated might be one which does not occur or has some other meaning in the target codeset. The following (somewhat contrived) example illustrates use of the `error` form:

```
# turn arrow keys into vi commands
map (vi_map) {
      string("\033[A" k) # up
      string("\033[B" j) # down
      error("!")
}
```

Given input of the *escape* character followed by `[A` or `[B`, a single character (`j` or `k`) is generated. If presented with the sequence `escape-[Q`, the module will produce the sequence `![Q`. The error string `!` replaces *escape* because the sequence failed to match when `Q` was received. The remaining characters are re-scanned, and neither `[` nor `Q` is found to begin a recognized sequence.

One-one mapping with strings or other defined forms (rather than via a `keylist` lookup table) is generally performed with a linear search operation when looking for bytes which begin sequences. However, if the table is specified as a `full` table, it is initially indexed rather than searched linearly, and thus processed much more quickly when there are a large number of entries. This should be kept in mind in codeset mapping applications where nearly all characters are mapped, and many (or most) are one-one mappings. If only a very few characters are mapped with string functions, one must decide on whether to trade a small gain in processing speed for the space needed to store the index if a table is made `full`.

The `link` form is used to produce a *composite table*. A composite table is really a form of linkage that allows several tables to be used together in sequence as if the sequence were a single table. The string argument to `link` is of the following form:

`composite:component`*1*`,component`*2*`,component`*n*

The target composite name is followed by a colon, and the ordered component list is comma-separated. If the string argument contains spaces or special characters, it must be quoted. (This string is *not* interpreted by `kbdcomp`, but is left intact in the output file; it is interpreted by the module at runtime.) When a composite table is used, the effect is similar to pushing more than one instance of the `kbd` module in the sense that the component tables function sequentially but it is accomplished within a single instance of the module. As output is produced by processing with one table in the composite, the data is subsequently processed by the next component and so forth until the final result emerges at the end of the sequence. (There is no restriction on the use of any combination of `full` and `sparse` tables in a composite.)

Composite tables are useful for simplifying complex mapping situations by modularizing the processing and for increasing the re-usability of tables for different mapping applications. Tables primarily implementing codeset mappings may be linked to other tables primarily implementing compose- or dead-key sequences. With a single table implementing a common codeset mapping, several different tables implementing combinations of codeset mapping and compose-key layouts may be built. A typical configuration might use one table for mapping from an external to internal codeset, then use one or more separate tables *working in the internal codeset* to provide compose- or dead-key functionality, as in the following example. One table, `646Sp-8859` maps from an ISO 646 variant (Spanish) external codeset to ISO 8859-1; this is combined with two other tables respectively implementing ISO 8859-1 by compose-sequences, and by dead-key sequences:

```
link("composed:646Sp-8859,8859-1-cmp")
link("deadkey:646Sp-8859,8859-1-dk")
```

Composite tables can also be built while the module is running from the `kbdload` command line [see kbdload(1M) for details]. The component tables are linked and processed in the given order (left-to-right). Because the `link` argument is actually parsed at run-time by the `kbd` module, it is not an error to refer to tables that are not contained in the file currently being compiled. An error will be generated when the file is loaded if any component of a link is not present in memory at that time.

The `extern` form can be used to declare an external function managed by the `alp` module. External functions are managed in a list by that module, and are available for use as if they were simple tables in `kbd`. External functions are not downloaded, they are resident in the kernel and merely accessed by the `kbd` module [see alp(7) for more information]. Such functions can also be declared dynamically when needed [see kbdload(1M)].

The directive `timed` may appear any place within a `map` declaration. If used, it causes the table within which it is defined to be interpreted in *timeout mode*. In this mode, string mappings are considered to not match if more than a specified amount of time elapses after receipt of the first byte of a sequence without it is fully received and mapped. Given a timed map in which `abc` is to be mapped to `xyz` and the timeout value is 30, if the user types `ab`, then waits for longer than 30 time units before typing `c`, the entire sequence will not be translated. In this case the sequence is treated as any other mismatch would be: `a` is passed to the neighboring module, and `b` is checked to see if it begins a sequence. The timer is reset when a mismatch occurs, so that if `bc` is defined in this situation and `c` has just been received, it will be mapped as expected. The default timeout is typically 1/5 to 1/3 of a second [see kbd(7) for details].

Timeout mode is generally useful in situations where terminal *function keys* are being interpreted, to distinguish between a string typed by the user and a function key string sent by the terminal; it is not intended for use with "batch" applications such as the `iconv` command, nor generally in pipelines [see pipe(2)]. In a composite table, some components may be timed and some not, making the mode useful for combinations of codeset mapping and function key mapping.

Timing depends on several factors, including terminal baud-rate, system load, and the user's typing speed. If the timeout value is too long, then typed sequences that happen to be the same as function keys will be erroneously mapped; if the value is too short, then function keys may be missed under a heavy system load or with low speed devices. See kbdset(1) for information on how to change the timeout value, and kbd(7) for information on how an administrator may change the *default* timeout value. This directive should never be used in tables that implement codeset mapping, as it makes the results quite unpredictable. Long timeouts, on the order of seconds, may be useful in some contexts.

Building & Debugging

Users who intend to build their own tables may study the source tables supplied with the distribution in the directory `/usr/lib/kbd`.

If characters other than alpha-numerics are to be used, quoted strings are preferred to unquoted strings; quotation is required for some characters, as mentioned above. Map names and the first arguments of `define` should be alpha-numeric tokens.

The report generated by the `-r` option may be useful for debugging complex tables. The report (produced on standard error) consists of two octal lists. One list contains byte values that cannot be generated from the lookup table (if `keylist` forms are used). The other list contains byte values that cannot be generated in any way; in other words, values that are *neither* parts of "result text" (that is, products of string mappings) *nor* generated by the lookup table (if there is one), but that are *used* in other sequences. The report does not exhaustively list unreachable paths, but may indicate whether they exist and help pinpoint them.

Output Files

The files produced by `kbdcomp` begin with a header. The magic string is `kbd!map`, with a version number. This header is immediately followed by the tables themselves. (A file can contain more than one table.) The lines below can be added to the `/etc/magic` file for the `file` command to recognize `kbd` files.

```
0       string          kbd!map         kbd map file
>8      byte            >0              Ver %d:
>10     short           >0              with %d table(s)
```

LIMITATIONS

A maximum length of 128 bytes for input strings and 256 bytes for output strings is imposed. The total amount of space consumed by a single table is limited to around 65,000 bytes. Versions are strictly incompatible; "object" tables are machine-dependent in their byte order and structure size. Thus, while source files are portable, the output of `kbdcomp` is not. This implies that when using remote devices across a network between heterogeneous machines, tables must be loaded on the machine where the module is actually pushed (that is, the remote side).

FILES

`/usr/lib/kbd`	directory containing system standard map files
`/usr/lib/kbd/*.map`	source for some system map files

SEE ALSO

`kbdload`(1M), `kbdset`(1), `iconv`(1), `alp`(7), `kbd`(7).

NAME

kbdload - load or link kbd tables

SYNOPSIS

`kbdload [-p]` *filename*

`kbdload -u` *table*

`kbdload -l` *string*

`kbdload -L` *string*

`kbdload -e` *string*

DESCRIPTION

Tables included in the file *filename* are *loaded* into the `kbd` *STREAMS* module, which must already have been pushed into the standard input *STREAM*. (In this context *loaded* means copied from a disk file into main memory within the operating system.) This program is intended both to provide for loading and linking of both *shared* or *public* tables and *private* tables implementing user-specific functionality. New users should refer to kbdcomp(1M) and kbd(7) for a general description of the module's capabilities.

Files are searched for only by the name given on the command line; no search path is implied. Tables loaded by the super user with the `-p` option from an absolute path beginning at `/usr/lib/kbd` are made publicly available and permanently resident, otherwise the loaded tables are available only to the caller, and are automatically unloaded when the `kbd` module is popped from the *STREAM*.

The `-u` option can be used to unload private tables and by the super-user to remove public tables. Tables may be unloaded only if they are not currently in use. (Tables which are members of *composite tables* always have non-zero reference counts since they are "used" in the composite; all composites which refer to them must be unloaded first.)

The `-L` and `-l` options are used for making composite tables on-the-fly. The `-L` option, if executed by the super-user, causes the composite to be made publicly available; it is otherwise private and `-L` is equivalent to `-l`. The `string` argument is constructed in the same manner as the `link` statement [see kbdcomp(1M)] in the compiler. If any component of the intended composite is not presently loaded in memory or if a component of a *public* table is not also *public*, an error message is printed and the linkage fails. More than one composite may be created in a single invocation by using either option sequentially.

The `-e` option with a string argument causes `kbdload` to declare to the `kbd` module a subroutine called *string*, which is assumed to be a subroutine managed by and registered with the `alp` module [see alp(7)]. These "external" subroutines may be used exactly as any other loaded table; they may participate as members of composite tables, etc.

Security Issues

Allowing users other than the super-user to load public tables is a security risk and is thus disallowed. (In general, any manipulation of a module instance by a user who is neither the super-user nor the user who originally pushed it is disallowed.) The library directory and all files contained in it should be protected by being *unwritable*. Administrators are encouraged to remember that the `kbd` system can be

used to arbitrarily re-map the entire keyboard of a terminal, *as well as* the entire output *STREAM*; thus in extremely hostile environments, it might be prudent to remove execution permissions from `kbdload` for non-administrative users (for example, setting the owner to *bin* or *root* and giving it a mode of 0500).

The `kbdload` command checks to insure that the real-uid of the invoker is the same as the *owner* of both standard input and standard output files, unless the real-uid of the invoking user is the super user. Paths to public tables are scrutinized for legitimacy. The `kbdload` command refuses to work as a *set-uid* program.

EXIT VALUES

Exit status is 0 if all tables could be loaded and/or all operations succeeded. In the event of any I/O error (for example, attempting to load a table with the same name as one already loaded and accessible to the caller) or failure to load a table, exit status is 1 and a message is printed indicating the error.

CAVEATS

Composite tables may be unloaded while they are actually in use without affecting current users, though *new* users may no longer attach to it. This is because composite tables are copied and expanded when they are attached in order to keep state information related to the attaching user. The "original" composite always has a zero reference count, and is never itself attached. This is not strictly a bug, it's an "anomaly"; the effect on the user is that a composite table may be attached and functional, yet not appear in the output of a `kbdset` query.

FILES

`/usr/lib/kbd` directory containing system standard map files

SEE ALSO

`kbdcomp`(1M), `kbdset`(1), `alp`(7), `kbd`(7).

NAME

kbdpipe - use the KBD module in a pipeline

SYNOPSIS

`kbdpipe -t` *table* `[-f` *tablefile*`] [-F] [-o` *outfile*`] [`*infile(s)*`]`

DESCRIPTION

`kbdpipe` allows the use of KBD tables as pipeline elements between user programs [see kbdcomp(1M) and kbd(7) for general descriptions of the module and its capabilities]. The `kbdpipe` command is mostly useful in codeset conversion applications. If an output file is given, then all *infiles* are piped to the given output file. With no arguments other than `-t`, standard input is converted and sent to standard output.

The required option argument `-t` identifies the table to be used for conversion. If the table has already been loaded as a shared table it is attached. If, however, the table has not been loaded, an attempt is made to load it. If the given table name is not an absolute pathname then the name of the system mapping library is prepended to the argument, and an attempt is made to load the table from the resulting pathname (that is, it becomes an argument to the loader, `kbdload`). Assuming the table can be loaded, it is attached.

The argument to `-f` defines the filename from which the table will be loaded, overriding the default action described above. The file is loaded (in its entirety), and the named table attached. This option should be used if the default action would fail.

The output file specified by `-o` must not already exist (a safety feature.) The option `-F` may be used to override the check for existence of the output file; in this case, any existing *outfile* will be truncated before being written.

EXAMPLES

The following example converts two input files into relative nonsense by mapping ASCII into Dvorak keyboard equivalents using the `Dvorak` table. The table is assumed to reside in the file `/usr/lib/kbd/Dvorak`. The existing output file is forcefully overwritten:

```
kbdpipe -F -t Dvorak -o iapxai.vj file1 file2
```

The following example loads the `Dvorak` table from a different file, then converts standard input to standard output. The Dvorak table (assumed to be non-resident) is explicitly loaded from an absolute path beginning at the user's home directory:

```
kbdpipe -t Dvorak -f $HOME/tables/Dvorak.tab
```

LIMITATIONS

Because `kbdpipe` uses the `kbdload` command to load tables, it cannot resolve link references. Therefore, if a composite table is to be used, the relevant portions must either be already loaded and public, or be contained in the file indicated (via the `-f` option) on the command line; in this case, the composite elements must be loaded earlier than the link entry.

CAVEATS

Users may now use KBD tables in programs at user level, by just opening a pipe, pushing the module, and setting via related commands; there is thus no need to use `kbdpipe`. This command may not be supported in future releases of the system.

FILES

`/usr/lib/kbd` directory containing system standard table files

SEE ALSO

`kbdload`(1M), `kbdset`(1), `kbd`(7).

NAME

kbdset - attach to kbd mapping tables, set modes

SYNOPSIS

`kbdset [-oq] [-a` *table*`] [-v` *string*`] [-k` *hotkey*`] [-m` *x*`] [-t` *ticks*`]`

`kbdset [-oq] [-d` *table*`] [-v` *string*`] [-k` *hotkey*`] [-m` *x*`] [-t` *ticks*`]`

DESCRIPTION

`kbdset` is the normal user interface to the `kbd` *STREAMS* module [see kbdcomp(1M) and kbd(7) for general descriptions of the module's capabilities]. The `kbdset` command allows users to attach to pre-loaded tables, detach from tables, and to set options. Options are provided for setting hot-keys to toggle tables and for controlling modes of the module.

Arguments and options are scanned and acted upon in command line order. If the `-o` option is given, subsequent options affect the output side of the *STREAM*, otherwise the input side is assumed.

Presence of the `-q` option causes the `kbdset` command to list tables which can be accessed by the invoking user. In this case all subsequent options are ignored. The output from the `-q` option lists the user's current hot-key settings, current timer values, and for each available table an identifier, the name, size, attachments (input and/or output sides), reference count, number of components, and type (private or public). In the following example, there is one composite table, two tables are attached on the input side, and one on the output side.

```
In Hot Key = ^_
Timers: In = 20 ; Out = 20
ID        Name             Size I/O Ref Cmp Type
4039f300  Ucase              56 - o   1   -  ext
403a0480  Case/Dvorak        68 - -   0   2  pri
            [4039f300]  [4037e400]
4036ce00  Deutsche          332 i -   4   -  pub
4037e400  Dvorak            312 i -   2   -  pri
```

The `ID` field is an identifier unique to a given table (actually its address in memory). Currently attached tables are marked `i` or `o`, otherwise the `I/O` fields are marked with a dash. `Ref` is a reference count of attached users (including composites that refer to simple tables) and if non-zero, indicates that the table is in use. `Size` is the total size in bytes of the table and associated overhead in memory. If the table is a composite table, the `Cmp` field contains a number instead of a dash, and the following line lists an identifier for each component, in order of processing (allowing identification of the components in a composite table). Publicly available tables are marked with the type `pub` and private tables with `pri`. Private tables are available only to the invoking user and within the current *STREAM*. Tables which are really external functions [see kbd(7)] are marked *ext*; they are always of the type `pub`. Tables that are interpreted in timeout mode [see kbdcomp(1M)] have an asterisk (`*`) preceding the `Type` field; members of composite tables that are interpreted in timeout mode have an asterisk after their bracketed identifier (on the second output line). External functions are never time-sensitive, unless by their own internal specification.

The option `-a` accompanied by an argument attaches to the named table. A table may not be multiply attached by a single user. When a table is attached and no other table is already attached then the table is automatically made current. The option `-d` detaches from the named table [see kbdload(1M) for a description of how tables are loaded].

The `-k` option sets the user's hot-key. Setting a hot-key with only a single active table allows mapping to be toggled on and off, depending on the hot-key mode. A hot-key is a single byte, typically set to a relatively unused control character, that is caught by the `kbd` module and used for module control rather than being translated in any way. The key used as a hot-key becomes unavailable for other uses (unless it is generated by mapping). The hot-key may be reset at any time, independently from other options. Note that `kbdset` does not interpret ^X-type sequences; it expects a literal hot-key character.

The `-m` option with an integer argument controls the hot-key mode. Legal modes are 0, 1 (the default), and 2. Mode 0 allows one to toggle through the list of attached tables. Upon reaching the end of the list, the cycle returns to the beginning of the list. Use of mode 0 with only one table loaded does not allow mapping to be turned off. Mode 1 toggles to the unmapped state upon reaching the end of the list (for example, given two tables, the sequence is table1, table2, off, table1, and so on). Mode 2 toggles to the unmapped (or off) state between every table in the list of attached tables (for example, given two tables, the sequence is table1, off, table2, off, table1, and so on).

The `-v` option turns on verbose mode, which can be useful when multiple tables are used in interactive sessions. In verbose mode, the name of the table can be output to the terminal whenever the user changes to a new table with the hot-key. The string associated with the option can be any short string. If the character sequence `%n` appears in the string, the name of the current table (or a null string) will be substituted for the `%n`. (A null argument to `-v` is equivalent to terse mode.) One useful sequence for this mode is *save-cursor, goto-status-line, clear-to-end-of-line, ''%n'', restore-cursor*. This causes output of the current table name on the terminal's status line; in the absence of a status-line, a simple sequence is to print the table name and RETURN [see terminfo(4) for the appropriate escape sequences]. Verbose mode is only available to show input table status to the output side of the *STREAM*. The output string for verbose mode is not itself passed through the mapping process, but is transmitted directly downstream with no other interpretation (it should thus be a string of ASCII characters or in some other externally available codeset).

The `-t` option with an argument is used to change the timer for tables in the *STREAM* that are interpreted in timeout mode. Values (in ''clock ticks'') between 5 and 400 are acceptable. (Depending on the hardware, the clock is usually either 60Hz or 100Hz, thus one tick is either 1/60 or 1/100 of a second; with a bit of experimentation, a suitable value for one's own system and typing speed can be found.) When a table that uses timeout mode is attached, it is assigned the current timer value. All tables that are attached after setting the timer value will take on the new value, but tables currently attached are unaffected (this allows one to set different values for different tables). The option does not affect other users' values. The timer value may be set independently for input and output sides by using `-t` in conjunction with `-o`. The value for a currently attached table may be reset by detaching the table, setting the value, then re-attaching the table.

In the query output, the line beginning with `Timers:` shows the timer values for input and output sides of the module.

LIMITATIONS

A table may be detached while it is current; however, in this case, it is first made non-current; this allows error recovery under adverse circumstances. Detachment of a current table is not affected by the current hot-key mode, but always toggles to a state where no table is current.

CAVEATS

It is not possible with the `-q` option to see the timer values assigned to currently attached tables, nor to reset the value for a table that is currently attached.

FUTURE DIRECTIONS

Better control of timeout mode and values should be provided.

FILES

`/usr/lib/kbd` directory containing system standard map files

SEE ALSO

`alpq`(1), `kbdcomp`(1M), `kbdload`(1M), `alp`(7), `kbd`(7).

NAME

kcrash - examine system images

SYNOPSIS

kcrash [-w] [-c | -k] *dumpfile* [*namelist*]

DESCRIPTION

The kcrash program, similar to the crash(1M) program, examines system crash dumps. Unlike crash, the kcrash command interface is based on the kernel debugger [see kdb(1M)]. All commands accepted by the kernel debugger can be used identically in kcrash, with the following exceptions:

- Execution commands (for example, go and tr) do not work.
- Multi-processor commands (for example, gos, ss, and cpu) do not work.
- Instruction and memory breakpoint commands do not work.
- Instruction tracing commands do not work.

Commands that modify memory (actually modify the crash dump file) work only if the -w flag is present in the command line.

If the -k flag is present, *dumpfile* can be /dev/mem, allowing kcrash to be used on the running system. In addition, the following commands work only in kcrash (not in the kernel debugger):

If the -c flag is present, *dumpfile* will be interpreted as a crashdump file that was obtained by the *crashdump* function in the kernel. Otherwise it will be assumed that *dumpfile* was created by the rom debugger.

< filename
<< filename
: Read and execute commands from the given file. Note that these commands are like *dbcmd*(1M) used with the kernel debugger.

! shell-command
!! shell-command
: Executes the given shell command.

q
qq
quit Quits kcrash.

FILES

/usr/lib/kdb/macros/* macros that are useful for kernel debugging
/unix default namelist

SEE ALSO

crash(1M), kdb(1M)

NOTES

The back trace command runs very slow when used to examine a running system.

NAME

kdb - kernel debugger (with multi-processor support)

DESCRIPTION

The kernel debugger is a simple debugger that resides in the UNIX kernel and allows the programmer to examine and modify memory, disassemble instructions, download and execute programs, set breakpoints, and single-step instructions, on all the on-line processors.

You can configure the kernel debugger as part of the kernel load file (/stand/unix). Type the string "@@P" (configurable in the kdb master file) on the console or push the software abort button to enter the debugger.

Multi-processor Support (88K only)

The kernel debugger allows each processor independently to be either in the debugger or running at any time. Processors in the debugger are in one of two modes: *master mode* or *slave mode*. At most one processor is in master mode at any time, although master mode may be transferred among the processors with a debugger command described below. When any processor enters the debugger from a state in which all processors are running, that processor becomes the master and forces all the other processors to become slaves, thereby suspending execution over the entire multi-processor system. All the commands described below execute on the current master processor unless otherwise noted. Slave processors do nothing until instructed by the master as a result of a debugger command.

Commands

All debugger commands are brief mnemonics (usually two characters) followed by zero or more arguments. In the following descriptions, optional arguments are enclosed in square brackets. Arguments are separated by spaces or commas, and each argument must be one of the following:

1. A number in the current input radix (default hexadecimal), or in a different radix as specified by a prefix: 0x for hex, 0t for decimal, 0o for octal, or 0b for binary.
2. A percent sign followed by a register name, meaning the contents of that register, such as %r4, %r13, %sxip for the 88K platform and %d1, %a1, and %pc for the 68K platform.
3. A percent sign followed by b and an instruction breakpoint number, meaning the address referred to by that breakpoint, such as %b*x*.
4. A dollar sign, meaning the address of the last memory location that was displayed.
5. The name of a kernel symbol. This works only if the kernel debugger has been loaded with the UNIX symbol table by using the dbsym(1M) command. A sharp (#) prefix to a name forces the interpretation as a symbol, and not a hex number. (Without this, the name add for example would always be interpreted as 0xADD.)
6. The name of a user-defined debugger variable.
7. Any of the above combined by using the usual arithmetic operators (+ - * / & | ^), the relational operators used in the C programming language (== != <> <= >=), or the C language pointer-dereference operator (*). Two special operators perform instruction arithmetic: *A* @- *B* backs up *B* instructions

from address *A*; *A* `@+` *B* advances *B* instructions from address *A*; where *A* and *B* are expressions. All operators have equal precedence. Use parentheses to force a particular order of evaluation. Division by zero yields zero.

8. A string surrounded by single- (') or double-quotes ("). The C escape for the newline character (`\n`) may be used in the string.
9. A percent sign followed by `s` and an expression, meaning the NULL-terminated character string starting at that memory address.
10. A percent sign followed by `p` and an expression, meaning the physical memory address corresponding to that virtual address.
11. A percent sign followed by `v` and an expression, to test that virtual address' validity. If the virtual address is valid, this operation evaluates to 1, otherwise to 0.
12. A percent sign followed by a 1, 2, or 4 and an expression, meaning a one, two, or four byte pointer-dereference.
13. Any numeric expression preceded by `~`, meaning the ones-complement of the number.

Input Commands

The multi-processor debugger prompts with `K`*n*`>` , where *n* is the processor identification number (*cpuid*) of the current master processor, in hex. The single-processor debugger prompts with `K>`. This prompt indicates that the debugger is ready to accept any of the commands described below. Input characters can be erased with BACKSPACE or DEL. An entire input line can be erased with CTRL-U or CTRL-X. In addition, the debugger supports flow control (CTRL-S, CTRL-Q) and keyboard interrupt (CTRL-C).

On a multi-processor system, before each time the debugger issues a prompt, it checks the state of each processor and notifies the user of those processors that have entered slave mode since the last check. In this way, the user is kept informed of the activities on all the processors.

After a breakpoint or debug trap, the debugger prints a status line describing the trap, immediately followed by the debugger prompt, and is again ready to accept commands. In the case of a trace trap, the debugger automatically supplies the expected command, `tr`. If you want to enter a different command, erase the `tr` and retype a new command.

During any of the display, modify, examine or write commands, you can enter one of the following:

RETURN
: Moves to the next item.

+n
: Moves to the *n*th next item.

-
: Moves to the previous item.

-n
: Moves to the *n*th previous item.

=*addr* Moves to the item at address *addr*. Only valid when operating on memory, not on registers.

n Changes the value of the item to *n*. Only valid for modify or write commands, not display or examine. The `mi` command allows you to enter multiple numbers separated by spaces, to change more than one byte.

. (Or any character other than + - = or a hex number.) Exits the command and returns to the debugger prompt.

If ever an attempt is made to access an invalid virtual address, the command and all levels of invoked macros will be aborted and the debugger will prompt for the next command.

Display Commands

These commands allow you to examine memory only. This prevents accidental modification of system memory when in the debugger.

`dw` *addr* [*count*]
Displays memory as words (4 byte hex integers), 32 bytes at a time. If a *count* is given, memory is displayed 32 * *count* bytes at a time.

`dh` *addr* [*count*]
Displays memory as half words (2 byte hex integers), 32 bytes at a time.

`db` *addr* [*count*]
Displays memory as bytes (1 byte hex integers), 32 bytes at a time.

`di` [*addr*]
Displays memory as disassembled instructions. The default *addr* is the contents of `SXIP` on the 88K platform and the contents of `PC` on the 68K platform.

`dr` [*addr*]
Displays the CPU general registers stored at *addr*. The default *addr* is the automatically-determined register save area (see the `rg` command).

`dR` Displays the CPU "special" registers (shadow, control, supervisor, data pipe-line, cache control, and root pointers)

`df` Displays the CPU floating point registers.

`dy` *addr* [*count*]
Similar to `dw`, but displays the longs in symbolic form, if possible.

`se` *start end pattern* [*mask*]
`search` *start end pattern* [*mask*] Searches for the given pattern in the range of addresses starting at *start*, up to (but not including) *end*. The search is performed on words (4 bytes). If a *mask* is given, only those bits corresponding to 1 bits in the mask are significant in the search.

Examine Commands

`ew` *addr*
Examines memory as words, one at a time.

`eh` *addr*
Examines memory as half words, one at a time.

`eb` *addr*
Examines memory as bytes, one at a time.

`ei` [*addr*]
Examines memory as disassembled instructions. (Same as `di`.)

`er` Examines CPU general registers, one at a time.

`eR` Examines the CPU special registers, one at a time.

`ef` Examines the floating point registers, one at a time.

Modify Commands

`mw` *addr*
Examines and optionally modifies memory, as words.

`mh` *addr*
Examines and optionally modifies memory, as half words.

`mb` *addr*
Examines and optionally modifies memory, as bytes.

`mi` [*addr*]
Examines memory as instructions and optionally modifies (as bytes).

`mr` Examines and optionally modifies the CPU registers.

`mR` Examines and optionally modifies the CPU special registers.

`mf` Examines and optionally modifies the floating point registers.

Write Commands

`ww` *addr*
Writes memory as words, without examining.

`wh` *addr*
Writes memory as half words, without examining.

`wb` *addr*
Writes memory as bytes, without examining.

Execute Commands

`go` [*addr*]
Resumes execution on all processors. If an *addr* is given, the master processor resumes execution at *addr*.

`gos` [*cpuid* ...]
Resumes execution on only the processors whose *cpuid*'s are listed. If no *cpuid*'s are given, it resumes execution on only the current master processor.

`tr` [*addr*]
Trace: single-step one instruction on the master processor. If an *addr* is given, the master processor resumes execution at *addr*.

`trs` [*cpuid*]
Trace (single step) on the slave processor identified by *cpuid*, or on the master processor if *cpuid* is omitted.

`to` [*addr*]
Trace over: single step over "call" instructions on the master processor. If an *addr* is given, the master processor resumes execution at *addr*.

`tos` [*cpuid*]
Trace (single step) over "call" instructions on the slave processor identified by *cpuid*, or on the master processor if *cpuid* is omitted.

`stop` [*cpuid* ...]
Suspend execution on the running processors whose *cpuid*'s are listed, and force them into slave mode. If no *cpuid*'s are given, it suspends every currently running processor.

`call` *addr* [`args` ...]
Call a function with the specified arguments and show the return value.

Instruction Breakpoint Commands

An instruction breakpoint invokes the debugger just prior to the execution of a specified instruction. There are a total of sixteen instruction breakpoints available. Instruction breakpoints affect all processors; that is, every processor that hits an instruction breakpoint will enter the debugger.

`br` [*addr*]
Sets an instruction breakpoint. The default address is the contents of `SXIP`.

`bc` [*addr*]
Clears (removes) an instruction breakpoint.

`bC` Clears (removes) all instruction breakpoints.

`bx` [*addr*]
Sets a temporary (one-shot) instruction breakpoint.

`bo` [*addr*]
Turns an instruction breakpoint on or off. If a breakpoint is turned off, it acts as though it were cleared, but the breakpoint remains in the breakpoint table.

`bp` Displays instruction breakpoints.

Data Breakpoint Commands (88K only)

A data breakpoint invokes the debugger when a specified memory location is modified. There are a total of sixteen data breakpoints available.

`ur` *addr1* [*addr2*]
Sets a data breakpoint at *addr1* through *addr2*. The default for *addr2* is *addr1*.

`uc` *num*
Clears a data breakpoint.

`uC` Clears all data breakpoints.

`ux` *addr1* [*addr2*]
Like `ur`, but sets a temporary (one-shot) breakpoint.

`up` Prints all data breakpoints.

Miscellaneous Commands

`cpu` *cpuid*
Switch master mode from the current master processor to the slave processor identified by *cpuid*. The current master processor becomes a slave, and the designated slave becomes the new master.

`ss` [*cpuid*]
Show the multi-processor debugger status of the processor identified by *cpuid*, or of all processors if *cpuid* is omitted.

`bt` [*pc*] [*sp*] on the 88K platform
Displays a stack backtrace, using *sp* as the stack pointer and *pc* as the program counter. The default stack pointer is the contents of register `r31`. The default program counter is the contents of the `SXIP` register. If *pc* and *sp* are specified, the return address for the first routine must be already saved on the stack.

`bt` [*fp*] on the 68K platform
Displays a stack backtrace, using *fp* as the frame pointer. The default frame pointer is the contents of the fregister `fp`. If the *fp* is specified, the return address for the first routine must be already saved on the stack.

`fill` *start end value*
Fills memory from address *start* up to (but not including) address *end* with the byte *value*.

`pf` "*string*" [`args` ...]
`printf` "*string*" [`args` ...]
Prints the string. Percent signs in the string are treated as in `printf`(3S): `%d`, `%u`, `%x`, `%o`, `%b`, `%s`, `%c` are supported. In addition, `%y` prints its argument in symbolic form, if possible, and `%I` prints its argument in disassembled instruction form.

`rg` [*addr*]
Changes the pointer to the "register save area," from which all references to CPU registers retrieve registers. Normally, the register save area is set up automatically, but you can use a different set of registers when you use `rg` to change the pointer. To restore the pointer to its original value, use an *addr* of zero. If *addr* is missing, it displays the current register save area pointer.

`pg` [*n*] [*s*]
If *n* is 0, turns paging off. If *n* is 1, turns paging on. If *n* is missing, it simply reports whether paging is on or off. If paging is off, the debugger interprets all addresses as physical addresses. If paging is on, addresses are interpreted as linear (virtual) addresses. (Breakpoints are always linear addresses.) If *s* is present, the command is silent.

`su` [*n*]
If *n* is 0, turns user area addressing off. If *n* is 1, turns user area addressing on. If *n* is missing, it simply reports whether user area addressing is on or off. If user area addressing is off, the debugger interprets all addresses as supervisor area addresses. If user area addressing is on, addresses are user area addresses.

`ma` *addr*
`map` *addr*
Displays the page directory and page table entries used to map the given linear address to a physical address. This behaves the same whether paging is on or off.

`root` [*addr*]
Uses the specified supervisor area descriptor for translating linear-to-physical addresses. This address is obtained from the `MMU` if no `root` command is given. To restore the base to that original value, use an *addr* of zero. If *addr* is missing, it displays the current supervisor area descriptor.

`sp` [*addr*]
addr must be the address of a kernel `proc` structure. The debugger uses the context of that process to translate linear-to-physical addresses for the u page and user area addresses. Use of the `sp` command overrides the MMU.

`radix` [*n*]
Set the input radix to *n*. If *n* is omitted, it displays the current input radix. The default radix is hexadecimal.

`pr` *addr* [*radix*]
Prints the value of the address given as an argument in the specified radix, or in the current input radix if *radix* is missing. This is most useful if *addr* is an expression (see the earlier discussion of arguments).

`printbits` *"bit-desc" word*
Display the bits that are set (the 1 bits) in *word* symbolically according to *bit-desc*, which is a colon-separated list of names associated with the corresponding bit positions, starting with bit 0 (the least significant bit). For example, `printbits ''X:Y:Z:FOO:BAR'' 0x9D` prints `X Y Z FOO BAR`.

`ds` *addr*
Prints the value of the address as an offset from the nearest symbol.

`sy` [*n*] [*max*]
If *n* is 0, turns symbolic display off. If *n* is 1, turns symbolic display on. If *n* is missing, it simply reports whether symbolic display is on or off. If *max* is given, it specifies the maximum offset for printing symbols. For example, if *max* is 0x2000, a symbol may be displayed in the form *name* + *NNN*, where *NNN* is 1 through 0x2000, but if *NNN* would be greater than 0x2000, the non-symbolic display format is used.

`more` [*lines*]
Sets the number of display lines to *lines*. If *lines* is greater than zero, it enables output paging. When *lines* or more contiguous lines of information are printed without asking the user for input, the message `--press space for more--` is displayed and output is temporarily suspended until the user presses the space bar. This prevents the debugger from printing too many lines of output at once on video terminals. If *lines* is zero, it disables output paging. Output paging is disabled by default. If *lines* is missing, it reports whether output paging is enabled or disabled.

`pause` Pause until the user types something.

`he` or `help` or `?` or `??`
Lists the debugger commands.

`ve` Prints the version number of the debugger.

`#` or `##` or `no`
No-op. Input lines beginning with # are treated as comments and ignored.

Debugger Variable Commands

These commands manipulate variables the user defines. User-defined debugger variables may be used in expressions just like kernel symbols.

`set` *var value*
Set the variable named by *var* to have the given *value*. If the variable *var* has not previously been defined, it becomes defined; if it was previously defined, its old value is lost.

`read` *var*
Read an expression from the user and set the named debugger variable to the expression's value.

Macro Commands

`define` ''*name*'' [*arg-desc*] [*maxsize*]
Defines a macro, with the given name and the specified argument description string. The optional *maxsize* argument specifies the maximum size (in bytes) of the macro; the default size is 4096 bytes. The macro can be invoked after its definition by simply typing its name like any other command. The *arg-desc* string describes to the debugger what arguments the macro expects: each lower-case letter specifies the type of the corresponding argument, as follows:

`a` or `i`	address or integer, the result of an arbitrary expression
`s`	string
`?`	means the following arguments are optional
`*`	means any number of arguments or any type
`.`	means don't parse more arguments
`,`	is ignored.

For example, the argument description for the `se` command is "`aai?i`", and for `pf` it is "`s*`". If *arg-desc* is missing, the macro will be defined as requiring no arguments. After you enter the `dm` command, the debugger prompts with `mac>` for the body of the macro. Any debugger commands can be entered as the body of the macro, although interactive commands, such as `di` are not recommended (see the `interact` command). The expression `$`*n*, where *n* is a digit from 1 to 9, is replaced on invocation with the *n*th argument to the macro. The expression `$#` evaluates to the number of arguments to the macro. Entry of the macro body is terminated by a period (.) anywhere in the macro body. Include a period in the macro body by preceding the character with a backslash (\).

`dm` ''*name*'' *arg-count* [*maxsize*]
[Note: The `dm` command is obsolete; use `define` instead.] Defines a macro, with the given name and the specified number of arguments. The optional *maxsize* argument specifies the maximum size (in bytes) of the macro; the default size is 4096 bytes. The `cm` command (described below) can be used subsequently to invoke the macro.

`cm ''`*name*'' [`args` ...]
[Note: The `cm` command is obsolete; instead simply type the name of the macro as if it were a normal debugger command.] Calls a macro. The number of arguments must match the number given when the macro was defined. The effect is as if the body of the macro were entered in place of the `cm` command.

`em ''`*name*''
`delm ''`*name*''
Erases (deletes) the named macro.

`lm` [''*name*'']
Lists the named macro. If the macro name is omitted, lists all macros.

`nx` Repeats the call to the previously invoked macro. The arguments used are those used on the previous call, possibly modified by any intervening `sa` commands.

`sa` *n value*
`setarg` *n value*
Sets the *n*th macro argument to the given value. The value of *n* should be between 1 and 9. Useful within a macro to set up the arguments for the next call through an `nx` command.

`args` *n*
Sets the number of macro arguments to *n*.

`ec` [*n*]
`echo` [*n*]
If *n* is 1, macros are echoed when they are invoked. If *n* is 0 (the default), macros are not echoed. If *n* is missing, the status of the echo flag is printed. If the `ec` command is given within a macro body, it is in effect for that macro only.

`interact` *n*
If *n* is 1, interactive commands (such as `di`, `bt`, and `dr`) when invoked during macro execution will read input from the user. If *n* is 0 (the default), interactive commands inside macros will read input from the macro body. The `interact` command affects only the currently-executing macro and has no effect outside a macro body.

`onbreak` [''*name*'']
Set the on-break macro to the macro named by *name*. If *name* is omitted, it disables the on-break macro feature. The on-break macro, if one is specified, is executed on every entrance to the debugger resulting from any trap or breakpoint. This feature is very handy for implementing conditional breakpoints.

`do ''`*name*'' [`args` ...]
Repeatedly call the named macro with any `args` specified, until an `exit` command is executed. This is the only explicit form of iteration the debugger provides. The `args` are passed to the first — and, if `setarg` is not used, to every — iteration.

`exit` Stop iterating a repeated macro call (see `do`). Note that `exit` does not terminate the execution of the current macro; it merely prevents further iterations.

Predefined Macros

`buf` *addr* Print selected fields of a `struct buf` at the given address.

`buf+` *addr* Run `buf` *addr* and set up the debugger to display the next adjacent buffer.

`buf-` *addr* Run `buf` *addr* and set up the debugger to display the previous adjacent buffer.

`bufv` *addr* Run `buf` *addr* and set up the kernel debugger to display the buffer at `addr->av_forw` each time a carriage return is entered.

`dataunit` Prints information in the data pipeline.

`dmt` Prints information in the data unit.

`xintrq` *addr*
Print all the inter-CPU interrupt queues.

`xintrq+` *addr*
Run `xintrq` *addr* and set up the kernel debugger to print the next adjacent inter-CPU interrupt queue.

`inode` *addr* Print selected fields of a `struct inode` (in the same manner as `sys/inode.h`).

`inode+` *addr*
Run `inode` *addr* and set the debugger up to print the next adjacent inode address.

`mutex` *addr* Print selected fields of a `struct mutex`.

`curlock` Print the master processor's `curlock` stack.

`percpu` *addr*
Print selected fields of a `struct percpu`.

`percpup+` *addr*
Run `percpu` **addr* and set the debugger up to display the next adjacent cpu address.

`cpuinfo` *cpuid*
Print selected fields of a `struct cpuinfo` for the given CPU.

`proc` *addr* Print selected fields of a `struct proc`.

`proc+` *addr* Run `proc` *addr* and set up the debugger to display the next adjacent processor address.

`ps` Simulate `/bin/ps -l`.

`pss` Similar to `ps`, but prints process size and rss.

`psr` *addr* Prints the processor status register in the short form.

`psrl` *addr* Prints the processor status register in the long form.

`ptdat` *addr* Print the `pagetable data` data structure.

`pid` *pid* Find a process with the given *pid* (remember, the default debugger radix is hex, not decimal) and run `proc` on it.

`btproc` *addr*
Set the KDB process context to *addr* and run the backtrace `bt` command.

`reboot` Reboots the system. If the system does not reboot, this macro prints `"The system did not reset!"`

`strstat` Prints selected STREAMS statistics.

`strmsg` *addr*
Print selected fields of a `struct msgb`.

`strqueue` *addr*
Print selected fields of a `struct queue`.

`strqueue_band` *addr*
Print selected fields of a `struct qband`.

`stream` *addr*
Print selected fields of a `struct stdata` and substructures.

`streams` Print all `streams` except `muxs`.

`streams_muxs`
Print all `streams`.

`queues` Print all `streams` queues.

`queues_flag`
Print all `streams` queues with none of the flags set.

`stream_find` *addr*
Find and print a `stream` associated with the given `queue`.

`dumpnpi` *addr*
Interprets and displays the appropriate contents of the a `N_primitive` union as a Network Provider Interface message. *addr* should be the address of the message block.

`dumptpi` *addr*
Interprets and displays the appropriate contents of the a `T_primitive` union as a Transport Provider Interface message. *addr* should be the address of the message block.

`dumpque` *addr*
Dumps out the entire `queue` structure in the order defined in the structure.

`dumpmsgb` *addr*
Dumps out the entire `msgb` structure in the order defined in the structure.

`dumpdatab` *addr*
Dumps out the `datab` structure and the data from `db_base` to the end of data as a hex and ascii dump.

`findmsgs` Prints out all the queues that have message blocks attached to them.

`findsched` Prints out all the queues who's `q_link` field is non-zero. If the `q_link` is non-zero it may indicate that the queues is on the streams schedular list.

`tty` *addr* Print selected fields of a `struct tty`.

`tty+` *addr* Run `tty` *addr* and set the debugger up to run `tty` on the next adjacent address.

`user` *addr* Print selected fields of a `struct user`.

`vnode` *addr* Print selected fields of a `struct vnode`.

`vnode+` *addr*
Run `vnode` *addr* and set up the debugger to run `vnode` on the next adjacent address.

`memregion` *addr*
Print the content of a memory region structure. If *addr* is omitted, `memregion` prints the content of the first memory region.

`memregion+` *addr*
Run `memregion` *addr*, and set up the debugger to run `memregion` on the next memory region.

The following debugger command names can be used to create macro names for both user area access and for physical memory access. To be used for user area access, the names of these macros are created by adding a "u" to the end of the debugger command name. (For example, `ewu` *addr*.) To be used for physical memory access, the names of these macros are created by adding a "p" to the end of the debugger command name. (For example, `ewp` *addr*.)

`ew` *addr* Examine word.

`eh` *addr* Examine half word.

`eb` *addr* Examine byte.

`mw` *addr* Modify word.

`mh` *addr* Modify half word.

`mb` *addr* Modify byte.

`ww` *addr* Write word.

`wh` *addr* Write half word.

`wb` *addr* Write byte.

`map` *addr* Map address.

`fill` *aai* Fill memory.

`di` *a?i* Disassemble.

`ei` *a?i* Examine instructions.

`mi` *a?i* Modify instructions.

`dw` *a?i* Display word.

`dh` *a?i* Display half word.

`db` *a?i* Display byte.

`dy` *a?i* Display symbolic.

`se` *a?i* Search memory.

The following macros are used for virtual memory access.

`anon` *a?i*
`anon+` *addr*
`anoninfo` *a?i*
`anoninfo+` *addr*
`anon_map` *a?i*
`anon_map+` *addr*
Display information about the anonymous memory structures.

`page` *a?i*
`page` *addr* Display the page structure in a readable format. (`pagen` *addr* follows the `p_next` member, `pagev` *addr* follows the `p_vpnext` member, and `pageh` *addr* follows the `p_hash` member.)
`page+` *a?i*
`page+` *addr* Run *page addr* and set the debugger up to display the next adjacent inode address.
`page_map` *a?i*
`page_map` *addr*
Print the page structure along with its `pmapping` structure.
`page_pt` *a?i*
`page_pt` *addr*
Print a page structure and `ptdat` structure. The *addr* must point to a page that is a page table.
`pmapping` *a?i*
`pmapping` *addr*
Print the `pte`s that are in a mapping structure.
`pte` *a?i*
`pte+` *addr* Display information about a page table entry given a virtual address.
`physpte` *a?i*
`physpte+` *phys addr*
Display information about a page table entry given a physical address.
`ste` *a?i*
`ste+` *addr* Display information about segment table entries.
`area` *a?i*
`area+` *addr* Display the area pointer.
`batc` *a?i*

batc+ *addr* Display information about BATC entries.

hat *a?i* Display the contents of the hat structure provided as an argument.

ptdat *a?i*
ptdat+ *addr*
Display the contents of the ptdat structure.

as *a?i* Display information about an address space structure.

seg_ops *a?i*
Display information about a segment operations structure.

seg *a?i*
seg+ *addr* Display information about a segment.

segdev_crargs *a?i*
segdev_crargs+ *addr*
segdev_data *a?i*
segdev_data+ *addr*
Display information about selected segdev data structures.

segmap_crargs *a?i*
segmap_crargs+ *addr*
smap *a?i*
smap+ *addr*
segmap_data *a?i*
segmap_data+ *addr*
Display information about selected segmap data structures.

segu_segdata *a?i*
segu_segdata+ *addr*
segu_data *a?i*
segu_data+ *addr*
Display information about selected segu data structures.

segvn_crargs *a?i*
segvn_crargs+ *addr*
segvn_data *a?i*
segvn_data+ *addr*
Display information about selected segvn data structures.

vpage *a?i*
vpage+ *addr*
Display information about pages associated with a VNODE.

Conditional Commands

IF *expr*
EL
FI If the expression evaluates to zero, all commands up to the matching EL or FI are skipped. If the expression is non-zero, execution proceeds normally to the matching FI, unless a matching EL is found, in which case, commands between the EL and the FI are skipped. During any of this "skipping," the prompt changes from K*n*> to -K*n*> to indicate that the commands are being read but not executed.

`if` *expr*
`elseif` *expr*
`else`
`fi` Same as `IF-EL-FI` but with the `elseif` construct which allows chaining of conditional statements.

`ifdef` "*name*"
Like `if`, but the condition is "true" if a macro named *name* exists.

`ifsdef` "*name*"
Like `if`, but the condition is "true" if a symbol named *name* exists.

NOTES

The arguments displayed by the `bt` (backtrace) command for each function in the backtrace may not be correct. The number of arguments displayed may be greater than the actual number of argument to the routine.

Since the 68K has a variable sized instruction set, the heuristic used to back up instructions may disassemble the instructions incorrectly or may fail to disassemble the instructions.

The debugger does not prevent the user from setting data or instruction breakpoints or tracing routines the debugger itself requires for its normal operation. If breakpoints or tracing occur in these support routines or regions of memory, the debugger will hang the system.

Displaying memory with paging turned off or in I/O regions (e.g. 0xE0000000 - 0xFFFFFFFF) will cause a bus error panic, if the addresses given are not valid physical addresses.

FILES

`/usr/lib/kdb/macros` Directory containing macro files.

NAME

keylogin - decrypt and store secret key

SYNOPSIS

keylogin

DESCRIPTION

The keylogin command prompts for a password, and uses it to decrypt the user's secret key stored in the publickey(4) database. Once decrypted, the user's key is stored by the local key server process, keyserv(1M), to be used by any secure network service, such as NFS.

SEE ALSO

chkey(1), keylogout(1), publickey(4), keyserv(1M), newkey(1)

NAME

keyserv - server for storing public and private keys

SYNOPSIS

keyserv [-n] [-d]

DESCRIPTION

keyserv is a daemon that is used for storing the private encryption keys of each user logged into the system. These encryption keys are used for accessing secure network services such as secure NFS.

Normally, root's key is read from the file /etc/.rootkey when the daemon is started. This is useful during power-fail reboots when no one is around to type a password.

When the -n option is used, root's key is not read from /etc/.rootkey. Instead, keyserv prompts the user for the password to decrypt root's key stored in the publickey(4) database and then stores the decrypted key in /etc/.rootkey for future use. This option is useful if the /etc/.rootkey file ever gets out of date or corrupted.

To prohibit the nobody key or any other default keys, use the -d option.

FILES

/etc/.rootkey

SEE ALSO

publickey(4)

NAME

kill - terminate a process by default

SYNOPSIS

```
kill [-signal] pid...
kill -signal -pgid...
kill -l
```

DESCRIPTION

kill sends a signal to the specified processes. The value of signal may be numeric or symbolic [see signal(5)]. The symbolic signal name is the name as it appears in /usr/include/sys/signal.h, with the SIG prefix stripped off. Signal 15 (SIGTERM) is sent by default; this will normally kill processes that do not catch or ignore the signal.

pid and *pgid* are unsigned numeric strings that identify which process(es) should receive the signal. If *pid* is used, the process with process ID *pid* is selected. If *pgid* is used, all processes with process group ID *pgid* are selected.

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).

When invoked with the -l option, kill will print a list of symbolic signal names. 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 signaled process must belong to the current user unless the user is the super-user.

SEE ALSO

ps(1), sh(1), kill(2), signal(2), signal(5).

NAME

killall - kill all active processes

SYNOPSIS

/usr/sbin/killall [*signal*]

DESCRIPTION

killall is used by /usr/sbin/shutdown to kill all active processes not directly related to the shutdown procedure.

killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

killall sends *signal* [see kill(1)] to all processes not belonging to the above group of exclusions. If no *signal* is specified, a default of 9 is used.

FILES

/usr/sbin/shutdown

SEE ALSO

fuser(1M), shutdown(1M), kill(1), ps(1), signal(2).

NAME

ksh, rksh - KornShell, a standard/restricted command and programming language

SYNOPSIS

ksh [±aefhikmnoprstuvx] [±o *option*] ... [-c *string*] [*arg* ...]
rksh [±aefhikmnoprstuvx] [±o *option*] ... [-c *string*] [*arg* ...]

DESCRIPTION

ksh is a command and programming language that executes commands read from a terminal or a file. rksh is a restricted version of the command interpreter ksh; 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 *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 *functions* and *variables*. A *word* is a sequence of *characters* separated by one or more non-quoted *metacharacters*.

A *command* is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A special command is a command that is carried out by the shell without creating a separate process. Except for documented side effects, most special commands can be implemented as separate utilities.

Commands

A *simple-command* is a sequence of *blank* separated words which may be preceded by a variable 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(2) 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(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 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. 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 may appear in a *list*, instead of a semicolon, to delimit a command.

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 a `for` command is executed, *identifier* 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.

`select` *identifier* [`in` *word* ...] ; `do` *list* ; `done`
: A `select` command prints on standard error (file descriptor 2), the set of *word*s, each preceded by a number. If `in` *word* ... is omitted, then 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 *word*s, then 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 variable `REPLY`. The *list* is executed for each selection until a `break` or *end-of-file* is encountered. If the `REPLY` variable is set to `null` by the execution of *list*, then the selection list is printed before displaying the `PS3` prompt for the next selection.

`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 same as that used for filename 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, then the `if` command 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, 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 separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{ *list* ; }
: *list* is simply executed. The { must be followed by a space. Note that unlike the metacharacters (and), { and } are *reserved words* and must occur at the beginning of a line or after a ; in order to be recognized.

[[*expression*]]
: Evaluates *expression* and returns a zero exit status when *expression* is true. See *Conditional Expressions* below, for a description of *expression*.

function *identifier* { *list* ; }
identifier () { *list* ; }
: Define a function which is referenced 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 as well as the user and system time are printed on standard error.

The following reserved 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   {   }   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. An alias name consists of any number of characters excluding meta-characters, quoting characters, file expansion characters, parameter and command substitution characters and =. The replacement string can contain any valid Shell script including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for aliases. If the last character of the alias value is a *blank* then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the reserved words 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 scripts invoked by name, 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 an alias to take effect the alias definition command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the corresponding command is looked up and becomes undefined each time the PATH variable is reset. These aliases remain *tracked* so that the next subsequent reference will redefine the value. Several tracked aliases are compiled into the shell. The -h option of the set command makes each referenced command name into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, then 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 is 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 ~ by itself, or in front of a /, is replaced by $HOME. A ~ followed by a + or - is replaced by $PWD and $OLDPWD respectively.

In addition, *tilde* substitution is attempted when the value of a *variable assignment* begins with a ~.

Command Substitution

The standard output from a command enclosed in parentheses 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.

An arithmetic expression enclosed in double parentheses and preceded by a dollar sign [$(())] is replaced by the value of the arithmetic expression within the double parentheses.

Process Substitution

This feature is only available on versions of the UNIX operating system that support the /dev/fd directory for naming open files. Each command argument of the form <(*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 then writing on this file will provide input for *list*. If < is used, then the file passed as an argument will contain the output of the *list* process.

For example,

```
paste <(cut -f1 file1) <(cut -f4 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. Note that the file, which is passed as an argument to the command, is a UNIX *pipe*(2) so programs that expect to *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 *variable* (a parameter denoted by an identifier) has a *value* and zero or more *attributes*. *Variables* can be assigned `values` and *attributes* by using the `typeset` special command. The attributes supported by the Shell are described later with the `typeset` special command. Exported parameters pass values and attributes to the environment.

The shell supports a one-dimensional array facility. An element of an array variable is referenced by a *subscript*. A *subscript* is denoted by a `[`, followed by an *arithmetic expression* (see *Arithmetic Evaluation* below) followed by a `]`. To assign values to an array, use `set -A` *name value* .... The value of all subscripts must be in the range of 0 through 1023. Arrays need not be declared. Any reference to a variable with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the element zero.

The *value* of a *variable* may also be assigned by writing:

name=value [*name=value*] ...

If the integer attribute, `-i`, 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*`}`

The shell reads all the characters from `${` to the matching `}` as part of the same word even if it contains braces or metacharacters. The value, if any, of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a variable is subscripted. If *parameter* is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is `*` or `@`, then all the positional parameters, starting with `$1`, are substituted (separated by a field separator character). If an array *identifier* with subscript `*` or `@` is used, then the value for each of the elements 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*[*]}
The number of elements in the array *identifier* is substituted.

${*parameter*:-*word*}
If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null then 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 then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter*:+*word*}
If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

${*parameter*#*pattern*}
${*parameter*##*pattern*}
If the Shell *pattern* matches the beginning of the value of *parameter*, then 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 first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted. The result is unspecified when *parameter* is @, *, or an array variable with subscript @, or *.

${*parameter*%*pattern*}
${*parameter*%%*pattern*}
If the Shell *pattern* matches the end of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched part deleted; otherwise substitute the value of *parameter*. In the first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted. The result is unspecified when *parameter* is @, *, or an array variable with subscript @, or *.

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 executed command.
$	The process number of this shell.
_	Initially, the value _ is an absolute pathname of the shell or script being executed as passed in the *environment*. Subsequently it is assigned 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.

! The process number of the last background command invoked.

ERRNO The value of errno as set by the most recently failed system call. This value is system dependent and is intended for debugging purposes.

LINENO The line number of the current line within the script or function being executed.

OLDPWD The previous working directory set by the cd command.

OPTARG The value of the last option argument processed by the getopts special command.

OPTIND The index of the last option argument processed by the getopts special command.

PPID The process number of the parent of the shell.

PWD The present working directory set by the cd command.

RANDOM Each time this variable is referenced, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a numeric value to RANDOM.

REPLY This variable is set by the select statement and by the read special command when no arguments are supplied.

SECONDS Each time this variable is referenced, the number of seconds since shell invocation is returned. If this variable is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following variables are used by the shell:

CDPATH The search path for the cd command.

COLUMNS If this variable is set, the 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, then the corresponding option (see *Special Command* set below) will be turned on.

ENV If this variable is set, then parameter substitution is performed on the value to generate the pathname of the script that will 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.

FPATH The search path for function definitions. By default the FPATH directories are searched after the PATH variable. If an executable file is found, then it is read and executed in the current environment. FPATH is searched before PATH when a function with the -u attribute is referenced. The preset alias autoload preset alias causes a function with the -u attribute to be created.

IFS Internal field separators, normally space, tab, and new-line that is used to separate command words which result from command or parameter substitution and for separating words with the special command read. The first character of the IFS variable is used to separate arguments for the "$*"

substitution (see *Quoting* below).

HISTFILE If this variable is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history (see *Command re-entry* below).

HISTSIZE If this variable is set when the shell is invoked, then the number of previously entered commands that are accessible by 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 will print vertically until about two-thirds of `LINES` lines are filled.

MAIL If this variable is set to the name of a mail file *and* the `MAILPATH` variable is not set, then the shell informs the user of arrival of mail in the specified file.

MAILCHECK This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the `MAILPATH` or `MAIL` variables. The default value is 600 seconds. When the time has elapsed the shell will check before issuing the next prompt.

MAILPATH A colon (`:`) separated list of file names. If this variable is set then 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 that will be printed. The message will undergo parameter substitution with the variable, `$_` defined as the name of the file that has changed. 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 under `rksh` (except in *.profile*).

PS1 The value of this variable is expanded for parameter substitution to define the primary prompt string which by default is "`$ `". The character `!` in the primary prompt string is replaced by the *command* number (see *Command Re-entry* below). Two successive occurrences of `!` will produce a single `!` when the prompt string is printed.

PS2 Secondary prompt string, by default "`> `".

PS3 Selection prompt string used within a `select` loop, by default "`#? `".

PS4 The value of this variable is expanded for parameter substitution and precedes each line of an execution trace. If omitted, the execution trace prompt is "`+ `".

SHELL The pathname of the *shell* is kept in the environment. At invocation, if the basename of this variable is `rsh`, `rksh`, or `krsh`, then the shell becomes restricted.

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 If the value of this variable ends in emacs, gmacs, or vi then the corresponding option (see *Special Command* set below) will be turned on.

The shell gives default values to PATH, PS1, PS2, PS3, PS4, MAILCHECK, TMOUT and IFS. HOME, MAIL and SHELL are set by login(1).

Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the 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 scanned for the characters *, ?, and [unless the -f option has been set. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with lexicographically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a *pattern* is used for file name generation, the character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly. In other instances of pattern matching the / and . are not treated specially.

* Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" then any character not enclosed is matched. A - can be included in the character set by putting it as the first or last character.

A *pattern-list* is a list of one or more patterns separated from each other with a |. Composite patterns can be formed with one or more of the following:

?(*pattern-list*)
Optionally matches any one of the given patterns.

*(*pattern-list*)
Matches zero or more occurrences of the given patterns.

+(*pattern-list*)
Matches one or more occurrences of the given patterns.

@(*pattern-list*)
Matches exactly one of the given patterns.

!(*pattern-list*)
Matches anything, except one of the given patterns.

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* (that is, made to stand for itself) by preceding it with a \. The pair \new-line is removed. 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 variable assignment value or as a file name. However,

when used as a command argument, "$*" is equivalent to "$1*d* $2*d*...", where *d* is the first character of the IFS variable, whereas "$@" is equivalent to "$1"*d*"$2"*d*. . . Inside grave quote marks (` `) \ quotes the characters \, `, and $. If the grave quotes occur within double quotes then \ also quotes the character ".

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command let. Evaluations are performed using *long* arithmetic. Constants are of 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 then base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression of the C language. All the integral operators, other than ++, - -, ?:, and , are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a *variable* can be specified with the -i option of the typeset special command. Arithmetic evaluation is performed on the value of each assignment to a variable with the -i attribute. If you do not specify an arithmetic base, the first assignment to the variable determines the arithmetic base. This base is 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 which begins with a ((, all the characters until a matching)) are treated as a quoted expression. More precisely, ((. . .)) is equivalent to let ". . .".

Prompting

When used interactively, the shell prompts with the parameter expanded 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 (that is, the value of PS2) is issued.

Conditional Expressions

A *conditional expression* is used with the [[compound command to test attributes of files and to compare strings. Word splitting and file name generation are not performed on the words between [[and]]. Each expression can be constructed from one or more of the following unary or binary expressions:

-a *file*	True, if *file* exists.
-b *file*	True, if *file* exists and is a block special file.
-c *file*	True, if *file* exists and is a character special file.
-d *file*	True, if *file* exists and is a directory.
-f *file*	True, if *file* exists and is an ordinary file.
-g *file*	True, if *file* exists and is has its setgid bit set.
-k *file*	True, if *file* exists and is has its sticky bit set.

`-n` *string*	True, if length of *string* is non-zero.
`-o` *option*	True, if option named *option* is on.
`-p` *file*	True, if *file* exists and is a fifo special file or a pipe.
`-r` *file*	True, if *file* exists and is readable by current process.
`-s` *file*	True, if *file* exists and has size greater than zero.
`-t` *fildes*	True, if file descriptor number *fildes* is open and associated with a terminal device.
`-u` *file*	True, if *file* exists and is has its setuid bit set.
`-w` *file*	True, if *file* exists and is writable by current process.
`-x` *file*	True, if *file* exists and is executable by current process. If *file* exists and is a directory, then the current process has permission to search in the directory.
`-z` *string*	True, if length of *string* is zero.
`-L` *file*	True, if *file* exists and is a symbolic link.
`-O` *file*	True, if *file* exists and is owned by the effective user id of this process.
`-G` *file*	True, if *file* exists and its group matches the effective group id of this process.
`-S` *file*	True, if *file* exists and is a socket.
file1 `-nt` *file2*	True, if *file1* exists and is newer than *file2*.
file1 `-ot` *file2*	True, if *file1* exists and is older than *file2*.
file1 `-ef` *file2*	True, if *file1* and *file2* exist and refer to the same file.
string `=` *pattern*	True, if *string* matches *pattern*.
string `!=` *pattern*	True, if *string* does not match *pattern*.
string1 `<` *string2*	True, if *string1* comes before *string2* based on ASCII value of their characters.
string1 `>` *string2*	True, if *string1* comes after *string2* based on ASCII value of their characters.
exp1 `-eq` *exp2*	True, if *exp1* is equal to *exp2*.
exp1 `-ne` *exp2*	True, if *exp1* is not equal to *exp2*.
exp1 `-lt` *exp2*	True, if *exp1* is less than *exp2*.
exp1 `-gt` *exp2*	True, if *exp1* is greater than *exp2*.
exp1 `-le` *exp2*	True, if *exp1* is less than or equal to *exp2*.
exp1 `-ge` *exp2*	True, if *exp1* is greater than or equal to *exp2*.

In each of the above expressions, if *file* is of the form `/dev/fd/`*n*, where *n* is an integer, then the test applied to the open file whose descriptor number is *n*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

`(`*expression*`)`
: True, if *expression* is true. Used to group expressions.

`!` *expression*
: True if *expression* is false.

expression1 `&&` *expression2*
: True, if *expression1* and *expression2* are both true.

expression1 `||` *expression2*
: True, if either *expression1* or *expression2* is true.

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. 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 then it is created. If the file exists, is a regular file, and the `noclobber` option is on, this causes an error; otherwise, it is truncated to zero length.
>|*word*	Sames as >, except that it overrides the `noclobber` option.
>>*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*	Open file *word* for reading and writing as standard input.
<<[-]*word*	The shell input is read up to a line that is the same as *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, then 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 <<, then all leading tabs are stripped from *word* and from the document.
<&*digit*	The standard input is duplicated from file descriptor *digit* [see `dup`(2)]. Similarly for the standard output using >& *digit*.
<&-	The standard input is closed. Similarly for the standard output using >&-.
<&p	The input from the co-process is moved to standard input.
>&p	The output to the co-process is moved to standard output.

If one of the above is preceded by a digit, then the file descriptor number referred to 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.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

... 1>*fname* 2>&1

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (that is, *fname*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by `&` and job control is not active, 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.

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 names must be *identifiers* and the values are character strings. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a variable for each name found, giving it the corresponding value and marking it *export* . Executed commands inherit the environment. If the user modifies the values of these variables or creates new ones, using the `export` or `typeset -x` commands they 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 variable assignments. A variable assignment argument is a word of the form *identifier=value*. Thus:

```
TERM=450 cmd args                    and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned except for commands listed with one or two daggers, †, in the Special Commands section).

If the `-k` flag is set, *all* variable assignment 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
```

This feature is intended for use with scripts written for early versions of the shell and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions

The `function` reserved word, described in the *Commands* section above, is used to define shell functions. Shell functions are read in 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 and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on `EXIT` set inside a function is executed after the function completes in the environment of the caller. 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` or `+f` option of the `typeset` special command. The text of functions may also be listed with `-f`. Functions 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 a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be specified in the `ENV` file with the `-xf` option of `typeset`.

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 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 hit the key `^Z` (CTRL-z) 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 `^Z` takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

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". If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. A job can be referred to by the process id of any process of the job or by one of the following:

`%`*number*	The job with the given number.
`%`*string*	Any job whose command line begins with *string*.
`%?`*string*	Any job whose command line contains *string*.
`%%`	Current job.
`%+`	Equivalent to `%%`.
`%-`	Previous job.

This shell 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 it prints a prompt. This is done so that it does not otherwise disturb your work.

When the monitor mode is on, each background job that completes triggers any trap set for CHLD.

When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped(running) 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 stopped jobs will be terminated.

Signals

When a command is run in the background (that it, when it is followed by &) and the job monitor option is active, the command does not receive INTERRUPT or QUIT signals. When a command is run in the background (that it, when it is followed by &) and the job monitor option is not active, the command receives INTERRUPT or QUIT signals but ignores them. Otherwise, signals have the values inherited by the shell from its parent (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 within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. When the *function* completes or issues a return, the positional parameter list is restored and any trap set on EXIT within the function is executed. 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*, a process is created and an attempt is made to execute the command via exec(2).

The shell variable PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is /usr/bin: (specifying /usr/bin and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an a.out file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and variables, are removed in this case. If the shell command file does not have read permission, or if the *setuid* and/or *setgid* bits are set on the file, then the shell executes an agent whose job it is to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is 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 file denoted by the HISTFILE variable is not set or if the file it names is not writable. A shell can access the commands of all *interactive* shells which 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 then the value of the variable FCEDIT is used. If FCEDIT is not defined then /usr/bin/ed is

used. The edited command(s) 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 variable of the form *old=new* can be used to modify the command before execution. For example, if `r` is aliased to `'fc -e -'` then typing `'r bad=good c'` will re-execute the most recent command which starts with the letter `c`, replacing the first occurrence of the string `bad` with the string `good`.

In-line Editing Options

Normally, each command line entered from a terminal device is simply typed followed by a new-line ('RETURN' or 'LINE FEED'). If the If either the `emacs`, `gmacs`, or `vi` option is active, the user can edit the command line. To be in either of these edit modes `set` the corresponding option. 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 series 2621 terminal users should set the straps to 'bcGHxZ etX'.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of `COLUMNS` if it is defined, otherwise 80. If the window width is too small to display the prompt and leave at least 8 columns to enter input, the prompt is truncated from the left. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > (<, *) if the line extends on the right (left, both) side(s) of the window.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading ^ in the string restricts the match to begin at the first character in the line.

Emacs Editing Mode

This mode is entered by enabling either the *emacs* or *gmacs* option. The only difference between these two modes is the way they handle `^T`. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the 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 CTRL-`f`. This is entered by depressing 'f' while holding down the 'CTRL' (control) key. (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 just 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 emacs 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 forward to character *char* on current line.
M-^]*char*	Move cursor back 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.
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) then 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 lower case.
^K	Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less than the current cursor position, then delete from given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from cursor up to given cursor position.
^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 command, usually ^G or @.) Kill the entire current line. If two kill characters are entered in succession, all kill characters from then on 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-space	(Meta space) Set mark.
^J	(New line) Execute the current line.
^M	(Return) Execute the current line.
eof	End-of-file character, normally ^D, is processed as an End-of-file only if the current line is null.
^P	Fetch previous command. Each time ^P is entered the previous command back in time is accessed. Moves back one line when not on the first line of a multi-line command.
M-<	Fetch the least recent (oldest) history line.
M->	Fetch the most recent (youngest) history line.

^N	Fetch next command line. Each time ^N is entered the next command line forward in time is accessed.
^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 preceded by a ^, the matched line must begin with *string*. If *string* is omitted, then 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*, ^C, ^D, ^K, ^R, ^P, ^N, ^], M-., M-^], M-_, M-b, M-c, M-d, M-f, M-h M-l 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 will be inserted on the input queue. The *letter* must not be one of the above meta-functions.
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 will be inserted on the input queue. The can be used to program functions keys on many terminals.
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. An asterisk is appended if the word doesn't match any file or contain any special pattern characters.
M-ESC	File name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.
M-=	List files matching current word pattern 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.
M-#	Insert a # at the beginning of the line and execute it. This causes a comment to be inserted in the history file.

vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the *input* mode. To edit, the user enters *control* mode by typing ESC (\033) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* prior to the command.

When in `vi` mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or 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.

If the option `viraw` is also set, the terminal will always have canonical processing disabled.

Input Edit Commands

By default the editor is in input mode.

erase	(User defined erase character as defined by the stty command, usually `^H` or `#`.) Delete previous character.
`^W`	Delete the previous blank separated word.
`^D`	Terminate the shell.
`^V`	Escape next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a `^V`. The `^V` removes the next character's editing features (if any).
`\`	Escape the next *erase* or `kill` character.

Motion Edit Commands

These commands will move the cursor.

[*count*]`l`	Cursor forward (right) one character.
[*count*]`w`	Cursor forward one alpha-numeric 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*]`|`	Cursor to column *count*.
[*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`.
[*count*]`;`	Repeats *count* times, the last single character find command, `f`, `F`, `t`, or `T`.
[*count*]`,`	Reverses the last single character find command *count* times.
`0`	Cursor to start of line.
`^`	Cursor to first non-blank character in line.

`$` Cursor to end of line.

`%` Moves to balancing `(`, `)`, `{`, `}`, `[`, or `]`. If cursor is not on one of the above characters, the remainder of the line is searched for the first occurrence of one of the above characters first.

Search Edit Commands

These commands access your command history.

[*count*]`k` Fetch previous command. Each time `k` is entered the previous command back in time is accessed.

[*count*]`-` Equivalent to `k`.

[*count*]`j` Fetch next command. Each time `j` is entered the next command forward in time is accessed.

[*count*]`+` Equivalent to `j`.

[*count*]`G` The command number *count* is fetched. The default is the least recent history command.

/string Search backward through history for a previous command containing *string*. *String* is terminated by a RETURN or NEW LINE. If string is preceded by a `^`, the matched line must begin with *string*. If *string* is null the previous string will be used.

`?`*string* Same as `/` except that search will be 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*

Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is `c`, the entire line will be deleted and input mode entered.

`C` Delete the current character through the end of line and enter input mode. Equivalent to `c$`.

[*count*]`s` Delete *count* characters and enter input mode.

`S` Equivalent to `cc`.

`D` Delete the current character through the end of line. Equivalent to `d$`.

[*count*]`d`*motion*

d[*count*]*motion*
: Delete current character through the character that *motion* would move to. 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 0i.

[*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.

[*count*]r*c*
: Replace the *count* character(s) starting at the current cursor position with *c*, and advance the cursor.

[*count*]x
: Delete current character.

[*count*]X
: Delete preceding character.

[*count*].
: Repeat the previous text modification command.

[*count*]~
: Invert the case of the *count* character(s) starting at the current cursor position and advance the cursor.

[*count*]_
: Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.

*
: Causes an * to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

\
: Filename completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

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, then 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.

If the first character of the command is a #, then this command deletes this # and each # that follows a new line. Otherwise, sends the line after inserting a # in front of the line. Useful for causing the current line to be inserted in the history as a comment and removing comments from previous comment commands in the history file.

= List the file names that match the current word if an asterisk were appended it.

@*letter* Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue for processing.

Special Commands

The following simple-commands are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is zero. Commands that are preceded by one or two † are treated specially in the following ways:

1. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a variable assignment, are expanded with the same rules as a variable assignment. This means that tilde substitution is performed after the = sign and word splitting and file name generation are not performed.

† : [*arg* ...]
The command only expands parameters.

† . *file* [*arg* . . .]
Read the complete *file* then execute the commands. 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. The exit status is the exit status of the last command executed.

†† 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 pathname corresponding to the given *name*. The value becomes undefined when the value of PATH is reset but the aliases remain 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 for scripts invoked by name. The exit status is non-zero if a *name* is given, but

no value, for which no alias has been defined.

`bg` [*job*...]
This command is only on systems that support job control. Puts each specified *job* into the background. The current job is put in the background if *job* is not specified. See *Jobs* for a description of the format of *job*.

† `break` [*n*]
Exit from the enclosing `for`, `while`, `until` or `select` loop, if any. If *n* is specified then break *n* levels.

† `continue` [*n*]
Resume the next iteration of the enclosing `for`, `while`, `until` or `select` loop. If *n* is specified then resume at the *n*-th enclosing loop.

`cd` [*arg*]
`cd` *old new*
This command can be in 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 variable `HOME` is the default *arg*. The variable `PWD` is set to the current directory. The shell variable `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 / then 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 may not be executed by `rksh`.

`echo` [*arg* ...]
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* ...]
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 that are opened with this mechanism are closed when invoking another program.

† `exit` [*n*]
Causes the shell to exit with the exit status specified by *n*. The value will be the least significant 8 bits of the specified status. If *n* is omitted then the exit status is that of the last command executed. When `exit` occurs when executing a trap, the last command refers to the command that executed before the trap was invoked. An end-of-file will also cause the shell to exit

except for a shell which has the *ignoreeof* option (see `set` below) turned on.

†† `export` [*name*[=*value*]] ...
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 that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag `-l`, is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the variable `FCEDIT` (default /usr/bin/ed) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified then it will be set to *first*. 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 second form the *command* is re-executed after the substitution *old*=*new* is performed.

`fg` [*job*...]
This command is only on systems that support job control. Each *job* specified is brought to the foreground. Otherwise, the current job is brought into the foreground. See *Jobs* for a description of the format of *job*.

`getopts` *optstring name* [*arg* ...]
Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a `+` or a `-`. An option not beginning with `+` or `-` or the argument `--` ends the options. *optstring* contains the letters that `getopts` recognizes. If a letter is followed by a `:`, that option is expected to have an argument. The options can be separated from the argument by blanks.

`getopts` places the next option letter it finds inside variable *name* each time it is invoked with a `+` prepended when *arg* begins with a `+`. The index of the next *arg* is stored in `OPTIND`. The option argument, if any, gets stored in `OPTARG`.

A leading `:` in *optstring* causes `getopts` to store the letter of an invalid option in `OPTARG`, and to set *name* to `?` for an unknown option and to `:` when a required option is missing. Otherwise, `getopts` prints an error message. The exit status is non-zero when there are no more options.

`jobs` [`-lnp`] [*job* ...]
Lists information about each given job; or all active jobs if *job* is omitted. The `-l` flag lists process ids in addition to the normal information. The `-n` flag only displays jobs that have stopped or exited since last notified. The `-p` flag causes only the process group to be listed. See *Jobs* for a description of the format of *job*.

`kill` [*-sig*] *job* ...
`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 prefix "SIG"). If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument *job* can be the process id of a process that is not a member of one of the active jobs. See *Jobs* for a description of the format of *job*. In the second form, `kill -l`, the signal numbers and names are listed.

`let` *arg* ...

Each *arg* is a separate *arithmetic expression* to be evaluated. See *Arithmetic Evaluation* above, for a description of arithmetic expression evaluation.

The exit status is 0 if the value of the last expression is non-zero, and 1 otherwise.

† `newgrp` [*arg* ...]

Equivalent to `exec /usr/bin/newgrp` *arg* ....

`print` [`-Rnprsu`[*n*]] [*arg* ...]

The shell output mechanism. With no flags or with flag `-` or `--` the arguments are printed on standard output as described by `echo`(1). In raw mode, `-R` or `-r`, the escape conventions of `echo` are ignored. 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. The exit status is 0 unless the output file is not open for writing.

`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 fields using the characters in `IFS` as separators. The escape character, `\`, is used to remove any special meaning for the next character and for line continuation. In raw mode, `-r`, a `\` at the end of a line does not signify line continuation. The first field is assigned to the first *name*, the second field to the second *name*, and so on, with leftover fields 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 then `REPLY` is used as the default *name.* The exit status is 0 unless the input file is not open for reading or an end-of-file is encountered. An end-of-file with the `-p` option causes cleanup for this process so that another can be spawned. If the first argument contains a

`?`, the remainder of this word is used as a *prompt* on standard error when the shell is interactive. The exit status is 0 unless an end-of-file is encountered.

†† `readonly` [*name*[`=`*value*]] ...
The given *names* are marked readonly and these names cannot be changed by subsequent assignment.

† `return` [*n*]
Causes a shell *function* or . script to return to the invoking script with the return status specified by *n*. The value will be the least significant 8 bits of the specified status. If *n* is omitted then the return status is that of the last command executed. If `return` is invoked while not in a *function* or a . script, then it is the same as an `exit`.

`set` [±`aefhkmnopstuvx`] [±`o` *option*]... [±`A` *name*] [*arg* ...]
The flags for this command have meaning as follows:

- `-A` Array assignment. Unset the variable *name* and assign values sequentially from the list *arg*. If `+A` is used, the variable *name* is not unset first.
- `-a` All subsequent variables that are defined are automatically exported.
- `-e` If a command has a non-zero exit status, execute the `ERR` trap, if set, and exit. This mode is disabled while reading profiles.
- `-f` Disables file name generation.
- `-h` Each command becomes a tracked alias when first encountered.
- `-k` All variable assignment arguments are placed in the environment for a command, not just those that precede the command name.
- `-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. On systems with job control, this flag is turned on automatically for interactive shells.
- `-n` Read commands and check them for syntax errors, but do not execute them. Ignored for interactive shells.
- `-o` The following argument can be one of the following option names:

`allexport`	Same as `-a`.
`errexit`	Same as `-e`.
`bgnice`	All background jobs are run at a lower priority. This is the default mode.
`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`.

noclobber Prevents redirection > from truncating existing files. Require >| to truncate a file when turned on.
noexec Same as -n.
noglob Same as -f.
nolog Do not save function definitions in history file.
nounset Same as -u.
privileged 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.

If no option name is supplied then the current option settings are printed.

-p Disables processing of the $HOME/.profile file and uses the file /etc/suid_profile instead of the ENV file. This mode is on whenever the effective uid (gid) is not equal to the real uid (gid). Turning this off causes the effective uid and gid to be set to the real uid and gid.
-s Sort the positional parameters lexicographically.
-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 then the positional parameters are unset.

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 $-. Unless -A is specified, the remaining arguments are positional parameters and are assigned, in order, to $1 $2 .... If no arguments are given then the names and values of all variables 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 $#.

† times Print the accumulated user and system times for the shell and for processes run from the shell.

† trap [*arg*] [*sig*] ...
arg is a command to be 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 as the name of the signal. 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. If *arg* is omitted or is -, then all trap(s)

sig 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 *sig* is `ERR` then *arg* will be executed whenever a command has a non-zero exit status. *sig* is `DEBUG` then *arg* will be executed after each command. If *sig* is `0` or `EXIT` and the `trap` statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is `0` or `EXIT` for a `trap` set outside any function 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.

†† `typeset` [±`HLRZfilrtux`[*n*]] [*name*[=*value*]] ...

Sets attributes and values for shell variables and functions. When invoked inside a function, a new instance of the variable *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

- `-H` This flag provides UNIX to host-name file mapping on non-UNIX machines.
- `-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 variable is assigned to, it 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 variable 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.
- `-f` The names refer to function names rather than variable names. No assignments can be made and the only other valid flags are `-t`, `-u` and `-x`. The flag `-t` turns on execution tracing for this function. The flag `-u` causes this function to be marked undefined. The `FPATH` variable will be searched to find the function definition when the function is referenced. The flag `-x` allows the function definition to remain in effect across shell procedures invoked by name.
- `-i` Variable 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` All upper-case characters converted to lower-case. The upper-case flag, `-u` is turned off.
- `-r` The given *names* are marked readonly and these names cannot be changed by subsequent assignment.

`-t` Tags the variables. Tags are user definable and have no special meaning to the shell.

`-u` All lower-case characters are converted to upper-case characters. The lower-case flag, `-l` is turned off.

`-x` The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands.

The `-i` attribute can not be specified along with `-R`, `-L`, `-Z`, or `-f`.

Using + rather than - 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 *variables* which have these flags set is printed. (Using + rather than - keeps the values from being printed.) If no *name*s and flags are given, the *names* and *attributes* of all *variables* are printed.

`ulimit` [-[`HS`][`a` | `cdfnstv`]]

`ulimit` [-[`HS`][`c` | `d` | `f` | `n` | `s` | `t` | `v`]] *limit*

`ulimit` prints or sets hard or soft resource limits. These limits are described in `getrlimit`(2).

If *limit* is not present, `ulimit` prints the specified limits. Any number of limits may be printed at one time. The `-a` option prints all limits.

If *limit* is present, `ulimit` sets the specified limit to *limit*. The string `unlimited` requests the largest valid limit. Limits may be set for only one resource at a time. Any user may set a soft limit to any value below the hard limit. Any user may lower a hard limit. Only a super-user may raise a hard limit; see `su`(1).

The `-H` option specifies a hard limit. The `-S` option specifies a soft limit. If neither option is specified, `ulimit` will set both limits and print the soft limit. A hard limit cannot be increased once it is set. A soft limit can be increased up to the value of the hard limit. If neither the `-H` nor `-S` option is specified, the limit applies to both.

The following options specify the resource whose limits are to be printed or set. If no option is specified, the file size limit is printed or set.

`-c` maximum core file size (in 512-byte blocks)

`-d` maximum size of data segment or heap (in kbytes)

`-f` maximum file size (in 512-byte blocks)

`-n` maximum file descriptor plus 1

`-s` maximum size of stack segment (in kbytes)

`-t` maximum CPU time (in seconds)

`-v` maximum size of virtual memory (in kbytes)

If no option is given, `-f` is assumed.

`umask` [*mask*]

The user file-creation mask is set to *mask* [see `umask`(2)]. *mask* can either be an octal number or a symbolic value as described in `chmod`(1). If a symbolic value is given, the new umask value is the complement of the result of applying *mask* to the complement of the previous umask value.

If *mask* is omitted, the current value of the mask is printed.

`unalias` *name* ...
The variables given by the list of *name*s are removed from the *alias* list.

`unset` [`-f`] *name* ...
The variables given by the list of *name*s are unassigned, for example, their values and attributes are erased. Read-only variables cannot be unset. If the flag, `-f`, is set, then the names refer to *function* names. Unsetting `ERRNO`, `LINENO`, `MAILCHECK`, `OPTARG`, `OPTIND`, `RANDOM`, `SECONDS`, `TMOUT`, and _ causes removes their special meaning even if they are subsequently assigned to.

†`wait` [*job*]
Wait for the specified *job* and report its termination status. If *job* is not given then all currently active child processes are waited for. The exit status from this command is that of the process waited for. See *Jobs* for a description of the format of *job*.

`whence` [`-pv`] *name* ...
For each *name*, indicate how it would be interpreted if used as a command name.

`-v` produces a more verbose report.

`-p` does a path search for *name* even if name is an alias, a function, or a reserved word.

Invocation

If the shell is invoked by `exec`(2), and the first character of argument zero (`$0`) is `-`, then the shell is assumed to be a `login` shell and commands are read from `/etc/profile` and then from either `.profile` in the current directory or `$HOME/.profile`, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment variable `ENV` if the file exists. If the `-s` flag is not present and *arg* is, then a path search is performed on the first *arg* to determine the name of the script to execute. The script *arg* must have read permission and any `setuid` and `setgid` settings will be ignored. If the script is not found on the path, *arg* is processed as if it named a built-in command or function. 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 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. Shell output, except for the output 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 told by `ioctl`(2)) then this shell is *interactive*. In this case TERM is ignored (so that `kill 0` does not kill an interactive shell) and INTR 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.

rksh Only

rksh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of rksh are identical to those of sh, except that the following are disallowed:

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, 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 (that is, /usr/rbin) that can be safely invoked by rksh.

EXIT STATUS

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 (see also the exit command above). If the shell is being used non-interactively then execution of the shell file is abandoned. Run time errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, then the line number is also printed in square brackets ([]) after the command or function name.

FILES

/etc/passwd
/etc/profile
/etc/suid_profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cat(1), cd(1), chmod(1), cut(1), echo(1), emacs(1), env(1), gmacs(1), newgrp(1M), paste(1), stty(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), ioctl(2), lseek(2), pipe(2), signal(2), umask(2), ulimit(2), wait(2), rand(3C), a.out(4), environ(4), profile(4).

Morris I. Bolsky and David G. Korn, *The KornShell Command and Programming Language*, Prentice Hall, 1989.

NOTES

If a command which is a *tracked alias* is executed, and then 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 `-t` option of the `alias` command to correct this situation.

Some very old shell scripts contain a ^ as a synonym for the pipe character. |.

Using the `fc` built-in command within a compound command will cause the whole command to disappear from the history file.

The built-in command . *file* reads the whole file before any commands are executed. Therefore, `alias` and `unalias` commands in the file will not apply to any functions defined in the file.

Traps are not processed while a job is waiting for a foreground process. Thus, a trap on `CHLD` won't be executed until the foreground job terminates.

NAME

`labelit` (generic) - provide labels for file systems

SYNOPSIS

`labelit` [`-F` *FSType*] [`-V`] [*current_options*] [`-o` *specific_options*] *special* [*operands*]

DESCRIPTION

`labelit` can be used to provide labels for unmounted disk file systems or file systems being copied to tape.

The *special* name should be the disk slice (e.g., `/dev/rdsk/m328_c0d2s7`), or the cartridge tape (e.g., `/dev/rmt/ctape1`). The device may not be on a remote machine. *operands* are *FSType*-specific, and `labelit_`*FSType*(1M) should be consulted for a detailed description.

current_options are options supported by the `s5`-specific module of `labelit`. Other *FSTypes* do not necessarily support these options. *specific_options* indicate suboptions specified in a comma-separated list of suboptions and/or keyword-attribute pairs for interpretation by the *FSType*-specific module of the command. See `labelit_`*FSType*(1M) for details.

The options are:

`-F` specify the *FSType* on which to operate. The *FSType* should either be specified here or be determinable from `/etc/vfstab` by matching the *special* with an entry in the table.

`-V` echo complete command line. This option is used to verify and validate the command line. Additional information obtained via a `/etc/vfstab` lookup is included in the output. The command is not executed.

`-o` Specify *FSType*-specific options.

NOTE

This command may not be supported for all FSTypes.

FILES

`/etc/vfstab` list of default parameters for each file system

SEE ALSO

`labelit_s5`(1M), `labelit_ufs`(1M), `makefsys`(1M), `vfstab`(4).

NAME

`labelit` (s5) - provide labels for `s5` file systems

SYNOPSIS

`labelit` [`-F s5`] [*generic_options*] [`-n`] *special* [*fsname volume*]

DESCRIPTION

generic_options are options supported by the generic `labelit` command.

`labelit` can be used to provide labels for unmounted `s5` disk file systems or `s5` file systems being copied to tape.

With the optional arguments omitted, `labelit` prints current label values.

The *special* name should be the disk slice (e.g., `/dev/rdsk/m328_c0d2s7`), or the cartridge tape (e.g., `/dev/rmt/ctape1`). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (e.g., `root`, `u1`, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette or tape.

For file systems on disk, *fsname* and *volume* are recorded in the superblock.

The options are:

`-F s5` Specifies the `s5`-FSType. Used to ensure that an `s5` file system is labelled.

`-n` Provides for initial tape labeling only (This destroys the previous contents of the tape).

SEE ALSO

generic `labelit`(1M), `makefsys`(1M), s5_specific `mount`(1M).

NAME

`labelit` (ufs) - provide labels for `ufs` file systems

SYNOPSIS

`labelit` [`-F ufs`] [*generic_options*] *special* [*fsname volume*]

DESCRIPTION

generic_options are options supported by the generic `labelit` command.

`labelit` can be used to provide labels for unmounted disk file systems or file systems being copied to tape.

If neither *fsname* nor *volume* is specified, `labelit` prints the current values.

The *special* name should be the physical disk section (for example, `/dev/dsk/c0d0s6`), or the cartridge tape (for example, `/dev/rmt/ctape1`). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (for example, `root`, `u1`, and so on) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette, or tape.

The option is:

`-F ufs` Specifies the `ufs`-FSType.

SEE ALSO

generic `labelit`(1M), `makefsys`(1M), `ufs`(4)

NAME

`langinfo` - language information constants

SYNOPSIS

```
#include <langinfo.h>
```

DESCRIPTION

This header file contains the constants used to identify items of langinfo data. The mode of *items* is given in `nl_types`.

`DAY_1`	Locale's equivalent of 'sunday'
`DAY_2`	Locale's equivalent of 'monday'
`DAY_3`	Locale's equivalent of 'tuesday'
`DAY_4`	Locale's equivalent of 'wednesday'
`DAY_5`	Locale's equivalent of 'thursday'
`DAY_6`	Locale's equivalent of 'friday'
`DAY_7`	Locale's equivalent of 'saturday'
`ABDAY_1`	Locale's equivalent of 'sun'
`ABDAY_2`	Locale's equivalent of 'mon'
`ABDAY_3`	Locale's equivalent of 'tue'
`ABDAY_4`	Locale's equivalent of 'wed'
`ABDAY_5`	Locale's equivalent of 'thur'
`ABDAY_6`	Locale's equivalent of 'fri'
`ABDAY_7`	Locale's equivalent of 'sat'
`MON_1`	Locale's equivalent of 'january'
`MON_2`	Locale's equivalent of 'febuary'
`MON_3`	Locale's equivalent of 'march'
`MON_4`	Locale's equivalent of 'april'
`MON_5`	Locale's equivalent of 'may'
`MON_6`	Locale's equivalent of 'june'
`MON_7`	Locale's equivalent of 'july'
`MON_8`	Locale's equivalent of 'august'
`MON_9`	Locale's equivalent of 'september'
`MON_10`	Locale's equivalent of 'october'
`MON_11`	Locale's equivalent of 'november'
`MON_12`	Locale's equivalent of 'december'
`ABMON_1`	Locale's equivalent of 'jan'
`ABMON_2`	Locale's equivalent of 'feb'

ABMON_3	Locale's equivalent of 'mar'
ABMON_4	Locale's equivalent of 'apr'
ABMON_5	Locale's equivalent of 'may'
ABMON_6	Locale's equivalent of 'jun'
ABMON_7	Locale's equivalent of 'jul'
ABMON_8	Locale's equivalent of 'aug'
ABMON_9	Locale's equivalent of 'sep'
ABMON_10	Locale's equivalent of 'oct'
ABMON_11	Locale's equivalent of 'nov'
ABMON_12	Locale's equivalent of 'dec'
RADIXCHAR	Locale's equivalent of '.'
THOUSEP	Locale's equivalent of ','
YESSTR	Locale's equivalent of 'yes'
NOSTR	Locale's equivalent of 'no'
CRNCYSTR	Locale's currency symbol
D_T_FMT	Locale's default format for date and time
D_FMT	Locale's default format for the date
T_FMT	Locale's default format for the time
AM_STR	Locale's equivalent of 'AM'
PM_STR	Locale's equivalent of 'PM'

This information is retrieved by `nl_langinfo`.

The items `CRNCYSTR`, `RADIXCHAR` and `THOUSEP` are extracted from the fields `currency_symbol`, `decimal_point` and `thousands_sep` in the structure returned by `localeconv`.

The items `T_FMT`, `D_FMT`, `D_T_FMT`, `YESSTR` and `NOSTR` are retrieved from a special message catalog named `Xopen_info` which should be generated for each locale supported and installed in the appropriate directory [see `gettxt`(3C) and `mkmsgs`(1M)]. This catalog should have the messages in the order `T_FMT`, `D_FMT`, `D_T_FMT`, `YESSTR` and `NOSTR`.

All other items are as returned by `strftime`.

SEE ALSO

`chrtbl`(1), `mkmsgs`(1M), `gettxt`(3C), `localeconv`(3C), `nl_langinfo`(3C), `strftime`(3C), `cftime`(4), `nl_types`(5).

NAME

last - indicate last user or terminal logins

SYNOPSIS

last [-n *number* | *-number*] [-f *filename*] [*name* | *tty*] ...

DESCRIPTION

The last command looks in the /var/adm/wtmp, file which records all logins and logouts, for information about a user, a terminal or any group of users and terminals. Arguments specify names of users or terminals of interest. Names of terminals may be given fully or abbreviated. For example last 10 is the same as last term/10. If multiple arguments are given, the information which applies to any of the arguments is printed. For example last root console lists all of root's sessions as well as all sessions on the console terminal. last displays the sessions of the specified users and terminals, most recent first, indicating the times at which the session began, the duration of the session, and the terminal which the session took place on. If the session is still continuing or was cut short by a reboot, last so indicates.

The pseudo-user reboot logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

last with no arguments displays a record of all logins and logouts, in reverse order.

If last is interrupted, it indicates how far the search has progressed in /var/adm/wtmp. If interrupted with a quit signal (generated by a CTRL-\) last indicates how far the search has progressed so far, and the search continues.

The following options are available:

-n *number* | *-number*	Limit the number of entries displayed to that specified by *number*. These options are identical; the *-number* option is provided as a transition tool only and will be removed in future releases.
-f *filename*	Use *filename* as the name of the accounting file instead of /var/adm/wtmp.

FILES

/var/adm/wtmp	accounting file

SEE ALSO

utmp(4).

NAME

lastcomm - show the last commands executed, in reverse order

SYNOPSIS

/usr/ucb/lastcomm [*command-name*] . . . [*user-name*] . . . [*terminal-name*] . . .

DESCRIPTION

The lastcomm command gives information on previously executed commands. lastcomm with no arguments displays information about all the commands recorded during the current accounting file's lifetime. If called with arguments, lastcomm only displays accounting entries with a matching *command-name*, *user-name*, or *terminal-name*.

EXAMPLE

The command:

```
lastcomm a.out root term/01
```

would produce a listing of all the executions of commands named a.out, by user root while using the terminal term/01. and

```
lastcomm root
```

would produce a listing of all the commands executed by user root.

For each process entry, lastcomm displays the following items of information:

the command name under which the process was called

one or more flags indicating special information about the process. The flags have the following meanings:

- F The process performed a fork but not an exec.
- S The process ran as a set-user-id program.

the name of the user who ran the process

the terminal which the user was logged in on at the time (if applicable)

the amount of CPU time used by the process (in seconds)

the date and time the process exited

FILES

/var/adm/pacct accounting file

SEE ALSO

last(1), sigvec(3), acct(4), core(4).

NAME

ld - link editor for object files

SYNOPSIS

ld [*options*] *files* . . .

DESCRIPTION

The ld command combines relocatable object files, performs relocation, and resolves external symbols. ld operates in two modes, static or dynamic, as governed by the -d option. In static mode, -dn, relocatable object files given as arguments are combined to produce an executable object file; if the -r option is specified, relocatable object files are combined to produce one relocatable object file. In dynamic mode, -dy, the default, relocatable object files given as arguments are combined to produce an executable object file that will be linked at execution with any shared object files given as arguments; if the -G option is specified, relocatable object files are combined to produce a shared object. In all cases, the output of ld is left in a.out by default.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. The library may be either a relocatable archive or a shared object. For an archive library, only those routines defining an unresolved external reference are loaded. The archive library symbol table [see ar(4)] is searched sequentially with as many passes as are necessary to resolve external references that can be satisfied by library members. Thus, the ordering of members in the library is functionally unimportant, unless there exist multiple library members defining the same external symbol. A shared object consists of a single entity all of whose references must be resolved within the executable being built or within other shared objects with which it is linked.

The following options are recognized by ld:

-a
: In static mode only, produce an executable object file; give errors for undefined references. This is the default behavior for static mode. -a may not be used with the -r option.

-b
: In dynamic mode only, when creating an executable, do not do special processing for relocations that reference symbols in shared objects. Without the -b option, the link editor will create special position-independent relocations for references to functions defined in shared objects and will arrange for data objects defined in shared objects to be copied into the memory image of the executable by the dynamic linker at run time. With the -b option, the output code may be more efficient, but it will be less sharable.

-d[y | n]
: When -dy, the default, is specified, ld uses dynamic linking; when -dn is specified, ld uses static linking.

-e *epsym*
: Set the entry point address for the output file to be that of the symbol *epsym*.

-h *name*
: In dynamic mode only, when building a shared object, record *name* in the object's dynamic section. *name* will be recorded in executables that are linked with this object rather than the object's UNIX System file name. Accordingly, *name* will be used by the dynamic linker as the name of the shared object to search for at run time.

`-l`*x* Search a library `lib`*x*`.so` or `lib`*x*`.a`, the conventional names for shared object and archive libraries, respectively. In dynamic mode, unless the `-Bstatic` option is in effect, `ld` searches each directory specified in the library search path for a file `lib`*x*`.so` or `lib`*x*`.a`. The directory search stops at the first directory containing either. `ld` chooses the file ending in `.so` if `-l`*x* expands to two files whose names are of the form `lib`*x*`.so` and `lib`*x*`.a`. If no `lib`*x*`.so` is found, then `ld` accepts `lib`*x*`.a`. In static mode, or when the `-Bstatic` option is in effect, `ld` selects only the file ending in `.a`. A library is searched when its name is encountered, so the placement of `-l` is significant.

`-m` Produce a memory map or listing of the input/output sections on the standard output.

`-o` *outfile* Produce an output object file named *outfile*. The name of the default object file is `a.out`.

`-r` Combine relocatable object files to produce one relocatable object file. `ld` will not complain about unresolved references. This option cannot be used in dynamic mode or with `-a`.

`-s` Strip symbolic information from the output file. The debug and line sections and their associated relocation entries will be removed. Except for relocatable files or shared objects, the symbol table and string table sections will also be removed from the output object file.

`-t` Turn off the warning about multiply defined symbols that are not the same size.

`-u` *symname* Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from an archive library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the command line is significant; it must be placed before the library that will define the symbol.

`-z defs` Force a fatal error if any undefined symbols remain at the end of the link. This is the default when building an executable. It is also useful when building a shared object to assure that the object is self-contained, that is, that all its symbolic references are resolved internally.

`-z nodefs` Allow undefined symbols. This is the default when building a shared object. It may be used when building an executable in dynamic mode and linking with a shared object that has unresolved references in routines not used by that executable. This option should be used with caution.

`-z text` In dynamic mode only, force a fatal error if any relocations against non-writable, allocatable sections remain.

`-z sysinuser`
Allow system segments in user space, i.e. below address 0x80000000. This option applies to MC88000 based systems only.

`-z lowzeros`
: Allow NULL pointer dereferences.

`-B [dynamic | static]`
: Options governing library inclusion. `-Bdynamic` is valid in dynamic mode only. These options may be specified any number of times on the command line as toggles: if the `-Bstatic` option is given, no shared objects will be accepted until `-Bdynamic` is seen. See also the `-l` option.

`-Bsymbolic`
: In dynamic mode only, when building a shared object, bind references to global symbols to their definitions within the object, if definitions are available. Normally, references to global symbols within shared objects are not bound until run time, even if definitions are available, so that definitions of the same symbol in an executable or other shared objects can override the object's own definition. `ld` will issue warnings for undefined symbols unless `-z defs` overrides.

`-G`
: In dynamic mode only, produce a shared object. Undefined symbols are allowed.

`-I` *name*
: When building an executable, use *name* as the path name of the interpreter to be written into the program header. The default in static mode is no interpreter; in dynamic mode, the default is the name of the dynamic linker, `/usr/lib/libc.so.1`. Either case may be overridden by `-I`. `exec` will load this interpreter when it loads the `a.out` and will pass control to the interpreter rather than to the `a.out` directly.

`-L` *path*
: Add *path* to the library search directories. `ld` searches for libraries first in any directories specified with `-L` options, then in the standard directories. This option is effective only if it precedes the `-l` option on the command line.

`-M` *mapfile*
: In *static* mode only, read *mapfile* as a text file of directives to `ld`. Because these directives change the shape of the output file created by `ld`, use of this option is strongly discouraged.

`-Q[y | n]`
: Under `-Qy`, an `ident` string is added to the `.comment` section of the output file to identify the version of the link editor used to create the file. This will result in multiple `ld idents` when there have been multiple linking steps, such as when using `ld -r`. This is identical with the default action of the `cc` command. `-Qn` suppresses version.

`-V`
: Output a message giving information about the version of `ld` being used.

`-YP,` *dirlist*
: Change the default directories used for finding libraries. *dirlist* is a colon-separated path list.

The environment variable `LD_LIBRARY_PATH` may be used to specify library search directories. In the most general case, it will contain two directory lists separated by a semicolon:

dirlist1 ; *dirlist2*

If ld is called with any number of occurrences of -L, as in

ld... -L*path1* ... -L*pathn* ...

then the search path ordering is

dirlist1 path1 ... pathn dirlist2 LIBPATH

LD_LIBRARY_PATH is also used to specify library search directories to the dynamic linker at run time. That is, if LD_LIBRARY_PATH exists in the environment, the dynamic linker will search the directories named in it, before its default directory, for shared objects to be linked with the program at execution.

The environment variable LD_RUN_PATH, containing a directory list, may also be used to specify library search directories to the dynamic linker. If present and not null, it is passed to the dynamic linker by ld via data stored in the output object file.

FILES

lib*x*.so	libraries
lib*x*.a	libraries
a.out	output file
LIBPATH	usually /usr/ccs/lib:/usr/lib

SEE ALSO

as(1), cc(1), exec(2), exit(2), end(3C), a.out(4), ar(4).

NOTES

Through its options, the link editor gives users great flexibility; however, those who use the -M *mapfile* option must assume some added responsibilities. Use of this feature is strongly discouraged.

NAME

ld - link editor, dynamic link editor

SYNOPSIS

/usr/ucb/ld [*options*]

DESCRIPTION

/usr/ucb/ld is the link editor for the BSD Compatibility Package. /usr/ucb/ld is identical to /usr/bin/ld [see ld(1)] except that BSD libraries and routines are included *before* System V libraries and routines.

/usr/ucb/ld accepts the same options as /usr/bin/ld, with the following exceptions:

-L *dir* Add *dir* to the list of directories searched for libraries by /usr/bin/ld. Directories specified with this option are searched before /usr/ucblib and /usr/lib.

-Y LU, *dir*
Change the default directory used for finding libraries. Warning: this option may have unexpected results, and should not be used.

FILES

/usr/ucblib
/usr/lib
/usr/ucblib/libx.a
/usr/lib/libx.a

SEE ALSO

ar(1), as(1), cc(1), ld(1), lorder(1), strip(1), tsort(1).

NAME

ldd - list dynamic dependencies

SYNOPSIS

ldd [-d | -r] *file*

DESCRIPTION

The ldd command lists the path names of all shared objects that would be loaded as a result of executing *file*. If *file* is a valid executable but does not require any shared objects, ldd will succeed, producing no output.

ldd may also be used to check the compatibility of *file* with the shared objects it uses. It does this by optionally printing warnings for any unresolved symbol references that would occur if *file* were executed. Two options govern this mode of ldd:

-d Causes ldd to check all references to data objects.

-r Causes ldd to check references to both data objects and functions.

Only one of the above options may be given during any single invocation of ldd.

SEE ALSO

cc(1), ld(1).

DIAGNOSTICS

ldd prints its record of shared object path names to stdout. The optional list of symbol resolution problems are printed to stderr. If *file* is not an executable file or cannot be opened for reading, a non-zero exit status is returned.

NOTES

ldd doesn't list shared objects explicitly attached via dlopen(3X).

ldd uses the same algorithm as the dynamic linker to locate shared objects.

NAME

ldsysdump - load system dump from selected devices

SYNOPSIS

/usr/sbin/ldsysdump [-u] [-a] [-f *special device*] *destination_file*

DESCRIPTION

The ldsysdump command loads the memory image files from the special device (/dev/rmt/ctape1 default) used to take a crash dump and recombines them into a single file on the hard disk suitable for use by the crash command. The *destination_file* is the name of the hard disk file into which the data from the source media will be loaded.

When invoked, ldsysdump begins an interactive procedure that prompts the user to insert the media into the drive that the sysdump will be loaded from. In the case where the dump image is contained in a hard disk slice, no loading is required. The user has the option of quitting the session at any time. This allows only the portion of the system image needed to be dumped, if required.

If ldsysdump is invoked with the -a option, it will not prompt the user and fails if it encounters an error condition.

After the dump is recovered and if the source of the dump is a disk, ldsysdump writes a marker into the source dump image to mark the dump as used. It will refuse to recover such a marked dump unless the -u flag is provided.

EXAMPLES

This example loads the cartridge tape produced via a crash dump on a machine equipped with 8 MB of memory.

```
$ldsysdump -f /dev/rmt/ctape1 /var/tmp/cdump

Insert media segment 0.
Enter 'c' to continue, 'q' to quit: c

Attempting to read sysdump of Wed Apr  4 11:58:37 1990

Sysdump image is 8388608 bytes starting at physical memory location 0x0
.....................................................................
.....................................
1 Sysdump files coalesced, 8389632 bytes transferred
$
```

FILES

/dev/rmt/ctape1 device used for cartridge tape

SEE ALSO

crash(1M), crashconf(1M).

DIAGNOSTICS

If a cartridge tape is inserted out of sequence a message is printed. The user is allowed to insert a new one and continue the session.

NOTES

Since the Motorola Delta computer can be equipped with an amount much greater than 16 MB of memory, the *destination_file* can become quite large. The filesystem must have enough free space to hold the *destination_file*.

NAME

lex - generate programs for simple lexical tasks

SYNOPSIS

lex [-ctvn -V -Q[y|n]] [*file*]

DESCRIPTION

The lex command generates programs to be used in simple lexical analysis of text.

The input *file*s (standard input default) contain strings and expressions to be searched for and C text to be executed when these strings are found.

lex generates a file named lex.yy.c. When lex.yy.c is compiled and linked with the lex library, it copies the input to the output except when a string specified in the file is found. When a specified string 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 patterns in the *file*. The patterns 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 occurrence of, the previous character or character class. Thus, [a-zA-Z]+ matches a string of letters. 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 \.

Three 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 lex library contains a main() that calls it. The macros input and output read from and write to stdin and stdout, respectively.

The function yymore accumulates additional characters into the same yytext. The function yyless(*n*) pushes back yyleng -*n* characters into the input stream. (yyleng is an external int variable giving the length in bytes of yytext.) The function yywrap is called whenever the scanner reaches end of file and indicates whether normal wrapup should continue. The action REJECT on the right side of the rule causes the match to be rejected and the next suitable match executed. The action ECHO on the right side of the rule is equivalent to printf("%s", yytext).

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. Lines preceding %% that 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 {}. In this section, C code (and preprocessor statements) can also be included between %{ and %}. Note that curly brackets do not imply parentheses; only string substitution is done.

The external names generated by `lex` all begin with the prefix `yy` or `YY`.

The flags must appear before any files.

`-c` Indicates C actions and is the default.

`-t` Causes the `lex.yy.c` program to be written instead to standard output.

`-v` Provides a two-line summary of statistics.

`-n` Will not print out the `-v` summary.

`-V` Print out version information on standard error.

`-Q[y|n]` Print out version information to output file `lex.yy.c` by using `-Qy`. The `-Qn` option does not print out version information and is the default.

Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain default table sizes are too small for some users. The table sizes for the resulting finite state machine can be set in the definitions section:

`%p` *n* number of positions is *n* (default 2500)

`%n` *n* number of states is *n* (500)

`%e` *n* number of parse tree nodes is *n* (1000)

`%a` *n* number of transitions is *n* (2000)

`%k` *n* number of packed character classes is *n* (2500)

`%o` *n* size of output array is *n* (3000)

The use of one or more of the above automatically implies the `-v` option, unless the `-n` option is used.

EXAMPLE

```
D       [0-9]
%{
void
skipcommnts(void)
{
        for(;;)
        {
                while(input()!='*')
                        ;
                if(input()=='/')
                        return;
                else

                        unput(yytext[yyleng-1]);
        }
}
%}
%%
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");
"\n"    ;/*no action */
"/*"      skipcommnts();
%%
```

INTERNATIONAL FUNCTIONS

`lex` can process characters from supplementary code sets as well as ASCII characters.

Characters from supplementary code sets can be specified in comments which exist in definitions, rules, and user subroutines.

Characters from supplementary code sets can be specified in strings which exist in actions in rules and in user subroutines.

Character strings from supplementary code sets can be defined as tokens.

`input()`, `unput(c)` and `output(c)` functions are performed in byte. The value of `yyleng` is in bytes, not in characters.

Characters from supplementary code sets which define token are restricted to single-byte characters.

SEE ALSO

`yacc`(1).

NAME

lfmt - display error message in standard format and pass to logging and monitoring services

SYNOPSIS

`lfmt [-c][-f`*flags*`][-l`*label*`][-s`*severity*`][-g`*catalog*`:`*msgnum*`] `*format*` [`*args*`]`

DESCRIPTION

`lfmt` uses *format* for `printf` style formatting of *args*. If the `-g` option is specified, `lfmt` retrieves a localized version of the *format* string from a locale-specific message database. The output is displayed on *stderr*.

`lfmt` encapsulates the output in the standard error message format.

`lfmt` forwards its output to the logging and monitoring facility. Optionnaly, `lfmt` will display the output on the console, with a date and time stamp.

The `-c` option causes the message to also be displayed along with a date/time stamp on the console.

The `-f`flags option specifies logging information as a comma-separated list of keywords from the sets:

Major classification
: Identifies the source of the condition. Identifiers are: `hard` (hardware), `soft` (software), and `firm` (firmware).

Message source subclassification
: Identifies the type of software in which the problem is spotted. Identifiers are: `appl` (application), `util` (utility), and `opsys` (operating system).

The `-l`*label* option specifies the label string to be displayed with the message (e.g. `"UX:cat"`). *label* is a character string no more than 25 characters in length; it will be automatically suffixed with a colon (`:`). When unspecified, no label is displayed as part of the message.

The `-s`*severity* option specifies the severity string to be displayed with the message. Acceptable strings include the standard severities in either their print string (i.e. `HALT`, `ERROR`, `INFO`, `WARNING`, and `"TO FIX"`) or keyword (i.e. `halt`, `error`, `info`, `warn`, and `action`) forms, or any other user-defined string. A user-defined string will be assigned the integer severity value of 5. The severity will be suffixed with a colon (`:`). The `ERROR` severity will be used if no severity is specified.

The `-g`*catalog*`:`*msgnum* option specifies that a localized version of the *format* should be retrieved from a message database. *catalog* is used to indicate the message database that contains the localized version of the *format* string. *catalog* must be limited to 14 characters. These characters must be selected from a set of all characters values, excluding `\0` (null) and the ASCII codes for `/` (slash) and `:` (colon).

msgnum is a positive number that indicates the index of the string into the message database.

If the catalog does not exist in the current locale (identified by the `LC_MESSAGES` or `LANG` environment variables), or if the message number is out of bound, `lfmt` will attempt to retrieve the message from the C locale. If this second retrieval fails, `lfmt` uses the *format* string as passed on the command line.

`lfmt` will output `Message not found!!\n` as *format* string if *catalog* is not a valid catalog name, or if *msgnum* is not a valid number.

STANDARD ERROR MESSAGE FORMAT

`lfmt` displays error messages in the following format:

label`:` *severity*`:` *text*

If no *label* was defined using the `-l`*label* option, the message is displayed in the format:

severity`:` *text*

If `lfmt` is called twice to display an error message and a helpful *action* or recovery message, the output can look like:

label`:` *severity*`:` *text*
label`: TO FIX:` *text*

ERRORS

Upon success, `lfmt` exits with code 0. Upon failure, `lfmt` exits with the following codes:

1 write error.

2 cannot log or forward to console.

3 syntax error.

EXAMPLE

```
lfmt -fsoft,util -l UX:test -s info "test facility enabled\n"
```

displays the message to *stderr* and makes it available for logging:

```
UX:test: INFO: test facility enabled
```

SEE ALSO

`environ`(5), `gettxt`(1), `pfmt`(1), `lfmt`(3C), `printf`(1),

NAME

`line` - read one line

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.

SEE ALSO

`sh`(1), `read`(2).

NAME

link, unlink - link and unlink files and directories

SYNOPSIS

/usr/sbin/link *file1 file2*
/usr/sbin/unlink *file*

DESCRIPTION

The link command is used to create a file name that points to another file. Linked files and directories can be removed by the unlink command; however, it is strongly recommended that the rm and rmdir commands be used instead of the unlink command.

The only difference between ln and link and unlink is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the link and unlink system calls.

SEE ALSO

rm(1), link(2), unlink(2).

NOTES

These commands can be run only by the super-user.

NAME

lint - a C program checker

SYNOPSIS

lint [*options*] *files*

DESCRIPTION

lint detects features of C program files which are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compiler. lint issues error and warning messages. Among the things it detects are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. lint checks for 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 takes all the .c, .ln, and llib-l*x*.ln (specified by -l*x*) files and processes 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. When the -c option is used, the .ln and the llib-l*x*.ln files are ignored. When the -c option is not used, the second pass of lint checks the .ln and the llib-l*x*.ln list of files for mutual compatibility.

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.

-h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

-m Suppress complaints about external symbols that could be declared static.

-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:

-I*dir* Search for included header files in the directory *dir* before searching the current directory and/or the standard place.

-l*x* Include the 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.

-L*dir* Search for lint libraries in *dir* before searching the standard place.

-n Do not check compatibility against the standard C 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.

-s Produce one-line diagnostics only. lint occasionally buffers messages to produce a compound report.

-k Alter the behavior of /*LINTED [*message*]*/ directives. Normally, lint will suppress warning messages for the code following these directives. Instead of suppressing the messages, lint prints an additional message containing the comment inside the directive.

-y Specify that the file being linted will be treated as if the /*LINTLIBRARY*/ directive had been used. A lint library is normally created by using the /*LINTLIBRARY*/ directive.

-F Print pathnames of files. lint normally prints the filename without the path.

-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*x* Cause lint to create a lint library with the name llib-l*x*.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*x*.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.

Some of the above settings are also available through the use of "lint comments" (see below).

-V Write to standard error the product name and release.

-W*file* Write a .ln file to *file*, for use by cflow(1).

-R*file* Write a .ln file to *file*, for use by cxref(1).

lint recognizes many cc(1) command line options, including -D, -U, -g, -O, -Xt, -Xa, and -Xc, although -g and -O are ignored. Unrecognized options are warned about and ignored. The predefined macro lint is defined to allow certain questionable code to be altered or removed for lint. Thus, 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`:

/*ARGSUSED*n*/
: makes `lint` check only the first *n* arguments for usage; a missing *n* is taken to be 0 (this option acts like the `-v` option for the next function).

/*CONSTCOND*/ or /*CONSTANTCOND*/ or /*CONSTANTCONDITION*/
: suppresses complaints about constant operands for the next expression.

/*EMPTY*/
: suppresses complaints about a null statement consequent on an if statement. This directive should be placed after the test expression, and before the semicolon. This directive is supplied to support empty if statements when a valid else statement follows. It suppresses messages on an empty `else` consequent.

/*FALLTHRU*/ or /*FALLTHROUGH*/
: suppresses complaints about fall through to a `case` or `default` labeled statement. This directive should be placed immediately preceding the label.

/*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.

/*LINTED [*message*]*/
: suppresses any intra-file warning except those dealing with unused variables or functions. This directive should be placed on the line immediately preceding where the lint warning occurred. The `-k` option alters the way in which `lint` handles this directive. Instead of suppressing messages, `lint` will print an additional message, if any, contained in the comment. This directive is useful in conjunction with the `-s` option for post-lint filtering.

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code. [This comment is typically placed just after calls to functions like `exit`(2)].

/*PRINTFLIKE*n*/
: makes `lint` check the first *(n-1)* arguments as usual. The *nth* argument is interpreted as a `printf` format string that is used to check the remaining arguments.

/*PROTOLIB*n*/
: causes `lint` to treat function declaration prototypes as function definitions if *n* is non-zero. This directive can only be used in conjunction with the
/* LINTLIBRARY */ directive. If *n* is zero, function prototypes will be treated normally.

/*SCANFLIKE*n*/
: makes `lint` check the first *(n-1)* arguments as usual. The *nth* argument is interpreted as a `scanf` format string that is used to check the remaining arguments.

/*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. The use of the ellipsis terminator (...) in the definition is suggested in new or updated code.

`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, if `-s` is not specified. 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 filename 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 that 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`; 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.

FILES

LIBDIR	the directory where the lint libraries specified by the `-l`*x* option must exist
LIBDIR`/lint[12]`	first and second passes
LIBDIR`/llib-lc.ln`	declarations for C Library functions (binary format; source is in *LIBDIR*`/llib-lc`)
LIBPATH`/llib-lm.ln`	declarations for Math Library functions (binary format; source is in *LIBDIR*/llib-lm)
TMPDIR`/*lint*`	temporaries
TMPDIR	usually `/var/tmp` but can be redefined by setting the environment variable `TMPDIR` [see `tempnam` in `tmpnam`(3S)].
LIBDIR	usually `/ccs/lib`
LIBPATH	usually `/usr/ccs/lib:/usr/lib`

SEE ALSO

`cc`(1), `make`(1).

NAME

listdgrp - lists members of a device group

SYNOPSIS

listdgrp *dgroup*

DESCRIPTION

listdgrp displays the members of the device group specified by the *dgroup*.

ERRORS

This command will exit with one of the following values:

0 = successful completion of the task.

1 = command syntax incorrect, invalid option used, or internal error occurred.

2 = device group table could not be opened for reading.

3 = device group *dgroup* could not be found in the device group table.

EXAMPLE

To list the devices that belong to group slices:

```
$ listdgrp slices
root
swap
usr
```

FILES

/etc/dgroup.tab

SEE ALSO

putdgrp(1)

NAME

listen - network listener daemon

SYNOPSIS

/usr/lib/saf/listen [-m *devstem*] *net_spec*

DESCRIPTION

The listen process "listens" to a network for service requests, accepts requests when they arrive, and invokes servers in response to those service requests. The network listener process may be used with any connection-oriented network (more precisely, with any connection-oriented transport provider) that conforms to the Transport Interface (TLI) specification.

The listener internally generates a pathname for the minor device for each connection; it is this pathname that is used in the utmp entry for a service, if one is created. By default, this pathname is the concatenation of the prefix /dev/*netspec* with the decimal representation of the minor device number. When the -m *devstem* option is specified, the listener will use *devstem* as the prefix for the pathname. In either case, the representation of the minor device number will be at least two digits (for example, 05 or 27), but will be longer when necessary to accommodate minor device numbers larger than 99.

SERVER INVOCATION

When a connection indication is received, the listener creates a new transport endpoint and accepts the connection on that endpoint. Before giving the file descriptor for this new connection to the server, any designated STREAMS modules are pushed and the configuration script is executed, if one exists. This file descriptor is appropriate for use with either TLI (see especially t_sync(3N)) or the sockets interface library.

By default, a new instance of the server is invoked for each connection. When the server is invoked, file descriptor 0 refers to the transport endpoint, and is open for reading and writing. File descriptors 1 and 2 are copies of file descriptor 0; no other file descriptors are open. The service is invoked with the user and group IDs of the user name under which the service was registered with the listener, and with the current directory set to the HOME directory of that user.

Alternatively, a service may be registered so that the listener will pass connections to a standing server process through a FIFO or a named STREAM, instead of invoking the server anew for each connection. In this case, the connection is passed in the form of a file descriptor that refers to the new transport endpoint. Before the file descriptor is sent to the server, the listener interprets any configuration script registered for that service using doconfig(3N), although doconfig is invoked with both the NORUN and NOASSIGN flags. The server receives the file descriptor for the connection in a strrecvfd structure via an I_RECVFD ioctl(2).

For more details about the listener and its administration, see nlsadmin(1M).

FILES

/etc/saf/*pmtag*/*

SEE ALSO

nlsadmin(1M), pmadm(1M), sac(1M), sacadm(1M), doconfig(3N), nlsgetcall, nlsprovider(3N), streamio(7).

NOTES

When passing a connection to a standing server, the user and group IDs contained in the `strrecvfd` structure will be those for the listener (that is, they will both be 0); the user name under which the service was registered with the listener is not reflected in these IDs.

When operating multiple instances of the listener on a single transport provider, there is a potential race condition in the binding of addresses during initialization of the listeners if any of their services have dynamically assigned addresses. This condition would appear as an inability of the listener to bind a static-address service to its otherwise valid address, and would result from a dynamic-address service having been bound to that address by a different instance of the listener.

NAME

`listusers` - list user login information

SYNOPSIS

`listusers` [`-g` *groups*] [`-l` *logins*]

DESCRIPTION

Executed without any options, this command displays a list of all user logins, sorted by login, and the account field value associated with each login in `/etc/passwd`.

`-g` Lists all user logins belonging to `group`, sorted by login. Multiple groups can be specified as a comma-separated list.

`-l` Lists the user login or logins specified by `logins`, sorted by login. Multiple logins can be specified as a comma-separated list.

NOTES

A user login is one that has a UID of 100 or greater.

The `-l` and `-g` options can be combined. User logins will be listed only once, even if they belong to more than one of the selected groups.

NAME

ln - link files

SYNOPSIS

ln [-s] [-f] [-n] *file1 target*
ln [-s] [-f] [-n] *file1* [*file2 ...*] *targetdir*

DESCRIPTION

The ln command links *filen* to *target* by creating a directory entry that refers to *target*. By using ln with one or more file names, the user may create one or more links to *targetdir* (if target is a directory).

The ln command may be used to create both hard links and symbolic links; by default it creates hard links. A hard link to a file is indistinguishable from the original directory entry. Any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

Without the -s option, ln is used to create hard links. *filen* is linked to *target*. If *target* is a directory, another file named *filen* is created in *target* and linked to the original *filen*. If *target* is a file, its contents are overwritten.

If ln determines that the mode of *target* forbids writing, it will print the mode [see chmod(2)], ask for a response, and read the standard input for one line. If the line begins with y, the link occurs, if permissible; otherwise, the command exits.

There are three options to ln. If multiple options are specified, the one with the highest priority is used and the remainder are ignored. The options, in descending order of priority, are:

-s ln will create a symbolic link. A symbolic link contains the name of the file to which it is linked. Symbolic links may span file systems and may refer to directories. If the linkname exists, then do not overwrite the contents of the file. A symbolic link's permissions are always set to read, write, and execute permission for owner, group, and world (777).

-f ln will link files without questioning the user, even if the mode of *target* forbids writing. Note that this is the default if the standard input is not a terminal.

-n If the linkname is an existing file, do not overwrite the contents of the file. The -f option overrides this option.

If the -s option is used with two arguments, *target* may be an existing directory or a non-existent file. If *target* already exists and is not a directory, an error is returned. *filen* may be any path name and need not exist. If it exists, it may be a file or directory and may reside on a different file system from *target*. If *target* is an existing directory, a file is created in directory *target* whose name is *filen* or the last component of *filen*. This file is a symbolic link that references *filen*. If *target* does not exist, a file with name *target* is created and it is a symbolic link that references *filen*.

If the -s option is used with more than two arguments, *target* must be an existing directory or an error will be returned. For each *filen*, a file is created in *target* whose name is *filen* or its last component; each new *filen* is a symbolic link to the original *filen*. The *files* and *target* may reside on different file systems.

SEE ALSO

chmod(1), cp(1), mv(1), rm(1), link(2), readlink(2), stat(2), symlink(2)

NOTES

Doing operations that involve ".." (such as "cd ..") in a directory that is symbolically linked will reference the original directory not the target.

The -s option does not use the current working directory. In the command

ln -s *path target*

path is taken literally without being evaluated against the current working directory.

NAME

ln - make hard or symbolic links to files

SYNOPSIS

/usr/ucb/ln [-fs] *filename* [*linkname*]
/usr/ucb/ln [-fs] *pathname* . . . *directory*

DESCRIPTION

/usr/ucb/ln creates an additional directory entry, called a link, to a file or directory. Any number of links can be assigned to a file. The number of links does not affect other file attributes such as size, protections, data, and so on.

filename is the name of the original file or directory. *linkname* is the new name to associate with the file or filename. If *linkname* is omitted, the last component of *filename* is used as the name of the link.

If the last argument is the name of a directory, symbolic links are made in that directory for each *pathname* argument; /usr/ucb/ln uses the last component of each *pathname* as the name of each link in the named *directory*.

A hard link (the default) is a standard directory entry just like the one made when the file was created. Hard links can only be made to existing files. Hard links cannot be made across file systems (disk slices, mounted file systems). To remove a file, all hard links to it must be removed, including the name by which it was first created; removing the last hard link releases the inode associated with the file.

A symbolic link, made with the -s option, is a special directory entry that points to another named file. Symbolic links can span file systems and point to directories. In fact, you can create a symbolic link that points to a file that is currently absent from the file system; removing the file that it points to does not affect or alter the symbolic link itself.

A symbolic link to a directory behaves differently than you might expect in certain cases. While an ls(1V) on such a link displays the files in the pointed-to directory, an 'ls -l' displays information about the link itself:

```
example% /usr/ucb/ln -s dir link
example% ls link
file1 file2 file3 file4
example% ls -l link
lrwxrwxrwx  1 user            7 Jan 11 23:27 link -> dir
```

When you cd(1) to a directory through a symbolic link, you wind up in the pointed-to location within the file system. This means that the parent of the new working directory is not the parent of the symbolic link, but rather, the parent of the pointed-to directory. For instance, in the following case the final working directory is /usr and not /home/user/linktest.

```
example% pwd
/home/user/linktest
example% /usr/ucb/ln -s /var/tmp symlink
example% cd symlink
example% cd ..
example% pwd
/usr
```

C shell user's can avoid any resulting navigation problems by using the `pushd` and `popd` built-in commands instead of `cd`.

OPTIONS

`-f` Force a hard link to a directory — this option is only available to the super-user.

`-s` Create a symbolic link or links.

EXAMPLE

The commands below illustrate the effects of the different forms of the `/usr/ucb/ln` command:

```
example% /usr/ucb/ln file link
example% ls -F file link
file   link
example% /usr/ucb/ln -s file symlink
example% ls -F file symlink
file   symlink@
example% ls -li file link symlink
 10606 -rw-r--r--  2 user          0 Jan 12 00:06 file
 10606 -rw-r--r--  2 user          0 Jan 12 00:06 link
 10607 lrwxrwxrwx  1 user          4 Jan 12 00:06 symlink -> file
example% /usr/ucb/ln -s nonesuch devoid
example% ls -F devoid
devoid@
example% cat devoid
devoid: No such file or directory
example% /usr/ucb/ln -s /proto/bin/* /tmp/bin
example% ls -F /proto/bin /tmp/bin
/proto/bin:
x*      y*      z*

/tmp/bin:
x@      y@      z@
```

SEE ALSO

`cp`(1), `ls`(1), `mv`(1), `rm`(1) in the *User's Reference Manual*

`link`(2), `readlink`(2), `stat`(2), `symlink`(2) in the *Programmer's Reference Manual*

NOTES

When the last argument is a directory, simple basenames should not be used for *pathname* arguments. If a basename is used, the resulting symbolic link points to itself:

```
example% /usr/ucb/ln -s file /tmp
example% ls -l /tmp/file
lrwxrwxrwx  1 user           4 Jan 12 00:16 /tmp/file -> file
example% cat /tmp/file
/tmp/file: Too many levels of symbolic links
```

To avoid this problem, use full pathnames, or prepend a reference to the PWD variable to files in the working directory:

```
example% rm /tmp/file
example% /usr/ucb/ln -s $PWD/file /tmp
lrwxrwxrwx  1 user
      4 Jan 12 00:16 /tmp/file -> /home/user/subdir/file
```

NAME

loadimg - CD-ROM test image loader

SYNOPSIS

loadimg *[-f imagesource] diskdev*
loadimg *-t [-f imagesource]*

SUMMARY

loadimg is designed to be used as a test tool for CD-ROM images created by *fsgen*. The program loads an entire CD-ROM image onto a raw disk device where it can be examined as if it was recorded on a CD-ROM (but read-write). The intent is to allow examination of CD-ROM images that were prepared on tape. The *loadimg* program requires a scratch disk drive of sufficient size to hold the supplied image.

DESCRIPTION

The *loadimg* program normally requires only one argument, *diskdev,* or the destination disk drive. The raw disk device of the target disk drive should be supplied. The entire disk drive is used to load the supplied CD-ROM image; the original contents of the disk is *entirely destroyed*. Since access to the raw disk devices is usually restricted to **root**, this program must be run as Superuser.

An optional **-t** argument, provides a slice-table dump from the source image. If the **-t** option is used, no attempt is made to copy the image and it is not necessary to supply a *diskdev* argument.

The *loadimg* program assumes a default image source of the tape drive **/dev/rmt/ctape**. If the source image is in another location, use the **-f** option to supply a new source path. For example, to load an image from tape to the second disk drive on the first SCSI bus on an MVME328 disk controller, the command is:

```
$ loadimg -f /dev/rmt/r40t /dev/rdsk/m328_c0d2s7
```

or

```
$ loadimg /dev/rdsk/m328_c0d2s7
```

The *loadimg* program checks to see that the source image will fit on the supplied drive before copying the image. It also checks that both the source image and the target disk drive have valid initial formats, and that both have valid Motorola Volume Identification blocks (VIDs). For this reason, the target disk drive must have been correctly formatted and the CD-ROM image must begin with a MOTVID slice (see the *fsgen (1)* utility).

The *loadimg* utility preserves the initial contents of the VID block on the target disk drive except for those fields that indicate bootloader address and root file system offset. The target disks slice table are replaced with the source image´s slice table except for slice 7 (the entire disk). If the target disk drive needs to be returned to normal use, run the *fmthard* and *prtvtoc* utilities to reslice the disk and build the required file systems.

WARNINGS AND DIAGNOSTICS.

While running, various progress messages are printed, along with a slice table dump for the source image. If either the source image or the target disk drive are found not to have a valid VID block, the *loadimg* program aborts. Also, if the source image is found to be too big to fit on the target disk, the copy is not performed.

loadimg always terminates with a zero exit value unless an error is encountered. The exit value is picked from the common set of exit values used by the CD-ROM tools. These exit values are listed in the following table:

Error Class	Exit Value
Out of memory, or corrupted memory detected	11 (BADMEM)
File access problems (including missing files)	12 (BADFILE)
I/O operation errors	13 (BADIO)
Illegal input data format	14 (BADFMT)
Internal program errors detected	15 (INTERR)

SEE ALSO

`fsgen`(1), `fscoll`(1).
CD-ROM Image Generation Tools User's Guide

NAME

`lockd` - network lock daemon

SYNOPSIS

`/usr/lib/nfs/lockd` [`-t` *timeout*] [`-g` *graceperiod*]

DESCRIPTION

`lockd` processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. `lockd` forwards lock requests for remote data to the server site's lock daemon through RPC/XDR. `lockd` then requests the status monitor daemon, `statd`(1M), for monitor service. The reply to the lock request will not be sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all client-site lock daemons to submit reclaim requests. Client-site lock daemons, on the other hand, are notified by the status monitor daemon of the server recovery and promptly resubmit previously granted lock requests. If a lock daemon fails to secure a previously granted lock at the server site, the it sends SIGLOST to a process.

OPTIONS

`-t` *timeout*	Use *timeout* `seconds` as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.
`-g` *graceperiod*	Use *graceperiod* `seconds` as the grace period duration instead of the default value (45 seconds).

SEE ALSO

`statd`(1M), `fcntl`(2), `signal`(2), `lockf`(3C)

NAME

logger - add entries to the system log

SYNOPSIS

/usr/ucb/logger [-t *tag*] [-p *priority*] [-i] [-f *filename*] [*message*] . . .

DESCRIPTION

logger provides a method for adding one-line entries to the system log file from the command line. One or more *message* arguments can be given on the command line, in which case each is logged immediately. Otherwise, a *filename* can be specified, in which case each line in the file is logged. If neither is specified, logger reads and logs messages on a line-by-line basis from the standard input.

The following options are available:

-t *tag*	Mark each line added to the log with the specified *tag*.
-p *priority*	Enter the message with the specified *priority*. The message priority can be specified numerically, or as a *facility*.*level* pair. For example, '-p local3.info' assigns the message priority to the info level in the local3 facility. The default priority is user.notice.
-i	Log the process ID of the logger process with each line.
-f *filename*	Use the contents of *filename* as the message to log.
message	If this is unspecified, either the file indicated with -f or the standard input is added to the log.

EXAMPLE

```
logger System rebooted
```

will log the message 'System rebooted' to the facility at priority notice to be treated by syslogd as other messages to the facility notice are.

```
logger -p local0.notice -t HOSTIDM -f /dev/idmc
```

will read from the file /dev/idmc and will log each line in that file as a message with the tag 'HOSTIDM' at priority notice to be treated by syslogd as other messages to the facility local0 are.

SEE ALSO

syslog(3), syslogd(1M)

NAME

login - sign on

SYNOPSIS

login [-d *device*] [*name* [*environ* . . .]]

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. It is invoked by the system when a previous user has terminated the initial shell by typing a CTRL-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 it is 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.

If there are no lower-case characters in the first line of input processed, login assumes the connecting TTY is an upper-case-only terminal and sets the port's termio(7) options to reflect this.

login accepts a device option, *device*. *device* is taken to be the path name of the TTY port login is to operate on. The use of the device option can be expected to improve login performance, since login will not need to call ttyname(3).

If you make any mistake in the login procedure, the message

```
Login incorrect
```

is printed and a new login prompt will appear. If you make five incorrect login attempts, all five may be logged in /var/adm/loginlog (if it exists) and the TTY line will be dropped.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login (Bourne shell or Korn shell), accounting files are updated, the /etc/profile script is executed, the time you last logged in is printed, /etc/motd is printed, the user-ID, group-ID, supplementary group list, working directory, and command interpreter (usually sh) are initialized, and the file .profile in the working directory is executed, if it exists. (For the C shell, /etc/profile is not executed, /etc/motd is not printed, and .profile is not executed; instead, the C shell executes the startup file .cshrc.) The name of the command interpreter is - followed by the last component of the interpreter's path name (e.g., -sh). If this field in the password file is empty, then the default command interpreter, /usr/bin/sh is used. If this field is *, then the named directory becomes the root directory, the starting point for path searches for path names beginning with a /. At that point login is re-executed at the new level which must have its own root structure, including /var/adm/login and /etc/passwd.

`login` reads and sets its environment from `/etc/default/login`. The basic *environment* is initialized to:

`HOME=`*your-login-directory*
`LOGNAME=`*your-login-name*
`PATH=/usr/bin`
`SHELL=`*last-field-of-passwd-entry*
`MAIL=/var/mail/`*your-login-name*
`TZ=`*timezone-specification*

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 in 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. `login` understands simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such characters as spaces and tabs.

FILES

`/var/adm/utmp`	accounting
`/var/adm/wtmp`	accounting
`/var/mail/`*your-name*	mailbox for user *your-name*
`/var/adm/loginlog`	record of failed login attempts
`/etc/default/login`	environment variables
`/etc/motd`	message-of-the-day
`/etc/passwd`	password file
`/etc/profile`	system profile
`.profile`	user's login profile
`/var/adm/lastlog`	time of last login

SEE ALSO

`mail`(1), `newgrp`(1M), `sh`(1), `su`(1M), `loginlog`(4), `passwd`(4), `profile`(4), `environ`(5).

DIAGNOSTICS

`login incorrect` if the user name or the password cannot be matched.
`No shell, cannot open password file,` or `no directory: consult a system engineer.`
`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 a shell other than the initial shell.

NOTES

This command should only be used for ASCII terminals.

NAME

`logins` - list user and system login information

SYNOPSIS

`logins` [`-dmopstuxa`] [`-g groups`] [`-l` *logins*]

DESCRIPTION

This command displays information on user and system logins. Contents of the output is controlled by the command options and can include the following: user or system login, user id number, `/etc/passwd` account field value (user name or other information), primary group name, primary group id, multiple group names, multiple group ids, home directory, login shell, and four password aging parameters. The default information is the following: login id, user id, primary group name, primary group id and the account field value from `/etc/passwd`. Output is sorted by user id, displaying system logins followed by user logins.

`-d` Selects logins with duplicate uids.

`-m` Displays multiple group membership information.

`-o` Formats output into one line of colon-separated fields.

`-p` Selects logins with no passwords.

`-s` Selects all system logins.

`-t` Sorts output by login instead of by uid.

`-u` Selects all user logins.

`-x` Prints an extended set of information about each selected user. The extended information includes home directory, login shell and password aging information, each displayed on a separate line. The password information consists of password status (PS for passworded, NP for no password or LK for locked). If the login is passworded, status is followed by the date the password was last changed, the number of days required between changes, and the number of days allowed before a change is required. The password aging information shows the time interval that the user will receive a password expiration warning message (when logging on) before the password expires.

`-a` Adds two password expiration fields to the display. The fields show how many days a password can remain unused before it automatically becomes inactive and the date that the password will expire.

`-g` Selects all users belonging to `group`, sorted by login. Multiple groups can be specified as a comma-separated list.

`-l` Selects the requested login. Multiple logins can be specified as a comma-separated list.

NOTES

Options may be used together. If so, any login matching any criteria will be displayed. When the `-l` and `-g` options are combined, a user will only be listed once, even if they belong to more than one of the selected groups.

NAME

logname - get login name

SYNOPSIS

logname

DESCRIPTION

logname returns the name of the user running the process.

FILES

/etc/profile

SEE ALSO

env(1), login(1), cuserid(3C), environ(5).

NAME

look - find words in the system dictionary or lines in a sorted list

SYNOPSIS

/usr/ucb/look [-d] [-f] [-t*c*] *string* [*filename*]

DESCRIPTION

The look command consults a sorted *filename* and prints all lines that begin with *string*.

If no *filename* is specified, look uses /usr/ucblib/dict/words with collating sequence -df.

The following options are available:

-d Dictionary order. Only letters, digits, TAB and SPACE characters are used in comparisons.

-f Fold case. Upper case letters are not distinguished from lower case in comparisons.

-t*c* Set termination character. All characters to the right of *c* in *string* are ignored.

FILES

/usr/ucblib/dict/words

SEE ALSO

grep(1), sort(1).

NAME

lookbib - find references in a bibliographic database

SYNOPSIS

/usr/ucb/lookbib *database*

DESCRIPTION

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a '%', followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with '%'. See addbib.

lookbib uses an inverted index made by indxbib to find sets of bibliographic references. It reads keywords typed after the '>' prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another '>' prompt.

It is possible to search multiple databases, as long as they have a common index made by indxbib. In that case, only the first argument given to indxbib is specified to lookbib.

If lookbib does not find the index files (the .i[abc] files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a .ig suffix, suitable for use with fgrep (see grep). lookbib then uses this fgrep file to find references. This method is simpler to use, but the .ig file is slower to use than the .i[abc] files, and does not allow the use of multiple reference files.

FILES

*.ia
*.ib index files
*.ic

*.ig reference file

SEE ALSO

addbib(1), grep(1), indxbib(1), refer(1), roffbib(1), sortbib(1).

NOTES

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

lorder - find ordering relation for an object library

SYNOPSIS

lorder *file* . . .

DESCRIPTION

The input is one or more object or library archive *file*s [see ar(1)]. The standard output is a list of pairs of object file or archive member names; 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. Note that the link editor ld is capable of multiple passes over an archive in the portable archive format [see ar(4)] and does not require that lorder be used when building an archive. The usage of the lorder command may, however, allow for a 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

TMPDIR/*symref	temporary files
TMPDIR/*symdef	temporary files
TMPDIR	usually /var/tmp but can be redefined by setting the environment variable TMPDIR [see tempnam in tmpnam(3S)].

SEE ALSO

ar(1), ld(1), tsort(1), tempnam(3S), tmpname(3S), ar(4)

NOTES

lorder will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be .o.

NAME

lp, cancel - send/cancel requests to an LP print service

SYNOPSIS

lp [*printing-options*] *files*
lp -i *request-IDs printing-options*
cancel [*request-IDs*] [*printers*]
cancel -u *login_IDs* [*printers*]

DESCRIPTION

The first form of the lp shell command arranges for the named files and associated information (collectively called a *request*) to be printed. If no file names are specified on the shell command line, the standard input is assumed. The standard input may be specified along with named *files* on the shell command line by listing the file name(s) and specifying - for the standard input. The *files* will be printed in the order in which they appear on the shell command line.

The second form of lp is used to change the options for a request. The print request identified by the *request-ID* is changed according to the printing options specified with this shell command. The printing options available are the same as those with the first form of the lp shell command. If the request has finished printing, the change is rejected. If the request is already printing, it will be stopped and restarted from the beginning (unless the -P option has been given).

lp associates a unique *request-ID* with each request and prints it on the standard output. This *request-ID* can be used later when canceling or changing a request, or when determining its status. [See the section on cancel for details about canceling a request, the previous paragraph for an explanation of how to change a request, and lpstat(1) for information about checking the status of a print request.]

Sending a Print Request

The first form of the lp command is used to send a print request to a particular printer or group of printers.

Options to lp must always precede file names, but may be listed in any order. The following options are available for lp:

-c Make copies immediately of the *files* to be printed when lp is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the -c option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that if the -c option is not specified, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

-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 (unavailability of printers, file space limitations, and so on) requests for specific destinations may not be accepted [see 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)].

-f *form-name* [-d any]
Print the request on the form *form-name*. The LP print service ensures that the form is mounted on the printer. If *form-name* is requested with a printer destination that cannot support the form, the request is rejected. If *form-name* has not been defined for the system, or if the user is not allowed to use the form, the request is rejected [see lpforms(1M)]. When the -d any option is given, the request is printed on any printer that has the requested form mounted and can handle all other needs of the print request.

-H *special-handling*
Print the request according to the value of *special-handling*. Acceptable values for *special-handling* are hold, resume, and immediate, as defined below:

hold Don't print the request until notified. If printing has already begun, stop it. Other print requests will go ahead of a held request until it is resumed.

resume Resume a held request. If it had been printing when held, it will be the next request printed, unless subsequently bumped by an immediate request.

immediate (Available only to LP administrators)
Print the request next. If more than one request is assigned immediate, the requests are printed in the reverse order queued. If a request is currently printing on the desired printer, you have to put it on hold to allow the immediate request to print.

-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 *options*. Several such *options* may be collected by specifying the -o keyletter more than once (-o *option*$_1$ -o *option*$_2$ ... -o *option*$_n$), or by specifying a list of options with more than one -o keyletter (i.e., -o *option*$_1$, *option*$_2$, ... *option*$_n$). The standard interface recognizes the following options:

nobanner Do not print a banner page with this request. (The administrator can disallow this option at any time.)

nofilebreak
Do not insert a form feed between the files given, if submitting a job to print more than one file.

length=*scaled-decimal-number*
Print this request with pages *scaled-decimal-number* lines long. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter "i" for inches, or the letter "c" for centimeters. For length or width settings, an unscaled number indicates lines or columns; for line

pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with "i"). For example, `length=66` indicates a page length of 66 lines, `length=11i` indicates a page length of 11 inches, and `length=27.94c` indicates a page length of 27.94 centimeters.

This option cannot be used with the `-f` option.

`width=`*scaled-decimal-number*
Print this request with page-width set to *scaled-decimal-number* columns wide. (See the explanation of *scaled-decimal-numbers* in the discussion of `length`, above.) This option cannot be used with the `-f` option.

`lpi=`*scaled-decimal-number*
Print this request with the line pitch set to *scaled-decimal-number* lines per inch. This option cannot be used with the `-f` option.

`cpi=`*scaled-decimal-number*
Print this request with the character pitch set to *scaled-decimal-number* characters per inch. Character pitch can also be set to `pica` (representing 10 columns per inch) or `elite` (representing 12 columns per inch), or it can be `compressed` (representing as many columns as a printer can handle). There is no standard number of columns per inch for all printers; see the Terminfo database [`terminfo`(4)] for the default character pitch for your printer.

This option cannot be used with the `-f` option.

`stty=`*stty-option-list*
A list of options valid for the `stty` command; enclose the list with quotes if it contains blanks.

`-P` *page-list* Print the pages specified in *page-list*. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

The *page-list* may consist of range(s) of numbers, single page numbers, or a combination of both. The pages will be printed in ascending order.

`-q` *priority-level*
Assign this request *priority-level* in the printing queue. The values of *priority-level* range from 0, the highest priority, to 39, the lowest priority. If a priority is not specified, the default for the print service is used, as assigned by the system administrator.

`-s` Suppress messages from `lp` such as those that begin with `request id is`.

`-S` *character-set* [`-d any`]

`-S` *print-wheel* [`-d any`]
: Print this request using the specified *character-set* or *print-wheel*. If a form was requested and it requires a character set or print wheel other than the one specified with the `-S` option, the request is rejected.

  For printers that take print wheels: if the print wheel specified is not one listed by the administrator as acceptable for the printer specified in this request, the request is rejected unless the print wheel is already mounted on the printer.

  For printers that use selectable or programmable character sets: if the *character-set* specified is not one defined in the Terminfo database for the printer [see `terminfo`(4)], or is not an alias defined by the administrator, the request is rejected.

  When the `-d any` option is used, the request is printed on any printer that has the print wheel mounted or any printer that can select the character set, and that can handle any other needs of the request.

`-t` *title*
: Print *title* on the banner page of the output. The default is no title.

`-T` *content-type* [`-r`]
: Print the request on a printer that can support the specified *content-type*. If no printer accepts this type directly, a filter will be used to convert the content into an acceptable type. If the `-r` option is specified, a filter will not be used. If `-r` is specified, and no printer accepts the *content-type* directly, the request is rejected. If the *content-type* is not acceptable to any printer, either directly or with a filter, the request is rejected.

`-w`
: Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

`-y` *mode-list*
: Print this request according to the printing modes listed in *mode-list*. The allowed values for *mode-list* are locally defined. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

Canceling a Print Request

The `cancel` command cancels requests for print jobs made with the `lp` command. To cancel a job, specify one of the following arguments: the *request-ID* for it (as returned by the `lp` command); the name of the printer handling it; or the *login-ID* of the user who requested it. A printer class is not a valid argument.

Users without special privileges can cancel only requests associated with their own login IDs.

NOTES

Printers for which requests are not being accepted will not be considered when the `lp` command is run and the destination is `any`. (Use the `lpstat -a` command to see which printers are accepting requests.) On the other hand, if (1) a request is destined for a class of printers and (2) the class itself is accepting requests, then *all* printers in the class will be considered, regardless of their acceptance status.

For printers that take mountable print wheels or font cartridges, if you do not specify a particular print wheel or font with the `-S` option, whichever one happens to be mounted at the time your request is printed will be used. Use the `lpstat -p` *printer* `-l` command to see which print wheels are available on a particular printer, or the `lpstat -S -l` command to find out what print wheels are available and on which printers. For printers that have selectable character sets, you will get the standard character set if you don't use the `-S` option.

FILES

`/var/spool/lp/*`

SEE ALSO

`accept`(1M), `enable`(1), `lpstat`(1), `lpadmin`(1M), `lpfilter`(1M), `lpforms`(1M), `lpsched`(1M), `lpsystem`(1M), `lpusers`(1M), `mail`(1), `terminfo`(4).

NAME

lpadmin - configure the LP print service

SYNOPSIS

lpadmin -p [*printer*] *options*
lpadmin -x *dest*
lpadmin -d [*dest*]
lpadmin -S *print-wheel* -A *alert-type* [-W *minutes*] [-Q *requests*]

DESCRIPTION

lpadmin configures the LP print service by defining printers and devices. It is used to add and change printers, to remove printers from the service, to set or change the system default destination, to define alerts for printer faults, and to mount print wheels.

Adding or Changing a Printer

The first form of the lpadmin command (lpadmin -p [*printer*] *options*) is used to configure a new printer or to change the configuration of an existing printer. If the argument given for *printer* is not a valid printer name, *printer* defaults to all. The following *options* may appear in any order.

-A *alert-type* [-W *minutes*]
The -A option is used to define an alert to inform the administrator when a printer fault is detected, and periodically thereafter, until the printer fault is cleared by the administrator. The *alert-types* are:

mail Send the alert message via mail [see mail(1)] to the administrator.

write Write the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is chosen arbitrarily.

quiet Do not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the fault has been cleared and printing resumes, messages will again be sent when another fault occurs with the printer.

none Do not send messages; any existing alert definition for the printer will be removed. No alert will be sent when the printer faults until a different alert-type (except quiet) is used.

shell-command
Run the *shell-command* each time the alert needs to be sent. The shell command should expect the message in standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the mail and write values for this option are equivalent to the values mail *user-name* and write *user-name* respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command unless he or she has used the su command to change to another user ID. If the su command has been used to change the user ID, then the *user-name* for the new ID is used.

`list` Display the type of the alert for the printer fault. No change is made to the alert.

The message sent appears as follows:

> The printer *printer* has stopped printing for the reason given below. Fix the problem and bring the printer back on line. Printing has stopped, but will be restarted in a few minutes; issue an enable command if you want to restart sooner. Unless someone issues a change request
>
> `lp -i` *request-id* `-P ...`
>
> to change the page list to print, the current request will be reprinted from the beginning.
>
> The reason(s) it stopped (multiple reasons indicate reprinted attempts):
>
> *reason*

The LP print service can detect printer faults only through an adequate fast filter and only when the standard interface program or a suitable customized interface program is used. Furthermore, the level of recovery after a fault depends on the capabilities of the filter.

If the *printer* is `all` (the default), the alerting defined in this command applies to all existing printers.

If the `-W` option is not used to arrange fault alerting for *printer*, the default procedure is to mail one message to the administrator of *printer* per fault. This is equivalent to specifying `-W once` or `-W 0`. If *minutes* is a number greater than zero, an alert will be sent at intervals specified by *minutes*.

`-c` *class*
Insert *printer* into the specified *class*. *Class* will be created if it does not already exist.

`-D` *comment*
Save this *comment* for display whenever a user asks for a full description of *printer* [see `lpstat`(1)]. The LP print service does not interpret this comment.

`-e` *printer*$_1$
Copy the interface program of an existing *printer*$_1$ to be the interface program for *printer*. (Options `-i` and `-m` may not be specified with this option.)

`-F` *fault-recovery*
This option specifies the recovery to be used for any print request that is stopped because of a printer fault, according to the value of *fault-recovery*:

`continue`
Continue printing on the top of the page where printing stopped. This requires a filter to wait for the fault to clear before automatically continuing.

`beginning`
Start printing the request again from the beginning.

`wait` Disable printing on *printer* and wait for the administrator or a user to enable printing again.

During the wait the administrator or the user who submitted the stopped print request can issue a change request that specifies where printing should resume. (See the `-i` option of the `lp` command.) If no change request is made before printing is enabled, printing will resume at the top of the page where stopped, if the filter allows; otherwise, the request will be printed from the beginning.

`-f allow:`*form-list*
`-f deny:`*form-list*
Allow or deny the forms in *form-list* to be printed on *printer*. By default no forms are allowed on a new printer.

For each printer, the LP print service keeps two lists of forms: an "allow-list" of forms that may be used with the printer, and a "deny-list" of forms that may not be used with the printer. With the `-f allow` option, the forms listed are added to the allow-list and removed from the deny-list. With the `-f deny` option, the forms listed are added to the deny-list and removed from the allow-list.

If the allow-list is not empty, only the forms in the list may be used on the printer, regardless of the contents of the deny-list. If the allow-list is empty, but the deny-list is not, the forms in the deny-list may not be used with the printer. All forms can be excluded from a printer by specifying `-f deny:all`. All forms can be used on a printer (provided the printer can handle all the characteristics of each form) by specifying `-f allow:all`.

The LP print service uses this information as a set of guidelines for determining where a form can be mounted. Administrators, however, are not restricted from mounting a form on any printer. If mounting a form on a particular printer is in disagreement with the information in the allow-list or deny-list, the administrator is warned but the mount is accepted. Nonetheless, if a user attempts to issue a print or change request for a form and printer combination that is in disagreement with the information, the request is accepted only if the form is currently mounted on the printer. If the form is later unmounted before the request can print, the request is canceled and the user is notified by mail.

If the administrator tries to specify a form as acceptable for use on a printer that doesn't have the capabilities needed by the form, the command is rejected.

Note the other use of `-f`, with the `-M` option, below.

`-h` Indicate that the device associated with the printer is hardwired. If neither of the mutually exclusive options, `-h` and `-l`, is specified, this option is assumed.

`-I` *content-type-list*
Allow *printer* to handle print requests with the content types listed in a *content-type-list*. If the list includes names of more than one type, the names must be separated by commas or blank spaces. (If they are separated by blank spaces, the entire list must be enclosed in double quotes.)

The type `simple` is recognized as the default content type for files in the UNIX system. A `simple` type of file is a data stream containing only printable ASCII characters and the following control characters.

Control Character	Octal Value	Meaning
backspace	10_8	move back one character, except at beginning of line
tab	11_8	move to next tab stop
linefeed (newline)	12_8	move to beginning of next line
form feed	14_8	move to beginning of next page
carriage return	15_8	move to beginning of current line

To prevent the print service from considering `simple` a valid type for the printer, specify either an explicit value (such as the printer type) in the *content-type-list*, or an empty list. If you do want `simple` included along with other types, you must include `simple` in the *content-type-list*.

Except for `simple`, each *content-type* name is freely determined by the administrator. If the printer type is specified by the `-T` option, then the printer type is implicitly considered to be also a valid content type.

`-i` *interface*
Establish a new interface program for *printer*. *Interface* is the pathname of the new program. (The `-e` and `-m` options may not be specified with this option.)

`-l` Indicate that the device associated with *printer* is a login terminal. The LP scheduler (`lpsched`) disables all login terminals automatically each time it is started. (The `-h` option may not be specified with this option.)

`-M -f` *form-name* [`-a` [`-o filebreak`]]
Mount the form *form-name* on *printer*. Print requests that need the pre-printed form *form-name* will be printed on *printer*. If more than one printer has the form mounted and the user has specified `any` (with the `-d` option of the `lp` command) as the printer destination, then the print request will be printed on the one printer that also meets the other needs of the request.

The page length and width, and character and line pitches needed by the form are compared with those allowed for the printer, by checking the capabilities in the `terminfo` database for the type of printer. If the form requires attributes that are not available with the printer, the administrator is warned but the mount is accepted. If the form lists a print wheel as mandatory, but the print wheel mounted on the printer is different, the administrator is also warned but the mount is accepted.

If the -a option is given, an alignment pattern is printed, preceded by the same initialization of the physical printer that precedes a normal print request, with one exception: no banner page is printed. Printing is assumed to start at the top of the first page of the form. After the pattern is printed, the administrator can adjust the mounted form in the printer and press return for another alignment pattern (no initialization this time), and can continue printing as many alignment patterns as desired. The administrator can quit the printing of alignment patterns by typing q.

If the -o filebreak option is given, a formfeed is inserted between each copy of the alignment pattern. By default, the alignment pattern is assumed to correctly fill a form, so no formfeed is added.

A form is "unmounted" either by mounting a new form in its place or by using the -f none option. By default, a new printer has no form mounted.

Note the other use of -f without the -M option above.

-M -S *print-wheel*
Mount the *print-wheel* on *printer*. Print requests that need the *print-wheel* will be printed on *printer*. If more than one printer has *print-wheel* mounted and the user has specified any (with the -d option of the lp command) as the printer destination, then the print request will be printed on the one printer that also meets the other needs of the request.

If the *print-wheel* is not listed as acceptable for the printer, the administrator is warned but the mount is accepted. If the printer does not take print wheels, the command is rejected.

A print wheel is "unmounted" either by mounting a new print wheel in its place or by using the option -S none. By default, a new printer has no print wheel mounted.

Note the other uses of the -S option without the -M option described below.

-m *model*
Select *model* interface program, provided with the LP print service, for the printer. (Options -e and -i may not be specified with this option.)

-o *printing-option*
Each -o option in the list below is the default given to an interface program if the option is not taken from a preprinted form description or is not explicitly given by the user submitting a request [see lp(1)]. The only -o options that can have defaults defined are listed below.

length=*scaled-decimal-number*
width=*scaled-decimal-number*
cpi=*scaled-decimal-number*
lpi=*scaled-decimal-number*
stty='*stty-option-list*'

The term "scaled-decimal-number" refers to a non-negative number used to indicate a unit of size. The type of unit is shown by a "trailing" letter attached to the number. Three types of scaled decimal numbers can be used with the LP print service: numbers that show sizes in centimeters (marked with a trailing c); numbers that show sizes in inches (marked with a trailing i); and numbers that show sizes in units appropriate to use (without a

trailing letter), that is, lines, characters, lines per inch, or characters per inch.

The first four default option values must agree with the capabilities of the type of physical printer, as defined in the `terminfo` database for the printer type. If they do not, the command is rejected.

The *stty-option-list* is not checked for allowed values, but is passed directly to the `stty` program by the standard interface program. Any error messages produced by `stty` when a request is processed (by the standard interface program) are mailed to the user submitting the request.

For each printing option not specified, the defaults for the following attributes are defined in the `terminfo` entry for the specified printer type.

```
length
width
cpi
lpi
```

The default for `stty` is

```
stty='9600 cs8 -cstopb -parenb ixon
     -ixany opost -olcuc onlcr -ocrnl -onocr
     -onlret -ofill nl0 cr0 tab0 bs0 vt0 ff0'
```

You can set any of the `-o` options to the default values (which vary for different types of printers), by typing them without assigned values, as follows:

```
length=
width=
cpi=
lpi=
stty=
```

`-o nobanner`
Allow a user to submit a print request specifying that no banner page be printed.

`-o banner`
Force a banner page to be printed with every print request, even when a user asks for no banner page. This is the default; you must specify `-o nobanner` if you want to allow users to be able to specify `-o nobanner` with the `lp` command.

`-r` *class*
Remove *printer* from the specified *class*. If *printer* is the last member of *class*, then *class* will be removed.

`-S` *list*
Allow either the print wheels or aliases for character sets named in *list* to be used on the printer.

If the printer is a type that takes print wheels, then *list* is a comma or space separated list of print wheel names. (Enclose the list with quotes if it contains blanks.) These will be the only print wheels considered mountable on the printer. (You can always force a different print wheel to be mounted, however.) Until the option is used to specify a list, no print wheels will be

considered mountable on the printer, and print requests that ask for a particular print wheel with this printer will be rejected.

If the printer is a type that has selectable character sets, then *list* is a comma or blank separated list of character set name "mappings" or aliases. (Enclose the list with quotes if it contains blanks.) Each "mapping" is of the form

known-name=alias

The *known-name* is a character set number preceded by `cs` (such as `cs3` for character set three) or a character set name from the `Terminfo` database entry `csnm`. [See `terminfo`(4) in the reference manual.] If this option is not used to specify a list, only the names already known from the Terminfo database or numbers with a prefix of `cs` will be acceptable for the printer.

If *list* is the word `none`, any existing print wheel lists or character set aliases will be removed.

Note the other uses of the `-S` with the `-M` option described above.

`-s` *system-name*[`!`*printer-name*]
Make a remote printer (one that must be accessed through another system) accessible to users on your system. *System-name* is the name of the remote system on which the remote printer is located; it must be listed in the systems table (`/etc/lp/Systems`). *Printer-name* is the name used on the remote system for that printer. For example, if you want to access $printer_1$ on $system_1$ and you want it called $printer_2$ on your system, enter `-p` $printer_2$ `-s` $system_1$`!`$printer_1$

`-T` *printer-type-list*
Identify the printer as being of one or more *printer-type*s. Each *printer-type* is used to extract data from the `terminfo` database; this information is used to initialize the printer before printing each user's request. Some filters may also use a *printer-type* to convert content for the printer. If this option is not used, the default *printer-type* will be `unknown`; no information will be extracted from `terminfo` so each user request will be printed without first initializing the printer. Also, this option must be used if the following are to work: `-o cpi`, `-o lpi`, `-o width`, and `-o length` options of the `lpadmin` and `lp` commands, and the `-S` and `-f` options of the `lpadmin` command.

If the *printer-type-list* contains more than one type, then the *content-type-list* of the `-I` option must either be specified as `simple`, as empty (`-I ""`), or not specified at all.

`-u allow:`*login-ID-list*
`-u deny:`*login-ID-list*
Allow or deny the users in *login-ID-list* access to the printer. By default all users are allowed on a new printer. The *login-ID-list* argument may include any or all of the following constructs:

login-ID	a user on any system
system-name`!`*login-ID*	a user on system *system-name*

system-name!all	all users on system *system-name*
all!*login-ID*	a user on all systems
all	all users on all systems

For each printer the LP print service keeps two lists of users: an "allow-list" of people allowed to use the printer, and a "deny-list" of people denied access to the printer. With the -u allow option, the users listed are added to the allow-list and removed from the deny-list. With the -u deny option, the users listed are added to the deny-list and removed from the allow-list.

If the allow-list is not empty, only the users in the list may use the printer, regardless of the contents of the deny-list. If the allow-list is empty, but the deny-list is not, the users in the deny-list may not use the printer. All users can be denied access to the printer by specifying -u deny:all. All users may use the printer by specifying -u allow:all.

-U *dial-info*

The -U option allows your print service to access a remote printer. (It does not enable your print service to access a remote printer service.) Specifically, -U assigns the "dialing" information *dial-info* to the printer. *Dial-info* is used with the dial routine to call the printer. Any network connection supported by the Basic Networking Utilities will work. *Dial-info* can be either a phone number for a modem connection, or a system name for other kinds of connections. Or, if -U direct is given, no dialing will take place, because the name direct is reserved for a printer that is directly connected. If a system name is given, it is used to search for connection details from the file /etc/uucp/Systems or related files. The Basic Networking Utilities are required to support this option. By default, -U direct is assumed.

-v *device*

Associate a *device* with *printer*. *Device* is the path name of a file that is writable by lp. Note that the same *device* can be associated with more than one printer.

Restrictions

When creating a new printer, one of three options (-v, -U, or -s) must be supplied. In addition, only one of the following may be supplied: -e, -i, or -m; if none of these three options is supplied, the model standard is used. The -h and -l options are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters A-Z, a-z, 0-9 and _ (underscore). If -s is specified, the following options are invalid: -A, -e, -F, -h, -i, -l, -M, -m, -o, -U, -v, and -W.

Removing a Printer Destination

The -x *dest* option removes the destination *dest* (a printer or a class), from the LP print service. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. If *dest* is all, all printers and classes are removed. No other *options* are allowed with -x.

Setting/Changing the System Default Destination

The -d [*dest*] option makes *dest*, an existing printer or class, the new system default destination. If *dest* is not supplied, then there is no system default destination. No other *options* are allowed with -d.

Setting an Alert for a Print Wheel

-S *print-wheel* -A *alert-type* [-W *minutes*] [-Q *requests*]

The -S *print-wheel* option is used with the -A *alert-type* option to define an alert to mount the print wheel when there are jobs queued for it. If this command is not used to arrange alerting for a print wheel, no alert will be sent for the print wheel. Note the other use of -A, with the -p option, above.

The *alert-types* are:

`mail` Send the alert message via the `mail` command to the administrator.

`write` Write the message, via the `write` command, to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is arbitrarily chosen.

`quiet` Do not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the *print-wheel* has been mounted and subsequently unmounted, messages will again be sent when the number of print requests reaches the threshold specified by the -Q option.

`none` Do not send messages until the -A option is given again with a different *alert-type* (other than `quiet`).

shell-command
Run the *shell-command* each time the alert needs to be sent. The shell command should expect the message in standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the `mail` and `write` values for this option are equivalent to the values `mail` *user-name* and `write` *user-name* respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command unless he or she has used the `su` command to change to another user ID. If the `su` command has been used to change the user ID, then the *user-name* for the new ID is used.

`list` Display the type of the alert for the print wheel on standard output. No change is made to the alert.

The message sent appears as follows:

```
The print wheel print-wheel needs to be mounted
on the printer(s):
printer (integer1 requests)
integer2 print requests await this print wheel.
```

The printers listed are those that the administrator had earlier specified were candidates for this print wheel. The number $integer_1$ listed next to each printer is the number of requests eligible for the printer. The number $integer_2$ shown after the printer list is the total number of requests awaiting the print wheel. It will be less than the sum of the other numbers if some requests can be handled by more than one printer.

If the *print-wheel* is `all`, the alerting defined in this command applies to all print wheels already defined to have an alert.

If the `-W` option is not given, the default procedure is that only one message will be sent per need to mount the print wheel. Not specifying the `-W` option is equivalent to specifying `-W once` or `-W 0`. If *minutes* is a number greater than zero, an alert will be sent at intervals specified by *minutes*.

If the `-Q` option is also given, the alert will be sent when a certain number (specified by the argument *requests*) of print requests that need the print wheel are waiting. If the `-Q` option is not given, or *requests* is 1 or the word `any` (which are both the default), a message is sent as soon as anyone submits a print request for the print wheel when it is not mounted.

FILES

`/var/spool/lp/*`
`/etc/lp`

SEE ALSO

`accept`(1M), `lpsched`(1M), and `lpsystem`(1M).
`enable`(1), `lp`(1), `lpstat`(1), and `stty`(1), `dial`(3C), `terminfo`(4).

NAME

lpc - line printer control program

SYNOPSIS

/usr/ucb/lpc [*command* [*parameter*...]]

DESCRIPTION

lpc controls the operation of the printer, or of multiple printers. lpc commands can be used to start or stop a printer, disable or enable a printer's spooling queue, rearrange the order of jobs in a queue, or display the status of each printer—along with its spooling queue and printer daemon.

With no arguments, lpc runs interactively, prompting with 'lpc>'. If arguments are supplied, lpc interprets the first as a *command* to execute; each subsequent argument is taken as a *parameter* for that command. The standard input can be redirected so that lpc reads commands from a file.

Commands may be abbreviated to an unambiguous substring. Note: the *printer* parameter is specified just by the name of the printer (as lw), not as you would specify it to lpr(1) or lpq(1) (not as -Plw).

? [*command*]...

help [*command*]...

Display a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort [all | [*printer*...]]

Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by lpr(1)) for the specified printers. The abort command can only be used by the privileged user.

clean [all | [*printer*...]]

Remove all files created in the spool directory by the daemon from the specified printer queue(s) on the local machine. The clean command can only be used by the privileged user.

disable [all | [*printer*...]]

Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by lpr(1). The disable command can only be used by the privileged user.

down [all | [*printer*...]] [*message*]

Turn the specified printer queue off, disable printing and put *message* in the printer status file. The message does not need to be quoted, the remaining arguments are treated like echo(1). This is normally used to take a printer down and let others know why (lpq(1) indicates that the printer is down, as does the status command).

enable [all | [*printer*...]]

Enable spooling on the local queue for the listed printers, so that lpr(1) can put new jobs in the spool queue. The enable command can only be used by the privileged user.

exit
quit Exit from lpc.

restart [all | [*printer* ...]]
Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. This command can be run by any user.

start [all | [*printer* ...]]
Enable printing and start a spooling daemon for the listed printers. The start command can only be used by the privileged user.

status [all | [*printer* ...]]
Display the status of daemons and queues on the local machine. This command can be run by any user.

stop [all | [*printer* ...]]
Stop a spooling daemon after the current job completes and disable printing. The stop command can only be used by the privileged user.

topq *printer* [*job#* ...] [*user* ...]
Move the print job(s) specified by *job#* or those job(s) belonging to *user* to the top (head) of the printer queue. The topq command can only be used by the privileged user.

up [all | [*printer* ...]] Enable everything and start a new printer daemon. Undoes the effects of down.

FILES

/var/spool/lp/*
/var/spool/lp/system/pstatus

SEE ALSO

lpq(1), lpr(1), lprm(1), echo(1), lpsched(1M).

DIAGNOSTICS

?Ambiguous command
The abbreviation you typed matches more than one command.

?Invalid command
You typed a command or abbreviation that was not recognized.

?Privileged command
You used a command can be executed only by the privileged user.

lpc: *printer*: unknown printer to the print service
The printer was not found in the System V LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use 'lptstat -p' to find the reason.

lpc: error on opening queue to spooler
The connection to lpsched on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon /usr/lib/lp/lpsched is running.

lpc: Can't send message to LP print service

`lpc: Can't receive message from LP print service`
These indicate that the LP print service has been stopped. Get help from the system administrator.

`lpc: Received unexpected message from LP print service`
It is likely there is an error in this software. Get help from system administrator.

NAME

lpfilter – administer filters used with the LP print service

SYNOPSIS

```
lpfilter -f filter-name -F path-name
lpfilter -f filter-name -
lpfilter -f filter-name -i
lpfilter -f filter-name -x
lpfilter -f filter-name -l
```

DESCRIPTION

The lpfilter command is used to add, change, delete, and list a filter used with the LP print service. These filters are used to convert the content type of a file to a content type acceptable to a printer. One of the following options must be used with the lpfilter command: -F *path-name* (or - for standard input) to add or change a filter; -i to reset an original filter to its factory setting; -x to delete a filter; or -l to list a filter description.

The argument all can be used instead of a *filter-name* with any of these options. When all is specified with the -F or - option, the requested change is made to all filters. Using all with the -i option has the effect of restoring to their original settings all filters for which predefined settings were initially available. Using the all argument with the -x option results in all filters being deleted, and using it with the -l option produces a list of all filters.

Adding or Changing a Filter

The filter named in the -f option is added to the filter table. If the filter already exists, its description is changed to reflect the new information in the input.

The filter description is taken from the *path-name* if the -F option is given, or from the standard input if the - option is given. One of the two must be given to define or change a filter. If the filter named is one originally delivered with the LP print service, the -i option will restore the original filter description.

When an existing filter is changed with the -F or - option, items that are not specified in the new information are left as they were. When a new filter is added with this command, unspecified items are given default values. (See below.)

Filters are used to convert the content of a request into a data stream acceptable to a printer. For a given print request, the LP print service will know the following: the type of content in the request, the name of the printer, the type of the printer, the types of content acceptable to the printer, and the modes of printing asked for by the originator of the request. It will use this information to find a filter or a pipeline of filters that will convert the content into a type acceptable to the printer.

Below is a list of items that provide input to this command, and a description of each item. All lists are comma or space separated.

```
Input types: content-type-list
Output types: content-type-list
Printer types: printer-type-list
Printers: printer-list
Filter type: filter-type
Command: shell-command
Options: template-list
```

`Input types` This gives the types of content that can be accepted by the filter. (The default is `any`.)

`Output types`
This gives the types of content that the filter can produce from any of the input content types. (The default is `any`.)

`Printer types`
This gives the type of printers for which the filter can be used. The LP print service will restrict the use of the filter to these types of printers. (The default is `any`.)

`Printers` This gives the names of the printers for which the filter can be used. The LP print service will restrict the use of the filter to just the printers named. (The default is `any`.)

`Filter type` This marks the filter as a `slow` filter or a `fast` filter. Slow filters are generally those that take a long time to convert their input. They are run unconnected to a printer, to keep the printers from being tied up while the filter is running. If a listed printer is on a remote system, the filter type for it must have the value `slow`. Fast filters are generally those that convert their input quickly, or those that must be connected to the printer when run. These will be given to the interface program to run connected to the physical printer.

`Command` This specifies the program to run to invoke the filter. The full program pathname as well as fixed options must be included in the *shell-command*; additional options are constructed, based on the characteristics of each print request and on the `Options` field. A command must be given for each filter.

The command must accept a data stream as standard input and produce the converted data stream on its standard output. This allows filter pipelines to be constructed to convert data not handled by a single filter.

`Options` This is a comma separated list of templates used by the LP print service to construct options to the filter from the characteristics of each print request listed in the table later.

In general, each template is of the following form:

keyword pattern = replacement

The *keyword* names the characteristic that the template attempts to map into a filter specific option; each valid *keyword* is listed in the table below. A *pattern* is one of the following: a literal pattern of one of the forms listed in the table, a single asterisk (`*`), or a regular expression. If *pattern* matches the value of the characteristic, the template fits and is used to generate a filter specific option. The *replacement* is what will be used as the option.

Regular expressions are the same as those found in the ed(1) or vi(1) commands. This includes the \(...\) and *n* constructions, which can be used to extract portions of the *pattern* for copying into the *replacement*, and the &, which can be used to copy the entire *pattern* into the *replacement*.

The *replacement* can also contain a *; it too, is replaced with the entire *pattern*, just like the & of ed(1).

lp Option	Characteristic	*keyword*	Possible *patterns*
-T	Content type (input)	INPUT	*content-type*
N/A	Content type (output)	OUTPUT	*content-type*
N/A	Printer type	TERM	*printer-type*
-d	Printer name	PRINTER	*printer-name*
-f, -o cpi=	Character pitch	CPI	*integer*
-f, -o lpi=	Line pitch	LPI	*integer*
-f, -o length=	Page length	LENGTH	*integer*
-f, -o width=	Page width	WIDTH	*integer*
-P	Pages to print	PAGES	*page-list*
-S	Character set	CHARSET	*character-set-name*
	Print wheel	CHARSET	*print-wheel-name*
-f	Form name	FORM	*form-name*
-y	Modes	MODES	*mode*
-n	Number of copies	COPIES	*integer*

For example, the template

```
MODES landscape = -l
```

shows that if a print request is submitted with the -y landscape option, the filter will be given the option -l. As another example, the template

```
TERM * = -T *
```

shows that the filter will be given the option -T *printer-type* for whichever *printer-type* is associated with a print request using the filter.

As a last example, consider the template

```
MODES prwidth\=\(.*\) = -w\1
```

Suppose a user gives the command

```
lp -y prwidth=10
```

From the table above, the LP print service determines that the -y option is handled by a MODES template. The MODES template here works because the *pattern* prwidth\=\(.*\) matches the prwidth=10 given by the user. The *replacement* -w\1 causes the LP print service to generate the filter option -w10.

If necessary, the LP print service will construct a filter pipeline by concatenating several filters to handle the user's file and all the print options. (See sh(1) for a description of a pipeline.) If the print service constructs a filter pipeline, the INPUT and OUTPUT values used for each filter in the pipeline are the types of the input and

output for that filter, not for the entire pipeline.

Deleting a Filter

The `-x` option is used to delete the filter specified in *filter-name* from the LP filter table.

Listing a Filter Description

The `-l` option is used to list the description of the filter named in *filter-name*. If the command is successful, the following message is sent to standard output:

```
Input types: content-type-list
Output types: content-type-list
Printer types: printer-type-list
Printers: printer-list
Filter type: filter-type
Command: shell-command
Options: template-list
```

If the command fails, an error message is sent to standard error.

SEE ALSO

`lpadmin`(1M),
`lp`(1).

NAME

lpforms – administer forms used with the LP print service

SYNOPSIS

`lpforms -f` *form-name options*
`lpforms -f` *form-name* `-A` *alert-type* [`-Q` *minutes*] [`-W` *requests*]

DESCRIPTION

The `lpforms` command is used to administer the use of preprinted forms, such as company letterhead paper, with the LP print service. A form is specified by its *form-name*. Users may specify a form when submitting a print request [see `lp`(1)]. The argument `all` can be used instead of *form-name* with either of the command lines shown above. The first command line allows the administrator to add, change, and delete forms, to list the attributes of an existing form, and to allow and deny users access to particular forms. The second command line is used to establish the method by which the administrator is alerted that the form *form-name* must be mounted on a printer.

With the first `lpforms` command line, one of the following options must be used:

`-F` *pathname*	To add or change form *form-name*, as specified by the information in *pathname*
`-`	To add or change form *form-name*, as specified by the information from standard input
`-x`	To delete form *form-name* (this option must be used separately; it may not be used with any other option)
`-l`	To list the attributes of form *form-name*

Adding or Changing a Form

The `-F` *pathname* option is used to add a new form, *form-name*, to the LP print service, or to change the attributes of an existing form. The form description is taken from *pathname* if the `-F` option is given, or from the standard input if the `-` option is used. One of these two options must be used to define or change a form. *Pathname* is the path name of a file that contains all or any subset of the following information about the form.

`Page length:` *scaled – decimal – number*$_1$
`Page width:` *scaled – decimal – number*$_2$
`Number of pages:` *integer*
`Line pitch:` *scaled – decimal – number*$_3$
`Character pitch:` *scaled – decimal – number*$_4$
`Character set choice:` *character-set/print-wheel* [`mandatory`]
`Ribbon color:` *ribbon-color*
`Comment:`
comment
`Alignment pattern:` [*content-type*]
content

The term "scaled-decimal-number" refers to a non-negative number used to indicate a unit of size. The type of unit is shown by a "trailing" letter attached to the number. Three types of scaled decimal numbers can be used with the LP print

service: numbers that show sizes in centimeters (marked with a trailing `c`); numbers that show sizes in inches (marked with a trailing `i`); and numbers that show sizes in units appropriate to use (without a trailing letter), that is, lines, characters, lines per inch, or characters per inch.

Except for the last two lines, the above lines may appear in any order. The `Comment:` and *comment* items must appear in consecutive order but may appear before the other items, and the `Alignment pattern:` and the *content* items must appear in consecutive order at the end of the file. Also, the *comment* item may not contain a line that begins with any of the key phrases above, unless the key phrase is preceded with a > sign. Any leading > sign found in the *comment* will be removed when the comment is displayed. Case distinctions in the key phrases are ignored.

When this command is issued, the form specified by *form-name* is added to the list of forms. If the form already exists, its description is changed to reflect the new information. Once added, a form is available for use in a print request, except where access to the form has been restricted, as described under the `-u` option. A form may also be allowed to be used on certain printers only.

A description of each form attribute is below:

`Page length and Page Width`
: Before printing the content of a print request needing this form, the generic interface program provided with the LP print service will initialize the physical printer to handle pages *scaled – decimal – number*$_1$ long, and *scaled – decimal – number*$_2$ wide using the printer type as a key into the `terminfo` database.

The page length and page width will also be passed, if possible, to each filter used in a request needing this form.

`Number of pages`
: Each time the alignment pattern is printed, the LP print service will attempt to truncate the *content* to a single form by, if possible, passing to each filter the page subset of 1-*integer*.

`Line pitch` and `Character pitch`
: Before printing the content of a print request needing this form, the interface programs provided with the LP print service will initialize the physical printer to handle these pitches, using the printer type as a key into the `terminfo` database. Also, the pitches will be passed, if possible, to each filter used in a request needing this form. *Scaled-decimal-number*$_3$ is in lines per centimeter if a `c` is appended, and lines per inch otherwise; similarly, *scaled – decimal – number*$_4$ is in characters per centimeter if a `c` is appended, and characters per inch otherwise. The character pitch can also be given as `elite` (12 characters per inch), `pica` (10 characters per inch), or `compressed` (as many characters per inch as possible).

`Character set choice`
: When the LP print service alerts an administrator to mount this form, it will also mention that the print wheel *print-wheel* should be used on those printers that take print wheels. If printing with this form is to be done on a printer that has selectable or loadable character sets instead of print wheels,

the interface programs provided with the LP print service will automatically select or load the correct character set. If `mandatory` is appended, a user is not allowed to select a different character set for use with the form; otherwise, the character set or print wheel named is a suggestion and a default only.

`Ribbon color`
When the LP print service alerts an administrator to mount this form, it will also mention that the color of the ribbon should be *ribbon-color*.

`Comment`
The LP print service will display the *comment* unaltered when a user asks about this form [see `lpstat`(1)].

`Alignment pattern`
When mounting this form an administrator can ask for the *content* to be printed repeatedly, as an aid in correctly positioning the preprinted form. The optional *content-type* defines the type of printer for which *content* had been generated. If *content-type* is not given, `simple` is assumed. Note that the *content* is stored as given, and will be readable only by the user `lp`.

When an existing form is changed with this command, items missing in the new information are left as they were. When a new form is added with this command, missing items will get the following defaults:

Page Length: `66`
Page Width: `80`
Number of Pages: `1`
Line Pitch: `6`
Character Pitch: `10`
Character Set Choice: `any`
Ribbon Color: `any`

Deleting a Form

The `-x` option is used to delete the form *form-name* from the LP print service.

Listing Form Attributes

The `-l` option is used to list the attributes of the existing form *form-name*. The attributes listed are those described under **Adding and Changing a Form,** above. Because of the potentially sensitive nature of the alignment pattern, only the administrator can examine the form with this command. Other people may use the `lpstat` command to examine the non-sensitive part of the form description.

Allowing and Denying Access to a Form

The `-u` option, followed by the argument `allow:`*login-ID-list* or `-u deny:`*login-ID-list* lets you determine which users will be allowed to specify a particular form with a print request. This option can be used with the `-F` or - option, each of which is described above under **Adding or Changing a Form.**

The *login-ID-list* argument may include any or all of the following constructs:

login-ID A user on any system

system_name!*login-ID*	A user on system *system_name*
system_name!all	All users on system *system_name*
all!*login-ID*	A user on all systems
all	All users on all systems

The LP print service keeps two lists of users for each form: an "allow-list" of people allowed to use the form, and a "deny-list" of people that may not use the form. With the -u allow option, the users listed are added to the allow-list and removed from the deny-list. With the -u deny option, the users listed are added to the deny-list and removed from the allow-list. (Both forms of the -u option can be run together with the -F or the - option.)

If the allow-list is not empty, only the users in the list are allowed access to the form, regardless of the contents of the deny-list. If the allow-list is empty but the deny-list is not, the users in the deny-list may not use the form, (but all others may use it). All users can be denied access to a form by specifying -f deny:all. All users can be allowed access to a form by specifying -f allow:all. (This is the default.)

Setting an Alert to Mount a Form

The -f *form-name* option is used with the -A *alert-type* option to define an alert to mount the form when there are queued jobs which need it. If this option is not used to arrange alerting for a form, no alert will be sent for that form.

The method by which the alert is sent depends on the value of the *alert-type* argument specified with the -A option. The *alert-types* are:

mail
: Send the alert message via the mail command to the administrator.

write
: Write the message, via the write command, to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is arbitrarily chosen.

quiet
: Do not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the form *form-name* has been mounted and subsequently unmounted, messages will again be sent when the number of print requests reaches the threshold specified by the -Q option.

none
: Do not send messages until the -A option is given again with a different *alert-type* (other than quiet).

shell-command
: Run the *shell-command* each time the alert needs to be sent. The shell command should expect the message in standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the mail and write values for this option are equivalent to the values mail *login-ID* and write *login-ID* respectively, where *login-ID* is the current name for the administrator. This will be the login name of the person submitting this command unless he or she has used the su command to change to another login-ID. If the su command has been used to change the user ID, then the *user-name* for

the new ID is used.

`list` Display the type of the alert for the form on standard output. No change is made to the alert.

The message sent appears as follows:

```
The form form-name needs to be mounted
on the printer(s):
printer (integer1 requests).
integer2 print requests await this form.
Use the ribbon-color ribbon.
Use the print-wheel print wheel, if appropriate.
```

The printers listed are those that the administrator had earlier specified were candidates for this form. The number *integer*$_1$ listed next to each printer is the number of requests eligible for the printer. The number *integer*$_2$ shown after the list of printers is the total number of requests awaiting the form. It will be less than the sum of the other numbers if some requests can be handled by more than one printer. The *ribbon-color* and *print-wheel* are those specified in the form description. The last line in the message is always sent, even if none of the printers listed use print wheels, because the administrator may choose to mount the form on a printer that does use a print wheel.

Where any color ribbon or any print wheel can be used, the statements above will read:

```
Use any ribbon.
Use any print-wheel.
```

If *form-name* is `any`, the alerting defined in this command applies to any form for which an alert has not yet been defined. If *form-name* is `all`, the alerting defined in this command applies to all forms.

If the `-W` option is not given, the default procedure is that only one message will be sent per need to mount the form. Not specifying the `-W` option is equivalent to specifying `-W once` or `-W 0`. If *minutes* is a number greater than 0, an alert will be sent at intervals specified by *minutes*.

If the `-Q` option is also given, the alert will be sent when a certain number (specified by the argument *requests*) of print requests that need the form are waiting. If the `-Q` option is not given, or the value of *requests* is `1` or `any` (which are both the default), a message is sent as soon as anyone submits a print request for the form when it is not mounted.

Listing the Current Alert

The `-f` option, followed by the `-A` option and the argument `list` is used to list the type of alert that has been defined for the specified form *form-name*. No change is made to the alert. If *form-name* is recognized by the LP print service, one of the following lines is sent to the standard output, depending on the type of alert for the form.

- When *requests* requests are queued:
  alert with *shell-command* every *minutes* minutes
- When *requests* requests are queued:
  write to *user-name* every *minutes* minutes
- When *requests* requests are queued:
  mail to *user-name* every *minutes* minutes
- No alert

The phrase every *minutes* minutes is replaced with once if *minutes* (-W *minutes)* is 0.

Terminating an Active Alert

The -A quiet option is used to stop messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the form has been mounted and then unmounted, messages will again be sent when the number of print requests reaches the threshold *requests*.

Removing an Alert Definition

No messages will be sent after the -A none option is used until the -A option is given again with a different *alert-type*. This can be used to permanently stop further messages from being sent as any existing alert definition for the form will be removed.

SEE ALSO

lpadmin(1M), terminfo(4).

NAME

lpq - display the queue of printer jobs

SYNOPSIS

/usr/ucb/lpq [-P*printer*] [-l] [+[*interval*]] [*job#* ...] [*username* ...]

DESCRIPTION

lpq displays the contents of a printer queue. It reports the status of jobs specified by *job#*, or all jobs owned by the user specified by *username*. lpq reports on all jobs in the default printer queue when invoked with no arguments.

For each print job in the queue, lpq reports the user's name, current position, the names of input files comprising the job, the job number (by which it is referred to when using lprm(1)) and the total size in bytes. Normally, only as much information as will fit on one line is displayed. Jobs are normally queued on a first-in-first-out basis. Filenames comprising a job may be unavailable, such as when lpr is used at the end of a pipeline; in such cases the filename field indicates the standard input.

If lpq warns that there is no daemon present (that is, due to some malfunction), the lpc(1M) command can be used to restart a printer daemon.

OPTIONS

-P *printer*	Display information about the queue for the specified *printer*. In the absence of the -P option, the queue to the printer specified by the PRINTER variable in the environment is used. If the PRINTER variable is not set, the queue for the default printer is used.
-l	Display queue information in long format; includes the name of the host from which the job originated.
+[*interval*]	Display the spool queue periodically until it empties. This option clears the terminal screen before reporting on the queue. If an *interval* is supplied, lpq sleeps that number of seconds in between reports.

FILES

/var/spool/lp	spooling directory.
/var/spool/lp/tmp/*system_name*/*-0	reqeust files specifying jobs

DIAGNOSTICS

printer is printing
: The lpq program queries the spooler LPSCHED about the status of the printer. If the printer is disabled, the superuser can restart the spooler using lpc(1M).

printer waiting for auto-retry (offline ?)
: The daemon could not open the printer device. The printer may be turned off-line. This message can also occur if a printer is out of paper, the paper is jammed, and so on. Another possible cause is that a process, such as an output filter, has exclusive use of the device. The only recourse in this case is to kill the offending process and restart the printer with lpc.

waiting for *host* to come up
: A daemon is trying to connect to the remote machine named *host*, in order to send the files in the local queue. If the remote machine is up, lpd on the remote machine is probably dead or hung and should be restarted using

lpc.

`sending to` *host*
: The files are being transferred to the remote *host*, or else the local daemon has hung while trying to transfer the files.

`printer disabled reason:`
: The printer has been marked as being unavailable with `lpc`.

`lpq: The LP print service isn't running or can't be reached.`
: The `lpsched` process overseeing the spooling queue does not exist. This normally occurs only when the daemon has unexpectedly died. You can restart the printer daemon with `lpc`.

`lpr:` *printer* `: unknown printer`
: The `printer` was not found in the System V LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use '`lpstat -p`' to find the reason.

`lpr: error on opening queue to spooler`
: The connection to `lpsched` on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon `/usr/lib/lpsched` is running.

`lpr: Can't send message to LP print service`

`lpr: Can't receive message from LP print service`
: These indicate that the LP print service has been stopped. Get help from the system administrator.

`lpr: Received unexpected message from LP print service`
: It is likely there is an error in this software. Get help from system administrator.

SEE ALSO

`lp`(1), `lpr`(1), `lprm`(1)

NOTES

Output formatting is sensitive to the line length of the terminal; this can result in widely-spaced columns.

NAME

lpr - send a job to the printer

SYNOPSIS

/usr/ucb/lpr [-P *printer*] [-# *copies*] [-C *class*] [-J *job*] [-T *title*]
[-i [*indent*]] [-w *cols*] [-B] [-r] [-m] [-h] [-s]
[*-filter_option*] [*filename* . . .]

DESCRIPTION

lpr forwards printer jobs to a spooling area for subsequent printing as facilities become available. Each printer job consists of copies of, or, with -s , complete pathnames of each *filename* you specify. The spool area is managed by the line printer spooler, lpsched. lpr reads from the standard input if no files are specified.

OPTIONS

-P *printer* — Send output to the named *printer*. Otherwise send output to the printer named in the PRINTER environment variable, or to the default printer, lp.

-# *copies* — Produce the number of *copies* indicated for each named file. For example:

```
lpr -#3 index.c lookup.c
```

produces three copies of index.c, followed by three copies of lookup.c. On the other hand,

```
cat index.c lookup.c | lpr -#3
```

generates three copies of the concatenation of the files.

-C *class* — Print *class* as the job classification on the burst page. For example,

```
lpr -C Operations new.index.c
```

replaces the system name (the name returned by *hostname*) with Operations on the burst page, and prints the file new.index.c.

-J *job* — Print *job* as the job name on the burst page. Normally, lpr uses the first file's name.

-T *title* — Use *title* instead of the file name for the title used by pr(1).

-i[*indent*] — Indent output *indent* SPACE characters. Eight SPACE characters is the default.

-w *cols* — Use *cols* as the page width for pr.

-r — Remove the file upon completion of spooling, or upon completion of printing with the -s option. This is not supported in the bsd compatibility package. However if the job is submitted to a remote SunOS system, these options will be sent to the remote system for processing.

-m — Send mail upon completion.

-h — Suppress printing the burst page.

`-s` Use the full pathnames (not symbolic links) of the files to be printed rather than trying to copy them. This means the data files should not be modified or removed until they have been printed. `-s` only prevents copies of local files from being made. Jobs from remote hosts are copied anyway. `-s` only works with named data files; if the `lpr` command is at the end of a pipeline, the data is copied to the spool.

filter_option The following single letter options notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

- `-p` Use `pr` to format the files (`lpr -p` is very much like `pr | lpr`).
- `-l` Print control characters and suppress page breaks.
- `-t` The files contain `troff`(1) (cat phototypesetter) binary data.
- `-n` The files contain data from `ditroff` (device independent troff).
- `-d` The files contain data from `tex` (DVI format from Stanford).
- `-g` The files contain standard plot data as produced by the `plot`(3X) routines (see also `plot`(1G) for the filters used by the printer spooler).
- `-v` The files contain a raster image. The printer must support an appropriate imaging model such as PostScript® in order to print the image.
- `-c` The files contain data produced by *cifplot*.
- `-f` Interpret the first character of each line as a standard FORTRAN carriage control character.

If no *filter_option* is given (and the printer can interpret PostScript), the string '`%!`' as the first two characters of a file indicates that it contains PostScript commands.

These filter options offer a standard user interface, and all options may not be available for, nor applicable to, all printers.

FILES

`/etc/passwd`	personal identification
`/usr/lib/lp/lpsched`	System V line printer spooler
`/var/spool/lp/tmp/*`	directories used for spooling
`/var/spool/lp/tmp/`*system*`/*-0`	spooler control files
`/var/spool/lp/tmp/`*system*`/*-`*N*	(*N* is an integer and > 0) data files specified in '`*-0`' files

DIAGNOSTICS

`lpr: `*printer*`: unknown printer`
: The `printer` was not found in the LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use '`lptstat -p`' to find the reason.

`lpr: error on opening queue to spooler`
: The connection to `lpsched` on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon `/usr/lib/lpsched` is running.

`lpr: `*printer*`: printer queue is disabled`
: This means the queue was turned off with

  `/usr/etc/lpc disable `*printer*

  to prevent `lpr` from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a privileged user with `lpc`.

`lpr: Can't send message to the LP print service`

`lpr: Can't receive message from the LP print service`
: These indicate that the LP print service has been stopped. Get help from the system administrator.

`lpr: Received unexpected message from LP print service`
: It is likely there is an error in this software. Get help from system administrator.

`lpr: There is no filter to convert the file content`
: Use the '`lpstat -p -l`' command to find a printer that can handle the file type directly, or consult with your system administrator.

`lpr: cannot access the file`
: Make sure file names are valid.

SEE ALSO

`lp`(1), `lpq`(1), `lprm`(1), `lpsched`(1), `pr`(1), `plot`(1G), `troff`(1), `plot`(3X).

NOTES

`lp` is the preferred interface.

Command-line options cannot be combined into a single argument as with some other commands. The command:

```
lpr -fs
```

is not equivalent to

```
lpr -f -s
```

Placing the `-s` flag first, or writing each option as a separate argument, makes a link as expected.

`lpr -p` is not precisely equivalent to `pr | lpr`. `lpr -p` puts the current date at the top of each page, rather than the date last modified.

Fonts for `troff`(1) and T_EX® reside on the printer host. It is currently not possible to use local font libraries.

`lpr` objects to printing binary files.

The `-s` option, intended to use symbolic links in SunOS, does not use symbolic links in the compatibility package. Instead, the complete path names are used. Also, the copying is avoided only for print jobs that are run from the printer host itself. Jobs added to the queue from a remote host are always copied into the spool area. That is, if the printer does not reside on the host that `lpr` is run from, the spooling system makes a copy the file to print, and places it in the spool area of the printer host, regardless of `-s`.

NAME

lprm - remove jobs from the printer queue

SYNOPSIS

/usr/ucb/lprm [-P*printer*] [-] [*job* # . . .] [*username* . . .]

DESCRIPTION

lprm removes a job or jobs from a printer's spooling queue. Since the spool directory is protected from users, using lprm is normally the only method by which a user can remove a job.

Without any arguments, lprm deletes the job that is currently active, provided that the user who invoked lprm owns that job.

When the privileged user specifies a *username*, lprm removes all jobs belonging to that user.

You can remove a specific job by supplying its job number as an argument, which you can obtain using lpq(1). For example:

```
lpq  -Phost
host is ready and printing
Rank Owner   Job     Files   Total Size
active       wendy   385     standard input  35501 bytes
lprm -Phost 385
```

lprm reports the names of any files it removes, and is silent if there are no applicable jobs to remove.

lprm Sends the request to cancel a job to the print spooler, LPSCHED.

OPTIONS

-P*printer* Specify the queue associated with a specific printer. Otherwise the value of the PRINTER variable in the environment is used. If this variable is unset, the queue for the default printer is used.

- Remove all jobs owned by you. If invoked by the privileged user, all jobs in the spool are removed. Job ownership is determined by the user's login name and host name on the machine where the lpr command was executed.

FILES

/var/spool/lp/* spooling directories

SEE ALSO

cancel(1), lpq(1), lpr(1), lpsched(1M), lp(1).

DIAGNOSTICS

lprm: *printer* : unknown printer
The printer was not found in the System V LP database. Usually this is a typing mistake; however, it may indicate that the printer does not exist on the system. Use 'lptstat -p' to find the reason.

lprm: error on opening queue to spooler
The connection to lpsched on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check if the printer spooler daemon /usr/lib/lpsched is running.

`lprm: Can't send message to the LP print service`

`lprm: Can't receive message from the LP print service`
These indicate that the LP print service has been stopped. Get help from the system administrator.

`lprm: Received unexpected message from the LP print service`
It is likely there is an error in this software. Get help from system administrator.

`lprm: Can't cancel request`
You are not allowed to remove another's request.

NOTES

An active job may be incorrectly identified for removal by an `lprm` command issued with no arguments. During the interval between an `lpq`(1) command and the execution of `lprm`, the next job in queue may have become active; that job may be removed unintentionally if it is owned by you. To avoid this, supply `lprm` with the job number to remove when a critical job that you own is next in line.

Only the privileged user can remove print jobs submitted from another host.

`lp` is the preferred interface.

NAME

lprof - display line-by-line execution count profile data

SYNOPSIS

lprof [-p] [-s] [-x] [-I *incdir*] [-r *srcfile*] [-c *cntfile*] [-o *prog*] [-V]

lprof -m *file1*.cnt *file2*.cnt *filen*.cnt [-T] -d *destfile*.cnt

DESCRIPTION

lprof reports the execution characteristics of a program on a (source) line by line basis. This is useful as a means to determine which and how often portions of the code were executed.

lprof interprets a profile file (*prog*.cnt by default) produced by the profiled program *prog* (a.out by default). *prog* creates a profile file if it has been loaded with the -ql option of cc. The profile information is computed for functions in a source file if the -ql option was used when the source file was compiled.

A shared object may also be profiled by specifying -ql when the shared object is created. When a dynamically linked executable is run, one profile file is produced for each profiled shared object linked to the executable. This feature is useful in building a single report covering multiple and disparate executions of a common library. For example, if programs prog1 and prog2 both use library libx.a, running these profiled programs will produce two profile files, prog1.cnt and prog2.cnt, which cannot be combined. However, if libx is built as a profiled shared object, libx.so, and prog1 and prog2 are built as profiled dynamically linked executables, then running these programs with the merge option will produce three profile files; one of them, libx.so.cnt, will contain the libx profile information from both runs.

By default, lprof prints a listing of source files (the names of which are stored in the symbol table of the executable file), with each line preceded by its line number (in the source file) and the number of times the line was executed.

The following options may appear singly or be combined in any order:

-p Print listing, each line preceded by the line number and the number of times it was executed (default). This option can be used together with the -s option to print both the source listing and summary information.

-s Print summary information of percentage of lines of code executed per function.

-x Instead of printing the execution count numbers for each line, print each line preceded by its line number and a [U] if the line was not executed. If the line was executed, print only the line number.

-I *incdir* Look for source or header files in the directory *incdir* in addition to the current directory and the standard place for #include files (usually /usr/include). The user can specify more than one directory by using multiple -I options.

-r *srcfile* Instead of printing all source files, print only those files named in -r options (to be used with the -p option only). The user can specify multiple files with a single -r option.

`-c` *cntfile* Use the file *cntfile* instead of *prog*`.cnt` as the input profile file.

`-o` *prog* Use the name of the program *prog* instead of the name used when creating the profile file. Because the program name stored in the profile file contains the relative path, this option is necessary if the executable file or profile file has been moved.

`-V` Print, on standard error, the version number of `lprof`.

Merging Data Files

`lprof` can also be used to merge profile files. The `-m` option must be accompanied by the `-d` option:

`-m` *file1*`.cnt` *file2*`.cnt` *filen*`.cnt -d` *destfile*`.cnt`
Merge the data files *file1*`.cnt` through *filen*`.cnt` by summing the execution counts per line, so that data from several runs can be accumulated. The result is written to *destfile*`.cnt`. The data files must contain profiling data for the same *prog* (see the `-T` option below).

`-T` Time stamp override. Normally, the time stamps of the executable files being profiled are checked, and data files will not be merged if the time stamps do not match. If `-T` is specified, this check is skipped.

CONTROLLING THE RUN-TIME PROFILING ENVIRONMENT

The environment variable `PROFOPTS` provides run-time control over profiling. When a profiled program (or shared object) is about to terminate, it examines the value of `PROFOPTS` to determine how the profiling data are to be handled. A terminating shared object will honor every `PROFOPTS` option except `file=`*filename*.

The environment variable `PROFOPTS` is a comma-separated list of options interpreted by the program being profiled. If `PROFOPTS` is not defined in the environment, then the default action is taken: The profiling data are saved in a file (with the default name, *prog*`.cnt`) in the current directory. If `PROFOPTS` is set to the null string, no profiling data are saved. The following are the available options:

`msg=[y | n]` If `msg=y` is specified, a message stating that profile data are being saved is printed to `stderr`. If `msg=n` is specified, only the profiling error messages are printed. The default is `msg=y`.

`merge=[y | n]` If `merge=y` is specified, the data files will be merged after successive runs. If `merge=n` is specified, the data files are not merged after successive runs, and the data file is overwritten after each execution. The merge will fail if the program has been recompiled, and the data file will be left in `TMPDIR`. The default is `merge=n`.

`pid=[y | n]` If `pid=y` is specified, the name of the data file will include the process ID of the profiled program. Inclusion of the process ID allows for the creation of different data files for programs calling `fork`. If `pid=n` is specified, the default name is used. The default is `pid=n`. For `lprof` to generate its profiling report, the `-c` option must be specified with `lprof` otherwise the default will fail.

`dir=`*dirname* The data file is placed in the directory *dirname* if this option is specified. Otherwise, the data file is created in the directory that is current at the end of execution.

`file=`*filename* *filename* is used as the name of the data file in *dir* created by the profiled program if this option is specified. Otherwise, the default name is used. For `lprof` to generate its profiling report, the `-c` option must be specified with `lprof` if the file option has been used at execution time; otherwise the default will fail.

FILES

prog`.cnt`	profile data
TMPDIR	usually `/var/tmp` but can be redefined by setting the environment variable `TMPDIR` [see `tempnam` in `tmpnam`(3S)].

SEE ALSO

`cc`(1), `prof`(1), `fork`(2), `tmpnam`(3S).

NOTES

For the `-m` option, if *destfile*`.cnt` exists, its previous contents are destroyed.

Optimized code cannot be profiled; if both optimization and line profiling are requested, profiling has precedence.

Including header files that contain code (such as `stat.h` or `utsname.h`) will cause erroneous data.

Different parts of one line of a source file may be executed different numbers of times (for example, the `for` loop below); the count corresponds to the first part of the line.

For example, in the following `for` loop

```
                main()
1    [2]        {
                    int j;

1    [5]            for (j = 0; j < 5; j++)
5    [6]            sub(j);

1    [8]        }

                sub(a)
                int a;
5    [12]           {
5    [13]           printf("a is %d\n", a);
5    [14]           }
```

line 5 consists of three parts. The line count listed, however, is for the initialization part, that is, `j = 0`.

NAME

lpsched, lpshut, lpmove - start/stop the LP print service and move requests

SYNOPSIS

/usr/lib/lp/lpsched
lpshut
lpmove *requests dest*
lpmove *dest1 dest2*

DESCRIPTION

lpsched starts the LP print service; this can be done only by root or lp.

lpshut shuts down the print service. All printers that are printing at the time lpshut is invoked will stop printing. When lpsched is started again, requests that were printing at the time a printer was shut down will be reprinted from the beginning.

lpmove moves requests that were queued by lp between LP destinations. The first form of the lpmove command shown above (under **SYNOPSIS**) moves the named *requests* to the LP destination *dest*. *Requests* are request-IDs as returned by lp. The second form of the lpmove command will attempt to move all requests for destination *dest1* to destination *dest2*; lp will then reject any new requests for *dest1*.

Note that when moving requests, lpmove never checks the acceptance status [see accept(1M)] of the new destination. Also, the request-IDs of the moved request are not changed, so that users can still find their requests. The lpmove command will not move requests that have options (content type, form required, and so on) that cannot be handled by the new destination.

If a request was originally queued for a class or the special destination any, and the first form of lpmove was used, the destination of the request will be changed to *new-destination*. A request thus affected will be printable only on *new-destination* and not on other members of the class or other acceptable printers if the original destination was any.

FILES

/var/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1).

NAME

lpstat - print information about the status of the LP print service

SYNOPSIS

lpstat [*options*]

DESCRIPTION

The lpstat command prints information about the current status of the LP print service.

If no options are given, then lpstat prints the status of all output (or print) requests made by lp [see lp(1)]. Any arguments that are not *options* are assumed to be *request-IDs* as returned by lp. The lpstat command prints the status of such requests. The *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 separated from one another by spaces. A list that includes spaces or shell special characters must be enclosed in double quotes. For example:

```
-u "user1, user2, user3"
```

Specifying all after any keyletters that take list as an argument causes all information relevant to the keyletter to be printed. For example, the command

```
lpstat -o all
```

prints the status of all output requests.

The omission of a *list* following such key letters causes all information relevant to the key letter to be prined. For example, the command

```
lpstat -o
```

prints the status of all output requests.

-a [*list*] Reports whether print destinations are accepting requests. *list* is a list of intermixed printer names and class names.

-c [*list*] Reports name of all classes and their members. *list* is a list of class names.

-d Reports the system default destination for output requests.

-f [*list*] [-l]
Prints a verification that the forms in *list* are recognized by the LP print service. *list* is a list of forms; the default is all. The -l option will list the form descriptions.

-o [*list*] Reports the status of output requests. *list* is a list of intermixed printer names, class names, and *request-IDs*.

-p [*list*] [-D] [-l]
Reports the status of printers. *list* is a list of printer names. If the -D option is given, a brief description is printed for each printer in *list*. If the -l option is given, and the printer is on the local machine, a full description of each printer's configuration is given, including the form mounted, the acceptable content and printer types, a printer description, the interface used, and so on. If the -l option is given and the printer is remote, the only information given is the remote machine and printer names, and the shell-commands used for file transfer and remote execution.

`-r` Reports whether the LP request scheduler is on or off.

`-R` Reports a number showing the position of the job in the print queue.

`-s` Displays a status summary, including the status of the LP scheduler, the system default destination, a list of class names and their members, a list of printers and their associated devices, a list of the machines sharing print services, a list of all forms currently mounted, and a list of all recognized character sets and print wheels.

`-S` [*list*] [`-l`]
Prints a verification that the character sets or the print wheels specified in *list* are recognized by the LP print service. Items in *list* can be character sets or print wheels; the default for the list is `all`. If the `-l` option is given, each line is appended by a list of printers that can handle the print wheel or character set. The list also shows whether the print wheel or character set is mounted or specifies the built-in character set into which it maps.

`-t` Displays all status information: all the information obtained with the `-s` option, plus the acceptance and idle/busy status of all printers.

`-u` [*login-ID-list*]
Displays the status of output requests for users. The *login-ID-list* argument may include any or all of the following constructs:

user_name	a user on the local system
system_name`!`*user_name*	a user on *system_name*
system_name`!all`	all users on *system_name*
`all!`*user_name*	a user not on the local system
`all!all`	all users not on the local system
`all`	all users on the local system

`-v` [*list*] Reports the names of printers and the pathnames of the devices associated with them. *list* is a list of printer names.

FILES

`/var/spool/lp/*`
`/etc/lp/*`

SEE ALSO

`enable`(1), `lp`(1).

NAME

lpsystem - register remote systems with the print service

SYNOPSIS

lpsystem [-t *type*] [-T *timeout*] [-R *retry*] [-y "*comment*"] *system-name*.br
[*system-name* . . .]
lpsystem -l [*system-name* . . .]
lpsystem -r *system-name* [*system-name* . . .]
lpsystem -A

DESCRIPTION

The lpsystem command is used to define parameters for the LP print service, with respect to communication (via a high-speed network such as STARLAN or TCP/IP) with remote systems. Only a privileged user (that is, the owner of the login root) may execute the lpsystem command.

Specifically, the lpsystem command is used to define remote systems with which the local LP print service can exchange print requests. These remote systems are described to the local LP print service in terms of several parameters that control communication: type, retry and timeout. These parameters are defined in /etc/lp/Systems. You can edit this file with a text editor (such as vi) but editing is not recommended.

The *type* parameter defines the remote system as one of two types: s5 (System V Release 4) or bsd (SunOS). The default type is s5.

The *timeout* parameter specifies the length of time (in minutes) that the print service should allow a network connection to be idle. If the connection to the remote system is idle (that is, there is no network traffic) for *N* minutes, then drop the connection. (When there is more work the connection will be restablished.) Legal values are n, 0, and *N*, where *N* is an integer greater than 0. The value n means "never time out"; 0 means "as soon as the connection is idle, drop it." The default is n.

The *retry* parameter specifies the length of time to wait before trying to re-establish a connection to the remote system, when the connection was dropped abnormally (that is, a network error). Legal values are n, 0, and *N*, where *N* is an integer greater than 0 and it means "wait *N* minutes before trying to reconnect. (The default is 10 minutes.) The value n means "do not retry dropped connections until there is more work"; 0 means "try to reconnect immediately."

The *comment* argument allows you to associate a free form comment with the system entry. This is visible when lpsystem -l is used.

System-name is the name of the remote system from which you want to be able to receive jobs, and to which you want to be able to send jobs.

The command lpsystem -l [*system-name*] will print out a description of the parameters associated with *system-name* (if a system has been specified), or with all the systems in its database (if *system-name* has not been specified).

The command lpsystem -r *system-name* will remove the entry associated with *system-name*. The print service will no longer accept jobs from that system or send jobs to it, even if the remote printer is still defined on the local system.

The command `lpsystem -A` will print out the TCP/IP address of the local machine in a format to be used when configuring the local port monitor to accept requests from a SunOS system.

NOTES:

With respect to `/etc/lp/Systems`, this information is relatively minimal with repect to controlling network communications. Network addresses and services are handled by the `Netconfig` and `Netdir` facilities. Port monitors handle listening for remote service requests and routing the connection to the print service.

If the `Netconfig` and `Netdir` facilities are not set up properly, out-bound remote print service probably will not work. Similarly, if the local port monitors are not set up to route remote print requests to the print service, then service for remote systems will not be provided.

With respect to the semantics of the *timeout* and *retry* values, the print service uses one process for each remote system with which it communicates, and it communicates with a remote system only when there is work to be done on that system or work being sent from that system.

The system initiating the connection is the "master" process and the system accepting the connection is the "slave" process. This designation serves only to determine which process dies (the slave) when a connection is dropped. This helps prevent there from being more than one process communicating with a remote system. Furthermore, all connections are bi-directional, regardless of the master/slave designation. You cannot control a system's master/slave designation. Now, keeping all this information in mind, if a master process times out, then both the slave and master will exit. If a slave times out, then it is possible that the master may still live and retry the connection after the retry interval. Therefore, one system's resource management strategy can effect another system's strategy.

With respect to `lpsystem -A`: a SunOS system (described with `-t bsd`) can be connected to your system only via TCP/IP, and print requests from a SunOS system can come in to your machine only via a special port (515). The address given to you from `lpsystem` will be the address of your system and port 515. This address is used by your TCP/IP port monitor (see `sacadm`(1M) and `nlsadmin`(1M)) to "listen" on that address and port, and to route connections to the print service. The important point here is that this is where you get the address refered to in that procedure.

The command `lpsystem -A` will not work if your system name and IP address are not listed in `/etc/inet/hosts` and the printer service is not listed in `/etc/inet/services`.

FILES

`/var/spool/lp/* /etc/lp/*`

SEE ALSO

`netconfig`(4)

NAME

`lptest` - generate lineprinter ripple pattern

SYNOPSIS

`/usr/ucb/lptest` [*length* [*count*]]

DESCRIPTION

`lptest` writes the traditional "ripple test" pattern on standard output. In 96 lines, this pattern will print all 96 printable ASCII characters in each position. While originally created to test printers, it is quite useful for testing terminals, driving terminal ports for debugging purposes, or any other task where a quick supply of random data is needed.

The *length* argument specifies the output line length if the the default length of 79 is inappropriate.

The *count* argument specifies the number of output lines to be generated if the default count of 200 is inappropriate.

NOTES

If *count* is to be specified, *length* must be also be specified.

This command is obsolescent.

NAME

lpusers - set printing queue priorities

SYNOPSIS

```
lpusers -d priority-level
lpusers -q priority-level -u login-ID-list
lpusers -u login-ID-list
lpusers -q priority-level
lpusers -l
```

DESCRIPTION

The lpusers command is used to set limits to the queue priority level that can be assigned to jobs submitted by users of the LP print service.

The first form of the command (with -d) sets the system-wide priority default to *priority-level*, where *priority-level* is a value of 0 to 39, with 0 being the highest priority. If a user does not specify a priority level with a print request [see lp(1)], the default priority is used. Initially, the default priority level is 20.

The second form of the command (with -q and -u) sets the default highest *priority-level* (0-39) that the users in *login-ID-list* can request when submitting a print request. The *login-ID-list* argument may include any or all of the following constructs:

login-ID	A user on any system
system_name!*login-ID*	A user on the system *system_name*
system_name!all	All users on system *system_name*
all!*login-ID*	A user on all systems
all	All users on all systems

Users that have been given a limit cannot submit a print request with a higher priority level than the one assigned, nor can they change a request already submitted to have a higher priority. Any print requests submitted with priority levels higher than allowed will be given the highest priority allowed.

The third form of the command (with -u) removes any explicit priority level for the specified users.

The fourth form of the command (with -q) sets the default highest priority level for all users not explicitly covered by the use of the second form of this command.

The last form of the command (with -l) lists the default priority level and the priority limits assigned to users.

SEE ALSO

lp(1).

NAME

ls - list contents of directory

SYNOPSIS

ls [-RadLCxmlnogrtucpFbqisf1] [*file* . . .]

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 alphabetically by default. 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.

There are three major listing formats. The default format for output directed to a terminal is multi-column with entries sorted down the columns. The -1 option allows single column output and -m enables stream output format. 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(4) database is used to determine the number of columns, based on the environment variable TERM. If this information cannot be obtained, 80 columns are assumed.

The ls command has the following options:

-R Recursively list subdirectories encountered.

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-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.

-L When listing status, if an argument is a symbolic link, list the status of the file or directory referenced by the link rather than that of the link itself.

-C Multi-column output with entries sorted down the columns. This is the default output format.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format; files are listed across the page, separated by commas.

-l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field instead contains the major and minor device numbers rather than a size. If the file is a symbolic link, the filename is printed followed by "->" and the pathname of the referenced file.

-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 the group is not printed.

-g The same as -l, except that the owner is not printed.

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

`-t` Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See `-n` and `-c`.)

`-u` Use time of last access instead of last modification for sorting (with the `-t` option) or printing (with the `-l` option).

`-c` Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (`-t`) or printing (`-l`).

`-p` Put a slash (`/`) after each filename if the file is a directory.

`-F` Put a slash (`/`) after each filename if the file is a directory, an asterisk (`*`) if the file is an executable, and an ampersand (`@`) if the file is a symbolic link.

`-b` Force printing of non-printable characters to be in the octal *\ddd* notation.

`-q` Force printing of non-printable characters in filenames as the character question mark (`?`).

`-i` For each file, print the i-node number in the first column of the report.

`-s` Give size in blocks, including indirect blocks, for each entry.

`-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.

`-1` Print one entry per line of output.

The mode printed under the `-l` option consists of ten characters. The first character may be one of the following:

`d` the entry is a directory
`l` the entry is a symbolic link
`b` the entry is a block special file
`c` the entry is a character special file
`m` the entry is XENIX shared data (memory) file
`p` the entry is a fifo (a.k.a., named pipe) special file
`s` the entry is a XENIX semaphore
`-` 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.

`ls -l` (the `l`ong list) prints its output as follows:

```
-rwxrwxrwx  1 smith  dev    10876  May 16 9:42 part2
```

Reading from right to left, you see that the current directory holds one file, named `part2`. The last time that file's contents were modified was 9:42 A.M. on May 16. The file contains 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group `dev` (perhaps indicating "development"), and his or her login name is `smith`. The number, in this case `1`, indicates the number of links to file `part2`; see `cp`(1). Finally, the dash and letters tell you that user, group, and others have permissions to read, write, and execute `part2`.

The execute (x) symbol here occupies the third position of the three-character sequence. A - in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

- r the file is readable
- w the file is writable
- x the file is executable
- - the indicated permission is *not* granted
- l mandatory locking occurs during access (the set-group-ID bit is on and the group execution bit is off)
- s the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
- S undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
- t the 1000 (octal) bit, or sticky bit, is on [see chmod(1)], and execution is on
- T the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than x or -. s also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user you login as.

In the case of the sequence of group permissions, l may occupy the third position. l refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by t or T. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

An example of a file's permissions is:

```
-rwxr--r--
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions is:

```
-rwsr-xr-x
```

This describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

Another example of a file's permissions is:

```
-rw-rwl---
```

This describes a file that is readable and writable only by the user and the group and can be locked during access.

An example of a command line:

```
ls -a
```

This command prints the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

Another example of a command line:

```
ls -aisn
```

This command provides information on all files, including those that begin with a dot (a), the i-number—the memory address of the i-node associated with the file—printed in the left-hand column (i); the size (in blocks) of the files, printed in the column to the right of the i-numbers (s); finally, the report is displayed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	user IDs for ls -l and ls -o
/etc/group	group IDs for ls -l and ls -g
/usr/share/lib/terminfo/?/*	terminal information database

INTERNATIONAL FUNCTIONS

ls can process directory and filenames containing characters from supplementary code sets. Multi-column output can be displayed correctly using the -C and -x options.

With the -b and -q options, ls considers all characters from supplementary code sets to be printable.

SEE ALSO

chmod(1), find(1).

NOTES

In a Remote File Sharing (RFS) environment, you may not have the permissions that the output of the ls -l command leads you to believe.

Unprintable characters in filenames may confuse the columnar output options.

The total block count will be incorrect if there are hard links among the files.

NAME

ls - list the contents of a directory

SYNOPSIS

/usr/ucb/ls [-aAcCdfFgilLqrRstu1] *filename* . . .

DESCRIPTION

For each *filename* which is a directory, ls lists the contents of the directory; for each *filename* which is a file, ls repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

Permissions Field

The mode printed under the -l option contains 10 characters interpreted as follows. If the first character is:

- d entry is a directory;
- b entry is a block-type special file;
- c entry is a character-type special file;
- l entry is a symbolic link;
- p entry is a FIFO (also known as named pipe) special file;
- s entry is an AF_UNIX address family socket, or
- \- entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next refers to permissions to others in the same user-group; and the last refers to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, execute permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

- r the file is readable;
- w the file is writable;
- x the file is executable;
- \- the indicated permission is not granted.

The group-execute permission character is given as s if the file has the set-group-id bit set; likewise the owner-execute permission character is given as s if the file has the set-user-id bit set.

The last character of the mode (normally x or '-') is true if the 1000 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. The following options are available:

-a List all entries; in the absence of this option, entries whose names begin with a '.' are *not* listed (except for the privileged user, for whom ls normally prints even files that begin with a '.').

`-A` Same as `-a`, except that '`.`' and '`..`' are not listed.

`-c` Use time of last edit (or last mode change) for sorting or printing.

`-C` Force multi-column output, with entries sorted down the columns; for `ls`, this is the default when output is to a terminal.

`-d` If 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.

`-F` Mark directories with a trailing slash ('`/`'), executable files with a trailing asterisk ('`*`'), symbolic links with a trailing at-sign ('`@`'), and `AF_UNIX` address family sockets with a trailing equals sign ('`=`').

`-g` For `ls`, show the group ownership of the file in a long output.

`-i` For each file, print the i-node number in the first column of the report.

`-l` List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. If the file is a special file the size field will instead contain the major and minor device numbers. If the time of last modification is greater than six months ago, it is shown in the format '*month date year*'; files modified within six months show '*month date time*'. If the file is a symbolic link the pathname of the linked-to file is printed preceded by '`->`'.

`-L` If argument is a symbolic link, list the file or directory the link references rather than the link itself.

`-q` Display non-graphic characters in filenames as the character `?`; for `ls`, this is the default when output is to a terminal.

`-r` Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

`-R` Recursively list subdirectories encountered.

`-s` Give size of each file, including any indirect blocks used to map the file, in kilobytes.

`-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) and/or printing (with the `-l` option).

`-1` Force one entry per line output format; this is the default when output is not to a terminal.

FILES

`/etc/passwd`	to get user ID's for '`ls -l`' and '`ls -o`'.
`/etc/group`	to get group ID for '`ls -g`'

NOTES

NEWLINE and TAB are considered printing characters in filenames.

The output device is assumed to be 80 columns wide.

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 doing this setting would make old shell scripts which used `ls` almost certain losers.

Unprintable characters in file names may confuse the columnar output options.

NAME

ls, lc - list contents of directory

SYNOPSIS

ls [-RadLCxmlnogrtucpFbqisf1] [*names*]
lc [-1CFLRabcfgilmnopqrstux] [*name...*]

DESCRIPTION

For each directory argument, ls lists the contents of the directory for each file argument. lc functions the same as ls except that the lc default output format is columnar, even if the output is redirected. ls repeats its name and any other information requested. The output is sorted alphabetically by default. 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.

There are three major listing formats. The default format for output directed to a terminal is multi-column with entries sorted down the columns. The -1 option allows single column output and -m enables stream output format. 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(4) database is used to determine the number of columns, based on the environment variable TERM. If this information cannot be obtained, 80 columns are assumed.

The ls command has the following options:

-R Recursively list subdirectories encountered.

-a List all entries, including those that begin with a dot (.), which are normally not listed.

-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.

-L If an argument is a symbolic link, list the file or directory the link references rather than the link itself.

-C Multi-column output with entries sorted down the columns. This is the default output format.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format; files are listed across the page, separated by commas.

-l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field instead contains the major and minor device numbers rather than a size. If the file is a symbolic link, the filename is printed followed by "->" and the pathname of the referenced file.

-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 the group is not printed.

`-g` The same as `-l`, except that the owner is not printed.

`-r` Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

`-t` Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See `-n` and `-c`.)

`-u` Use time of last access instead of last modification for sorting (with the `-t` option) or printing (with the `-l` option).

`-c` Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (`-t`) or printing (`-l`).

`-p` Put a slash (`/`) after each filename if the file is a directory.

`-F` Put a slash (`/`) after each filename if the file is a directory, an asterisk (`*`) if the file is an executable, and an ampersand (`@`) if the file is a symbolic link.

`-b` Force printing of non-printable characters to be in the octal *\ddd* notation.

`-q` Force printing of non-printable characters in file names as the character question mark (`?`).

`-i` For each file, print the i-number in the first column of the report.

`-s` Give size in blocks, including indirect blocks, for each entry.

`-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.

`-1` Print one entry per line of output.

The mode printed under the `-l` option consists of ten characters. The first character may be one of the following:

`d` the entry is a directory;
`l` the entry is a symbolic link;
`b` the entry is a block special file;
`c` the entry is a character special file;
`p` the entry is a fifo (named pipe) special file;
`-` the entry is an ordinary file.
`s` the entry is a XENIX semaphore.
`m` the entry is a XENIX shared data (memory).

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.

`ls -l` (the `l`ong list) prints its output as follows:

```
-rwxrwxrwx  1 smith  dev    10876  May 16 9:42 part2
```

Reading from right to left, you see that the current directory holds one file, named `part2`. Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file contains 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group `dev` (perhaps indicating "development"), and his or her login

name is smith. The number, in this case 1, indicates the number of links to file part2; see cp(1). Finally, the dash and letters tell you that user, group, and others have permissions to read, write, and execute part2.

The execute (x) symbol here occupies the third position of the three-character sequence. A - in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

- r the file is readable
- w the file is writable
- x the file is executable
- - the indicated permission is *not* granted
- l mandatory locking occurs during access (the set-group-ID bit is on and the group execution bit is off)
- s the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
- S undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
- t the 1000 (octal) bit, or sticky bit, is on [see chmod(1)], and execution is on
- T the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than x or -. s also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user you login as.

In the case of the sequence of group permissions, l may occupy the third position. l refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by t or T. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

An example of a file's permissions is:

```
-rwxr--r--
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions is:

```
-rwsr-xr-x
```

This describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

Another example of a file's permissions is:

```
-rw-rwl---
```

This describes a file that is readable and writable only by the user and the group and can be locked during access.

An example of a command line:

```
ls -a
```

This command prints the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

Another example of a command line:

```
ls -aisn
```

This command provides information on all files, including those that begin with a dot (a), the i-number—the memory address of the i-node associated with the file—printed in the left-hand column (i); the size (in blocks) of the files, printed in the column to the right of the i-numbers (s); finally, the report is displayed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	user IDs for ls -l and ls -o
/etc/group	group IDs for ls -l and ls -g
/usr/share/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod(1), find(1)

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the ls -l command leads you to believe.

Unprintable characters in file names may confuse the columnar output options.

Permuted Index

MOTOROLA

The reference manual set for UNIX System V Release 4 for Motorola Processors is the definitive source for complete and detailed specifications for all System V interfaces. Retitled and reorganized, this edition makes finding the manual page you need fast and easy. The following table reflects these changes.

Commands Reference Manual Volumes 1 and 2

- General-purpose user commands
- Basic networking commands
- Form and Menu Language Interpreter (FMLI)
- System maintenance commands
- Enhanced networking commands
- Miscellaneous reference information related to commands

System Files and Devices Reference Manual

- System file formats
- Special files (devices)

Device Driver Interface/Driver-Kernel Interface Reference Manual

- Driver Data Definitions
- Driver Entry Point Routines
- Kernel Utility Routines
- Kernel Data Structures
- Kernel Defines

System Calls and Library Functions Reference Manual

- System calls
- BSD system compatibility library
- Standard C library
- Executable and linking format library
- General-purpose library
- Math library
- Networking library
- Standard I/O library
- Specialized library
- Miscellaneous reference information related to programming

Master Permuted Index

- Permuted index of all manual pages

ISBN 0-13-088832-X

9 780130 888327 90000

UNIX PRESS

A Prentice Hall Title