Data General

Data General Corporation, Westboro, Massachusetts 01580

Customer Documentation

User's Reference for the DG/UX™ System

093-701054-03

AViiON®

PRODUCT LINE

User's Reference for the DG/UX™ System

093-701054-03

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

User's Reference for the DG/UX System

093-701054-03

Revision History:	Effective with:
Original Release – February 1990	DG/UX 4.20
Revision 1 – June 1990	DG/UX 4.30
Revision 2 – June 1991	DG/UX 5.4
Revision 3 – February 1992	DG/UX 5.4.1

Preface

This *User's Reference for the DG/UX™ System* describes the commands that constitute the basic software running on Data General AViiON® computers.

This manual is part of a five-volume reference set. The other manuals are the *System Manager's Reference for the DG/UX System* and the three-volume *Programmer's Reference for the DG/UX System*. These manuals contain in printed (typeset) form the online entries released with the DG/UX System in **/usr/catman** for access by the **man** command.

A more complete discussion of the user's environment is contained in *Using the DG/UX System* and *Using the DG/UX Editors*. Other related manuals are listed under "Related Manuals" at the end of this manual.

Man Pages

For historical reasons, each entry is called a "manual page" or "man page," though an entry may occupy more than one physical page and may contain more than one entry. If the man page contains more than one entry, it is alphabetized under its "primary" name; for example, the **rm** manual page describes the **rm** and **rmdir** commands.

Manual pages are assigned to classes ranging from 0 through 8 for easy cross-reference. The class number appears in parentheses following the name; for example, in **rm**(1) the (1) indicates that **rm** is a command. Some classes are subdivided with letters; for example, (1M) indicates a command manual page that is in the *System Manager's Reference*.

A command followed by a (1) or (1G) usually means that it is described in this manual. (Class 1 commands appropriate for use by programmers are located in the *Programmer's Reference*.) A man page name with a (1M), (4M), (7), or (8) following it means that the entry is in the *System Manager's Reference*. Names with (2) or (3*x*), (4), (5) [except **editread**(5)], or (6F) are in the *Programmer's Reference*. Occasionally, DG/UX man pages refer to other products' man pages, which are not part of the DG/UX documentation; these are so noted.

Manual Organization

The *User's Reference* has two chapters containing man pages in classes (1) and (5):

Chapter 1: Commands (1)
The entries in Chapter 1 describe programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are called by the user's programs. Commands generally reside in the directories **/usr/bin** (for binary programs) and **/usr/sbin**. In addition, some commands reside in **/sbin**. These directories are searched automatically by the command interpreter called the *shell*. Also, DG/UX systems often have a directory called **/usr/lbin**, containing local commands.

Chapter 1 begins with an **intro**(1) entry. The remaining entries are alphabetized.

Chapter 2: Miscellaneous Features (5)
This chapter contains the **editread**(5) manual page. Editread is a command-line editor available in the Bourne and C shells and certain other programs.

Appendix A: Contents and Permuted Index Man Pages
These manual pages contain information extracted from the DG/UX man pages in all five reference volumes.

Man Page Format

Each man page has at least some of the following sections:

NAME	gives the primary name (and secondary names, as the case may be) and briefly states its purpose.
SYNOPSIS	summarizes the usage of the program being described.
DESCRIPTION	discusses how to use these commands.
EXAMPLES	gives examples of usage, where appropriate.
FILES	contains the file names that are referenced by the program.
EXIT CODES	discusses values set when the command terminates. The value set is available in the shell environment variable "?" (see **sh**(1)).
DIAGNOSTICS	discusses the error messages that may be produced. Messages that are intended to be self-explanatory are not listed.
SEE ALSO	offers pointers to related information.
NOTES	gives information that may be helpful under the particular circumstances described.

Some man pages may contain other heads such as **ENVIRONMENT** and **CAVEATS**.

Man Page Notation Conventions

This manual uses certain symbols and styles of type to indicate different meanings in man pages. Those symbol and typeface conventions are defined in the following list. You should familiarize yourself with these conventions before reading the manual.

The description of convention meanings uses the terms "command line," "format line," and "syntax line." A command line is an example of a command string that you should type verbatim; it is preceded by a system prompt. A format line shows how to structure a command; it shows the variables that must be supplied and the available options. A syntax line is a fragment of program code that shows how to use a particular routine; some syntax lines contain variables.

Convention	Meaning
boldface	This font is used for section heads and subsection heads. It is also used to distinguish input from output in examples where the two are intermixed.
`constant width/ monospace`	In command formats and code syntax: This typeface indicates text (including punctuation) that you type verbatim from your keyboard. In text: This typeface is used for examples, code samples, pathnames, and the names of commands, files, directories, and manual pages. In all contexts: The following characters, which have special meanings explained below, do not have special meaning but simply represent themselves when they appear in constant-width font: `<` `>` `[` `]` `{` `}` `|`. In constant-width font they are are I/O redirection operators, brackets, braces, and the pipe symbol.
italic	In format lines: This font represents variables for which you supply values; for example, the names of your directories and files, your username and password, and possible arguments to commands.
[*optional*]	In format lines: Regular-font brackets surround an optional argument. Don't type the brackets; they only set off what is optional. These brackets should not be confused with constant-width brackets.
choice1|*choice2*	In format lines: The vertical bar indicates a choice between *choice1* and *choice2*.
...	In format lines and syntax lines: You can repeat the preceding argument as many times as desired.
{ }	In format lines: These regular-font braces surround either two or more choices or syntax elements that are repeatable as a group.
< >	In command lines and other examples: Angle brackets distinguish a command sequence or a keystroke (such as <**Ctrl-D**>, <**Esc**>, and <**3dw**>) from surrounding text. Note that these angle brackets are in regular type and that you do not type them; there are, however, constant-width versions of these symbols that you do type.
`$, %, #`	In command lines and other examples: These symbols represent the system command prompt symbols used for the Bourne and Korn shells, the C shell, and the superuser, respectively. Note that your system might use different symbols for the command prompts.

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please use the enclosed TIPS order form (United States only) or contact your local Data General sales representative. A list of related documents appears at the end of this manual with the TIPS order form.

For a complete list of AViiON® and DG/UX™ manuals, see the *Guide to AViiON® and DG/UX™ System Documentation* (069-701085). The on-line version of this manual found in **/usr/release/doc_guide** contains the most current list.

Telephone Assistance

If you are unable to solve a problem using any manual you received with your system, free telephone assistance is available with your hardware warranty and with most Data General software service options. If you are within the United States or Canada, contact the Data General Customer Support Center (CSC) by calling 1-800-DG-HELPS. Lines are open from 8:00 a.m. to 5:00 p.m., your time, Monday through Friday. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining Our Users Group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive FOCUS monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual Member Directory, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1-800-932-6663 or 1-508-443-3330.

End of Preface

Contents

Chapter 1 — User Commands and Application Programs

Tables

Chapter 1
User Commands and Application Programs

This chapter contains reference entries documenting DG/UX, TCP/IP, and ONC/NFS user commands and application programs. These pages are also supplied on the product release tape and can be accessed online via the **man** command.

The first entry, **intro**(1), is an introduction that provides an overview of the DG/UX commands and application programs. It describes how the commands are categorized and explains the categories and the kinds of user needs represented in the categories. The remaining entries are in alphabetical order.

The following DG/UX man pages are new for Revision 03:

chgrp(1)
exstr(1)
fmtmsg(1)
idc(1)
strchg(1)

In addition, the TCP/IP and ONC/NFS man pages for user commands have been added to this manual. Table 1-1 summarizes the TCP/IP and ONC/NFS user commands.

Table 1-1 Manual Pages for TCP/IP and ONC/NFS User Commands

Product	Command	Description
TCP/IP	**bftp**(1C)	Run the Background File Transfer Program.
	ftp(1C)	Run the File Transfer Protocol program.
	hostid(1C)	Set or print identifier of current host system.
	hostname(1C)	Set or print name of current host system.
	rcp(1C)	Copy files between hosts.
	remsh(1C)	Execute a command on a remote host.
	rlogin(1C)	Log in to another host on the network.
	ruptime(1C)	Show host status of local machines.
	rwho(1C)	Show who is logged in to hosts on local network.
	telnet(1C)	Log in to another host on the network.
	tftp(1C)	Run the Trivial File Transfer Program.
ONC/NFS	**chkey**(1)	Change your encryption key.
	domainname(1)	Set or display name of current NIS domain.
	keylogin(1)	Decrypt and store secret key.
	on(1C)	Execute command remotely but with local environment.
	rpcgen(1)	An RPC protocol compiler
	rup(1C)	Show host status of local machines (RPC version).
	rusers(1C)	Show who is logged in to local machines (RPC version).
	rwall(1C)	Write to all users over a network.
	ypcat(1)	Print values in an NIS database.
	ypmatch(1)	Print value of one or more keys from NIS map.
	yppasswd(1)	Change your network password in NIS.
	ypwhich(1)	Display which host is NIS server or map master.

NAME

`intro` – introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, publicly-accessible commands.

Command Syntax

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

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

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

Command Syntax Standard: Rules

All new commands will follow the syntax rules below. Because existing commands have been developed at various times by various people, some commands will not follow the rules below. `Getopts`(1) should be used by all shell procedures to parse positional parameters and to check for legal options. `Getopts`(1) supports Rules 3-10 below. The command itself must enforce the other rules.

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 (e.g., `-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.

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, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code," "exit status," or "return code," and is described only where special conventions are involved.

SEE ALSO

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

NOTES

Many commands do not adhere to the aforementioned syntax.

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

acctcom – search and print process accounting file(s)

SYNOPSIS

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

where:

options	One or more of the options listed below under **Options**
file	The name of an input file

DESCRIPTION

Acctcom reads *file*, the standard input, or /usr/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:

command name
user
ttyname
start time
end time
real (sec)
cpu (sec)
mean size(K)

It can optionally show:

f (the fork/exec flag: 1 for fork without exec)
stat (the system exit status)
hog factor
Kcore min
CPU factor
characters transferred
blocks read (total blocks read and written)

The command name is prepended with a # if it was executed with super-user 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), /usr/adm/pacct is read; otherwise, the standard input is read.

If any file arguments are given, they are read left to right. Each file is normally read in chronological order by process completion time. The file /usr/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 /usr/adm/pacct?.

Options

-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 divided by 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/line`.
`-u` *user*	Show only processes belonging to *user*, 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*, which can be 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 `ed`(1) except that + means one or more occurrences.
`-o` *ofile*	Copy selected process records in the input data format to *ofile*; supress standard output printing.
`-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 plus user, exceeding *sec* seconds.
`-q`	Do not print any output records, just print the average statistics as with the `-a` option.
`-I` *chars*	Show only processes transferring more characters than the cut-off number given by *chars*.

EXAMPLES

```
$ acctcom
```

This example will process and display the process accounting file. The output shows the following information for all processes executed since clearing the accounting log file:

command name
user
ttyname
start time

end time
real (sec)
cpu (sec)
mean size(K)

```
$ acctcom -q
cmds=2590 Real=147.66 CPU=1.30  USER=0.88  SYS=0.42  CHAR=54262.10
          BLK=536.25  USR/TOT=0.68            HOG=0.01
$
```

This example will process and display the process accounting file in summary form.

```
$ acctcom -u intern -b
COMMAND                    START    END        REAL      CPU    MEAN
NAME       USER  TTYNAME   TIME     TIME      (SECS)   (SECS) SIZE(K)
sh       intern  tty12    15:22:09 15:22:09    0.09     0.04   38.50
mail     intern  tty12    15:19:16 15:19:25    9.75     0.37   76.00
who      intern  tty12    15:19:09 15:19:10    1.13     0.30   52.40
ps       intern  tty12    15:19:03 15:19:05    2.32     0.52  137.46
mail     intern  tty12    15:18:28 15:18:59   31.92     0.28   79.64
vi       intern  tty12    15:13:05 15:17:58  293.84    35.89  157.32
acctcom  intern  tty12    15:12:51 15:13:01   10.67     7.04  129.75
cp       intern  tty12    15:06:16 15:06:16    0.59     0.08   42.50
more     intern  tty12    15:01:29 15:04:07  158.00     5.11   79.84
acctcom  intern  tty12    15:01:30 15:03:57  147.60    15.31   88.12
vi       intern  tty12    14:53:35 14:53:46   11.32     0.88  269.64
$
```

This example will process and display the process accounting file for user "intern" and display them in reverse order.

```
$ acctcom -n vi
COMMAND                      START    END         REAL      CPU    MEAN
NAME     USER     TTYNAME   TIME     TIME       (SECS)   (SECS) SIZE(K)
vi      intern    tty12     15:39:48 15:41:57   129.44     3.13  263.67
vi      clark     ttyq0     15:44:38 15:46:15    97.28     3.85  253.67
vi      intern    tty12     16:04:27 16:04:55    28.04     1.33  157.83
vi      haal      tty21     16:31:40 16:32:35    55.62     1.52  272.32
vi      harrise   tty07     09:01:37 09:01:55    18.69     0.87  241.84
vi      root      tty07     09:02:10 09:02:22    12.34     0.59  330.31
vi      haal      tty21     13:20:38 13:21:50    72.52     1.73  262.47
vi      mcadams   tty00     13:45:37 13:45:44     7.80     0.88  174.05
vi      clark     ttyq1     14:08:27 14:09:10    43.46     4.73  147.92
$
```

This example will process and display the process accounting file displaying all occurrences of the pattern "vi". This will report on all users that have executed vi.

FILES

```
/etc/passwd
/usr/adm/pacct
/usr/adm/pacct?
/etc/group
```

SEE ALSO

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

BUGS

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

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.

SEE ALSO

`kbdcomp`(1M), `kbdload`(1M), `alp`(7), `att_kbd`(7).

NOTES

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*.

NAME

`apropos` – locate commands by keyword lookup

SYNOPSIS

`apropos` *keyword* ...

where:

keyword A word for which to search, from the **NAME** section of an entry

DESCRIPTION

`apropos` shows which entries of the reference manual contain instances of any of the given *keyword*s in their titles. The **NAME** line of each matching entry is printed to the standard output. 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.

If an `apropos` output line starts with *filename*(*section*[*x*]), where *section* is a digit and *x* is a lowercase letter, you can enter the following command to get the documentation for it:

- `man` *section filename*

EXAMPLES

To display the title lines of all manual entries related to passwords:

```
apropos password
```

To find out what editors are available on the DG/UX System:

```
apropos editor
```

To locate and then display an entry discussing formatted printing subroutines:

```
apropos formatted
```

and then

```
man 3 printf
```

FILES

`/usr/catman/?_man/whatis`
Table of contents data bases

SEE ALSO

`man(1)`, `whatis(1)`.

NOTES

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

NAME

`at`, `batch` – execute commands at a later time

SYNOPSIS

`at` [`-f` *script*] [`-m`] *time* [*date*] [+ *increment*]

`at -l` [*job* ...]

`at -r` *job* ...

`batch`

where:

script	The name of a file containing commands
time	*h*, *hh*, *hhmm*, *h*:*m*, *h*:*mm*, *hh*:*m*, or *hh*:*mm*, where *h* is hours and *m* is minutes. 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`.
date	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. The default is `today` if the given hour is greater than the current hour, `tomorrow` if it is less. If the given month is less than the current month (and no year is given), next year is assumed.
increment	A number suffixed by one of the following: `minutes`, `hours`, `days`, `weeks`, `months`, or `years`. (The singular form is also accepted.) If `next` precedes *increment*, it means '+ 1'.
job	A job name or number

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.

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.

`at` and `batch` write the job number and schedule time to standard error. Both commands read from standard input. `sh`(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

Options

`-f` *script*	Read commands to be executed from the named script file.
`-l` [*job*]	Report all jobs scheduled for the invoking user, or just the jobs specified.
`-m`	Send 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*	Remove specified jobs previously scheduled using `at`.

Access Permissions

Users are permitted to use `at` if their name appears in the file `/etc/cron.d/at.allow`. If that file does not exist, the file `/etc/cron.d/at.deny` is checked to determine whether 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.

Date Format

If the `DATEMSK` environment variable is set, `at` uses its value as the pathname of a template file containing format strings. These format strings determine the valid *time* and *date* values instead of the values described above. The strings consist of field descriptors and text characters and 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)`).

For the allowable list of field descriptors, see `getdate(3C)`. 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.

Removing and Listing Jobs

`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 remove only your own jobs unless you are the privileged user.

EXAMPLES

Valid commands include:

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

This sequence can be used at a terminal:

```
batch
sort filename > outfile
Ctrl-D (hold down CTRL and press '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 `/etc/cron.d`.

```
%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 /etc/cron.d/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

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

DIAGNOSTICS

at can detect syntax errors and times out of range.

SEE ALSO

atq(1), atrm(1), calendar(1), crontab(1), date(1), environ(5), kill(1), mail(1), nice(1), ps(1), sh(1), sort(1).

cron(1M) in the *System Manager's Reference for the DG/UX System.*

getdate(3C) in the *Programmer's Reference for the DG/UX System.*

NAME

`atq` – display the jobs queued to run at specified times

SYNOPSIS

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

where:

username	A valid user name

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.

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) in the *System Manager's Reference for the DG/UX System*.

NAME

`atrm` – remove jobs spooled by at or batch

SYNOPSIS

`atrm` [`-a f i`] *arg* ...

where:

arg	A user name or job number

DESCRIPTION

`Atrm` removes delayed-execution jobs that were created with the `at(1)` command, but not yet executed. To display the list of these jobs and associated job numbers, use `atq(1)`.

`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.

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. Suppress all information regarding the removal of the specified jobs.
`-i`	Interactive. Ask whether 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)` in the *System Manager's Reference for the DG/UX System*.

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.

EXAMPLES

```
$ banner hello world
```

This example prints on the screen "hello world" in large letters on two lines.

```
$ banner "hi world"
```

This example prints on the screen "hi world" in large letters on one line.

SEE ALSO

`echo`(1), `printf`(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

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [-c] [-l] [*file* ...]

DESCRIPTION

Bc is an interactive processor for a language that resembles C but provides essentially unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The -l argument stands for the name of an arbitrary precision math library. The syntax for bc programs is as follows; L means letters a–z, E means expression, and S means statement.

Comments 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(1). Assignments to *ibase* or *obase* set the input and output number radix respectively.

A number is an unbroken string of the digits 0–9 and possibly, extended digits, for radices greater than 10. Extended digits, e.g. A-F in base 16, must be specified as capital letters only.

You can use the same letter 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, you must place empty square brackets after the array name.

`Bc` is actually a preprocessor for dc(1), which it invokes automatically, unless the `-c` (compile only) option is present. In this case, the `dc` input is sent to the standard output instead. `bc` is terminated by Ctrl-D (^d).

EXAMPLES

```
$ bc
scale=5 <NL>
12567/234 <NL>
53.70512
$
```

This example divides 12567 by 234 and prints the result with a precision of 5 decimal places.

FILES

`/usr/lib/lib.b`	Mathematical library
`/usr/bin/dc`	Desk calculator

SEE ALSO

dc(1).

NOTES

`&&` and | | are not implemented in the DG/UX System.
A `for` statement must have all three expressions.
`Quit` 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.

Options

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.

Input and Output

`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`*?????*

DIAGNOSTICS

Use `help` for explanations.

SEE ALSO

`diff(1)`, `help(1)`

NAME

`berk_diff` – Berkeley differential file and directory comparator

SYNOPSIS

`berk_diff [ -l ] [ -r ] [ -s ] [ -cefhn ] [ -biwt ] [ -S`*name* `]` *dir1 dir2*
`berk_diff [ -cefhn ] [ -biwt ]` *file1 file2*
`berk_diff [ -D`*string* `] [ -biw ]` *file1 file2*

DESCRIPTION

If both arguments are directories, `berk_diff` sorts the contents of the directories by name, and then runs the regular file `berk_diff` algorithm (described below) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed. Options when comparing directories are:

`-l` long output format; each text file `berk_diff` is piped through `pr`(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.

`-r` causes application of `berk_diff` recursively to common subdirectories encountered.

`-s` causes `berk_diff` to report files which are the same, which are otherwise not mentioned.

`-S`*name*
starts a directory `berk_diff` in the middle beginning with file *name*.

When run on regular files, and when comparing text files which differ during directory comparison, `berk_diff` tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, `berk_diff` finds a smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, then either may be given as '–', in which case the standard input is used. If *file1* is a directory, then a file in that directory whose file-name is the same as the file-name of *file2* is used (and vice versa).

There are several options for output format; the default output format 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 *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert *file2* into *file1*. As in `ed`, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

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 '>'.

Except for `-b`, `-w`, `-i` or `-t` which may be given with any of the others, the following options are mutually exclusive:

`-e` produces a script of `a`, `c` and `d` commands for the editor `ed`, which will recreate *file2* from *file1*. 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 `berk_diff` need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo ´1,$p´) | ed - $1
```

Extra commands are added to the output when comparing directories with `-e`, so that the result is a `sh`(1) script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2.*

`-f` produces a script similar to that of `-e`, not useful with `ed`, and in the opposite order.

`-n` produces a script similar to that of `-e`, but in the opposite order and with a count of changed lines on each insert or delete command. This is the form used by `rcsdiff`(1).

`-c` produces a berk_diff with lines of context. The default is to present 3 lines of context and may be changed, e.g to 10, by `-c10`. With `-c` the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from *file1* are marked with '– '; those added to *file2* are marked '+ '. Lines which are changed from one file to the other are marked in both files with with '! '.

Changes which lie within *context* lines of each other are grouped together on output. (This is a change from the previous "berk_diff -c" but the resulting output is usually much easier to interpret.)

`-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.

`-D`*string* causes `berk_diff` to create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1,* while defining *string* will yield *file2.*

`-b` causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equal.

`-w` is similar to `-b` but causes whitespace (blanks and tabs) to be totally ignored. E.g., "if (a == b)" will compare equal to "if(a==b)".

`-i` ignores the case of letters. E.g., "A" will compare equal to "a".

`-t` will expand tabs in output lines. Normal or `-c` output adds character(s) to the front of each line which may misalign the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

FILES

`/tmp/d`*?????*
`/usr/lib/diffh` for `-h`
`/bin/diff` for directory diffs
`/bin/pr`

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

SEE ALSO

`berk_diff3`(1), `cc`(1), `cmp`(1), `comm`(1), `diff`(1), `diff3`(1), `ed`(1).

NOTES

Editing scripts produced under the `-e` or `-f` option are naive about creating lines consisting of a single '`.`'.

When comparing directories with the `-b`, `-w` or `-i` options specified, `berk_diff` first compares the files ala `cmp`, and then decides to run the `berk_diff` algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

NAME

`berk_diff3` – Berkeley 3-way differential file comparison

SYNOPSIS

`berk_diff3 [ -exEX3 ]` *file1 file2 file3*

DESCRIPTION

`Berk_diff3` compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

====	all three files differ
====1	*file1* is different
====2	*file2* is different
====3	*file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f : *n1* a	Text is to be appended after line number *n1* in file *f,* where *f* = 1, 2, or 3.
f : *n1*, *n2* c	Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2,* the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a `c` indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the `-e` option, `berk_diff3` publishes a script for the editor `ed` that will incorporate into *file1* all changes between *file2* and *file3, i.e.* the changes that normally would be flagged ==== and ====3. Option `-x` (`-3`) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to 'file1'.

```
        (cat script; echo ´1,$p´) | ed - file1
```

The `-E` and `-X` are similar to `-e` and `-x`, respectively, but treat overlapping changes (i.e., 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.

For example, suppose lines 7-8 are changed in both file1 and file2. Applying the edit script generated by the command

```
berk_diff3 -E file1 file2 file3
```

to file1 results in the file:

```
lines 1-6
of file1
<<<<<<< file1
lines 7-8
of file1
=======
lines 7-8
of file3
>>>>>>> file3
rest of file1
```

The `-E` option is used by `rcsmerge`(1) to insure that overlapping changes in the merged files are preserved and brought to someone's attention.

FILES

`/tmp/d3?????`
`/usr/lib/berk_diff3`

SEE ALSO

`berk_diff`(1), `rcsmerge`(1).

NOTES

Text lines that consist of a single '.' will defeat `-e`.

NAME

`bfs` – big file scanner

SYNOPSIS

`bfs` [–] *name*

DESCRIPTION

The `bfs` command is like `ed`(1), but 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. `Bfs` is usually more efficient than `ed` for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where `csplit`(1) can 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 you type `P` and a newline as in `ed`. Turn prompting off again by inputting another `P` and newline. Note that 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. For 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` will both print the first ten lines. The `f` command prints only the name of the file being scanned; there is no *remembered* file name. The `w` command is independent of output diversion, truncation, or crunching (see the `xo`, `xt` and `xc` commands, below). The following additional commands are available:

`xf` *file*
: Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received, or an error occurs, reading resumes with the file containing the `xf`. 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 mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion truncates or creates 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 also inserts 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 if: Either address is not between `1` and `$`.

  The second address is less than the first, or

  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*. If the command fails, the jump is not made and the next statement is executed. This command is the only one that does not issue an error message on bad addresses, so you can use it to test whether addresses are bad before other commands are executed. Note that the command

```
xb/^/ label
```

is an unconditional jump.
The `xb` command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe, only a downward jump is possible.

`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`:

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

will all print the first 100 lines.

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

would globally search for the characters `100` and print each line containing a match. To escape the special meaning of `%`, a \ must precede it.

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

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

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

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

would put the current line into variable `5`, print it, and increment the variable `6` by one. To escape the special meaning of `!` as the first character of *value*, precede it with a \.

xv7\!date

stores the value `!date` into variable `7`.

xbz *label*

xbn *label*

These two commands will test the last saved *return code* from the execution of a DG/UX system command (!*command*) or nonzero value, respectively, to the specified label. The two examples below search for the next five lines containing the string size.

```
xv55
: 1
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2

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

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.

DIAGNOSTICS

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

SEE ALSO

csplit(1), ed(1).
regcmp(3X) in the *Programmer's Reference for the DG/UX System*

NAME

`bftp` – Background File Transfer Program

SYNOPSIS

`bftp`

DESCRIPTION

`bftp` is the user interface to the Background File Transfer Program (BFTP). `bftp` may be used to submit a request to have a file transferred at some time in the future via the standard internet File Transfer Protocol (FTP), which is described in RFC-959.

BFTP makes use of third party FTP, so the source and the destination hosts do not have to be operational at the time the request is submitted. At least one of the hosts must correctly support the PASV command of the FTP protocol. Transfers are scheduled locally via the system batch processor, `at`.

For more information on BFTP see *Using TCP/IP on the DG/UX™ System*, *Managing TCP/IP on the DG/UX™ System*, and RFC-1068 (BFTP).

Bftp Standard Transfer Commands

`ddir` *directory_name*
: Sets the destination directory. If `ddir` is not set and `dfile` is not a complete pathname, `dfile` will be relative to the user's home directory on the destination host.

`dfile` *destination-filename*
: Sets the destination filename. Can be a full or a relative pathname. If `ddir` is not set and `dfile` is not a complete pathname, the pathname will be relative to `$HOME` on the destination host.

`dhost` *destination-hostname user password*
: Sets the destination host, user, and password. If the destination user does not have a password, the password argument is not required.

`prompt`
: Prompts you for all commonly-used parameters. This combines `shost`, `sdir`, `sfile`, `dhost`, `ddir`, `dfile`, `dhost`, `ddir`, `dfile`, `set type`, and `set copy` | `move` | `delete`.

`sdir` *directory_name*
: Sets the source directory. If `sdir` is not set and `sfile` is not a complete pathname, `sfile` will be relative to the user's home directory on the source host.

`sfile` *file_name*
: Sets the source filename. Can be a full or a relative pathname. If `sdir` is not set and `sfile` is not a complete pathname, the pathname will be relative to `$HOME` on the source host.

`shost` *hostname/number user password*
: Sets the source host, user and password. If the source user does not have a password, the password argument is not required.

`submit`
: Submits the current request for background FTP transfer. `bftp` will prompt for the *StartTime*, *ReturnMailbox*, and *RequestKeyword*.

`transfer`
: Perform the current request in the foreground.

Bftp Information Commands

`?` List the legal options.

`explain`
Displays a short explanation of how to use BFTP.

`help` [*command*] Prints local help information. If a command is supplied as an argument, prints information only on that command.

`status`
Lists the transfers that are currently submitted and provides a summary of each transfer. Use the `find` command for more detailed information on a transfer.

`verify`
Makes the connections necessary to conduct the current transfer, using the specified parameters. Does not make the transfer, but checks the parameters.

`show` Displays the current parameter values.

Bftp Transfer Control Commands

`cancel`
Prevents the specified transfer from taking place. Unlike the `find` command, `cancel` also works after the transfer has begun. This command requires that the source host be running DG/UX 4.30 or higher. To check the version you are running, invoke the `verify` command with `verbose` set to true.

`clear` Returns all parameters to their default values.

`find` Finds and displays the parameters for a transfer request and a log summarizing transfer activity. `bftp` will prompt for the (optional) *RequestID* and the *RequestKeyword*. Once a request has been located and displayed, it can be changed and resubmitted, or canceled.

`hold` Suspends a transfer that is currently active (`Running` and not between retries). This may be used to ease congestion on a slow data link between the two hosts. This command requires that the source host be running DG/UX 4.30 or higher. To check the version you are running, invoke the `verify` command with `verbose` set to true.

`quit` Returns all parameters to their default values and exits the BFTP program.

`unhold`
Restarts a transfer that has been suspended by the `hold` command. This command requires that the source host be running DG/UX 4.30 or higher. To check the version you are running, invoke the `verify` command with `verbose` set to true.

Bftp Request Commands

`request delete` *name*
Deletes request file `bftp-save.`*name*.

`request list`
Lists all request files.

`request load` *name*
Reads `bftp-save.`*name* in as the current request.

`request store` *name*
Saves the current request in a file named `bftp-save.`*name*. Currently,

name can consist of numbers and letters only.

Bftp Set Commands

`set account` *account-name*
: Sets the account for logging in to the source and destination hosts. Many hosts do not require this.

`set append true |false`
: Sets to true or false the request to append transferred file to destination files. If the destination file does not exist, the file is created. The default is false.

`set copy`
: Source file will be copied to the destination filename. Copy is the default.

`set delete`
: Source file will be deleted. Note that when delete is set, no connection is made to the destination host, so only source parameters are required.

`set mailbox` *mailbox-name*
: Sets the mailbox where BFTP transfer results are returned. The mailbox should be in standard internet format, for example: `farah@doc`. The default is *username@host*.

`set mode stream | block | compress`
: Sets the FTP transfer mode to stream, block, or compress. The default mode is stream.

`set move`
: When `set move` is specified, the source file will be deleted after it has been copied.

`set multiple true | false`
: Sets to true or false the request to transfer multiple files. To use wildcards in sourcefile names (for example, datafile*), `multiple` must be set to true. The default is false.

`set port source` *n* `| destination` *n*
: Sets the port for the source or destination system of FTP connection. The default is 21 for both source and destination.

`set structure file | record | page`
: Sets the FTP structure to file, record, or page. The default is file.

`set time` *StartTime retry-interval maximum-retries*
: Sets the start time, the starting retry interval, and the maximum number of tries for a transfer. The default time is now, the default retry interval is 15 minutes, and the default number of tries is 5. Each time that a transfer is retried following a failure, the retry interval is doubled, up to a maximum of 4 hours. You must press the New Line key after *StartTime* because *StartTime* may contain spaces. BFTP prompts you for the retry-interval and maximum number of tries.

`set type image | ascii |ebcdic | local`
: Sets the FTP type and format and byte size parameters. Note that a normal text file is usually `ascii`, and binary file is often the same as an image file. The default is `ascii` and `nonprint`.

: The representation type may be one of network ASCII, EBCDIC, image, or local byte size with a specified byte size (for PDP-10's and PDP-20's mostly). The network ASCII and EBCDIC types have a further subtype which specifies whether vertical format control (NEWLINE characters, form feeds, etc.) are to be passed through (nonprint), provided in TELNET format, or provided in ASA carriage control format.

`set unique true | false`
: Sets to true or false the request to use the STOU command. If the STOU command is supported by the destination host, the file will be stored into a file having a unique filename. The default is false.

`set verbose true | false`
: Sets to true or false the request to display full FTP conversations for the `verify` and `transfer` commands. The default is false. Transfers run by the `submit` command always run as if `verbose` is true.

Special Editing Characters

`<return>`	Accept current command/field.
`<escape>`	Complete current command/field, or display default.
`<space>`	Complete and delimit current command/field.
`<delete>`	Erase last character.
`<control-L>`	Refresh screen.
`<control-R>`	Refresh line.
`<control-U>`	Erase line.
`<control-W>`	Erase current token.

FILES

`bftp` creates a number of files that are used to keep track of requests that are in progress:

```
bftp123456789.atjob
bftp123456789.cmd
bftp123456789.list
bftp123456789.msg
bftp123456789.req
bftp_saved_info
```

The files that are saved via the `request save` command are as follows:

`bftp-save.`*request-name*

`bftp` usually stores its files in the home directory of the user who is logged on. To have `bftp` store these files in another directory, use the system `setenv` command to set `$BFTPDIR`, for example

`setenv BFTPDIR` *~/var/spool/bftp/yourname*

`/etc/bftp.conf`, sets the maximum number of simultaneous transfers controlled by this host. This can be used to limit network congestion. No file, or a file containing the value 0 means no limit.

`/usr/bin/fts` (File Transfer Service) is the program that actually coordinates the transfer. It should only be invoked via BFTP.

SEE ALSO

`at`(1), `cron`(1M), `crontab`(1M), `ftp`(1C), `ftpd`(1M).

NOTES

Some hosts do not correctly support the FTP PASV command. This may cause a `Malformed PASV reply` or a `Connection refused` error.

Transfers from a DG/UX 4.20 source host may not always complete, depending on the `mode`, `structure`, and `type` selected.

NAME

`cal` – print calendar

SYNOPSIS

`cal` [[*month*] *year*]

where:

month	An integer from 1 to 12
year	An integer from 1 to 9999

DESCRIPTION

`Cal` prints a calendar for the specified year. If you also give a month, a calendar just for that month is printed. If you give neither, a calendar for the present month is printed.

Note that `cal 85` refers to 0085, not 1985.

EXAMPLES

```
$ cal
    June 1991
 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

`Cal` with no arguments prints the current month.

```
$ cal 9 1752
   September 1752
 S  M Tu  W Th  F  S
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

The example above shows the transition from the Julian calendar to the Gregorian calendar.

SEE ALSO

`calendar(1)`.

NOTES

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

The transition from Julian to Gregorian is computed as being in September 1752, when the British Empire, including the American colonies, converted. Various countries switched in October 1582, February 1918, or at other times.

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, etc., 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 content 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), date(1), crontab(1), mail(1).
cron(1M), environ(5) in the *System Manager's Reference for the DG/UX System*.

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

`cat` – concatenate and type files to standard output

SYNOPSIS

`cat [ -u ] [ -s ] [ -v [ -t ] [ -e ] ] [ -|`*file*` ] ...`

where:

file Name of file being typed

DESCRIPTION

`cat` reads each *file* (from left to right) and writes it on the standard output. If no input file is given, or if the argument is `-`, `cat` reads from standard input.

When the standard input is the keyboard and the standard output is the screen, `cat` prints back each line as you enter it (the new-line character and all other special characters cannot be escaped). `cat` does not interpret characters.

Using `cat >file1` is a good way to create short files quickly. Type `^d` (Ctrl-D) to end input to the file.

Options are:

`-u` Unbuffered output. The output is buffered unless you give this option.

`-s` Be silent about non-existent files; no error message is given.

`-v` Visible printing of nonprinting characters, except tabs (Ctrl-I), new lines (Ctrl-J), and form feeds (Ctrl-L). Control characters are printed `^`*X* (representing Ctrl-*X*); the DEL character (octal 0177) is printed `^?`. Non-ASCII characters (with the high bit set) are printed as `M-`*x*, where *x* is the character specified by the seven low-order bits.

`-t` Print each tab as `^I`, but only if the `-v` option is also present. Otherwise, it is ignored.

`-e` Print a `$` character at the end of each line (prior to the newline), but only if the `-v` option is also present. Otherwise, it is ignored.

International Features

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

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

EXAMPLES

```
cat file
```

prints the file on the screen.

```
cat file1 file2 > file3
```

concatenates the first two files and places the result on the third.

```
$ cat file1
     The
     quick
     brown
     fox
$ cat file2
     jumped
     over
     the
     lazy
     dog.
$ cat file1 file2 > file3
$ cat file3
     The
     quick
     brown
     fox
     jumped
     over
     the
     lazy
     dog.
```

The above example shows the concatenation of two different files into one file.

SEE ALSO

cp(1), head(1), more(1), pg(1), pr(1), tail(1).

NOTE

Sh(1) creates and/or opens the files for the output of the cat command before reading the files for its input. Therefore, command formats such as

`cat` *file1 file2* > *file1*

cause the original data in *file1* to be lost; take care when using the shell special characters to specify files for cat to use.

NAME

`catexstr` – extract strings from source files, replace with catgets calls.

SYNOPSIS

`catexstr` [`-l`*lang*] [`-c`*cat*] [`-b`*beg*] [`-e`*end*] *file* ... > *strings*
`catexstr -r` [`-l`*lang*] [`-c`*cat*] [`-b`*beg*] [`-e`*end*] *file* < *strings* > *file.new*

DESCRIPTION

The `catexstr` utility is used to extract strings from source files and replace them with calls to the X-Open-style message retrieval function or command (see `catgets`(1,3C)), and generate a message catalog (.msg file) that contains the messages. The .msg file can then be translated into other natural languages. The source files may contain C language source, or source code in other languages, such as shell scripts.

`Catexstr` has the following options:

`-r`	Runs pass two of `catexstr` (replace mode), generating a new version of the source file on the standard output, and simultaneously generating a message catalog (.msg file).
`-l`*lang*	Specifies the source code language of the file(s) being manipulated. The choices that are recognized are *c*, *sh* (shell script), and *gen* (generic). The `-l` option establishes values to be used as the format of the string and the name of the catalog to be inserted into the new source file, and the strings that will be recognized as the beginning and end of comments. These may be overridden with the other options listed here.
`-f`*fmt*	Specifies the format string to be used when creating the modified version of the source code file. The default formats for various languages are shown below.
`-c`*cat*	Specifies the catalog name used when creating the modified version of the source code file. This name is inserted into the source code file; it is *not* used as the name of the .msg file to be created.
`-b`*beg*	Specifies the string to be treated as the beginning of a comment.
`-e`*end*	Specifies the string to be treated as the end of a comment. This may be one or two bytes long. Nesting of comments is not recognized.

If none of `-l`, `-f`, `-c`, `-b`, or `-e` are specified, then `-lc` is assumed (for compatibility with earlier versions of `catexstr`). If a source code language is specified with `-l`, then the default values associated with that language (shown below) are assumed. These defaults may be overridden with the other options described above. If `-l` is not used, but one of `-f`, `-c`, `-b`, or `-e` are, then `-lgen` is assumed. The default values for each of the supported languages are:

Lang	Format string	catalog name	comment begin	comment end
c	catgets(%s, %d, %d, "%s")	catd	/*	*/
sh	‘catgets %s %d %d "%s"‘	*	#	\n
gen	catgets %s %d %d "%s"	*	none	\n

The parameters passed to `sprintf` in conjunction with the format strings are, respectively:

the catalog name, as specified here or with the `-c` option;
the message set number;
the message number; and
the message text.

* For languages *sh* and *gen*, the default catalog name is the name of the source file (with any existing extension stripped off), and `.cat` appended.

In pass one (without the `-r` option), `catexstr` extracts a list of strings from the named source files, with positional information. This list is produced on standard output in the following format:

file:line:position:length:setnum:msgnum:"string"

file	the name of the source file
line	line number in the file
position	character position in the line
length	length of the original string
setnum	null
msgnum	null
string	the extracted, modified text string, surrounded by double quotes.

Normally you would redirect this output into a file (the "message list file", shown as *strings* on the command line above). Then you would edit this file as described below. Then you would use `catexstr -r` to generate a new version of the source file, and a message (.msg) file.

Any '%' characters in the source file that are not part of a "%%" pair will be translated into "%nn$" sequences in the message list file, where the "nn" numbers enumerate the uses of '%' in the message. For example, the message
"File %s has %d blocks."
would become
"File %1$s has %2$d blocks."
This allows the human translator to modify the order of the '%' tokens in the message to accommodate the syntax requirements of the target natural language, while still accommodating the order of the parameters to the printf call. If the message has only one occurrence of '%', then this modification is not really necessary, but it is done anyway.

Next, examine this list and determine which messages can be translated and subsequently retrieved by `catgets`. Modify this message list file by deleting lines that can't be translated. In particular, text associated with '#include "filename"' lines must be deleted, and '#define foo "bar"' lines must be scrutinized.

If you wish to specify the set number(s) and message number(s) to use (see `gencat`(1)), you may do so by inserting these numbers into the fifth (*setnum*) and sixth (*msgnum*) fields in the message list file. If you do not specify the set number to use for a particular message, set number one is used, unless some other set has been specified for an earlier message, in which case that set number is used. If you do not specify any message numbers, the messages are numbered sequentially, starting with number one. If any message is explicitly numbered, that number is used for that message, and automatic numbering resumes from that number.

You are free to modify the text of the message in the message list file in any other way that you consider appropriate. For example, you might use this occasion to clarify an ambiguous English sentence. Make sure that the text is enclosed in double quotes ("). Do not modify any of the first four fields on these lines, even if you change the length of the message.

The message list file should *not* be translated into any other natural language. The file to translate into other languages is the message file (.msg file) that will be produced by the second pass of `catexstr`.

Note, however that you must not make *any* modifications to the source file between running the first and second passes of `catexstr`.

After editing the message list file, use this modified message list file as input to `catexstr -r` *file*. You should provide the same set of options (except `-r`) to this second pass of `catexstr` that you gave to the first pass. The second pass of `catexstr` will produce a new version of the original source file, in which the messages have been replaced by calls to the message retrieval function or command `catgets`. At the same time, a message file that is of the correct format to be used as input to `gencat` is generated, with the name *file*`.msg`.

If you are manipulating `C` source code, then once the new version of the `.c` file has been created, you must edit it to include a declaration for the catalog descriptor variable (normally `catd`) as type `nl_catd`. This variable is used in the calls to `catgets` (see `catgets`(3C)). Usually, you would declare one `catd` variable and use it throughout the program. Also, you must add a call to `catopen`. Generally this is at the top of the `main` routine (see `catopen`(3C)). You may also wish to add a call to `catclose`. The program must also call `setlocale` (see `setlocale`(3C)) if it does not do so already. This will probably entail inclusion of `locale.h`.

The `catexstr` program cannot correctly replace strings in all instances. For example, a static character string initialization cannot be replaced by a call to `catexstr`. A second example is an escape sequence which should not be translated. In some cases the C code may require modification so that strings can be extracted and replaced by calls to the message retrieval function.

Shell Scripts

Shell scripts present a variety of challenges. Here are a few pointers in dealing with them.

Before running the first pass of `catexstr`, examine the shell script for back-quote (‘) characters within double-quoted strings (strings enclosed in double-quote marks ("')). Such occurrences will not be handled correctly by `catexstr`, and must be modified either before or after running `catexstr`.

Also look for strings that should be translated, that are not enclosed in double quotes. This includes strings enclosed in single quotes (’).

Similarly, look for strings that must be passed as a single argument to a command, rather than being broken into separate arguments (words) by the shell. Such cases can be handled by assigning the value of the string to a temporary shell variable, and then using the shell variable in the call to the command. For example,

```
    log_error "This must be one argument, not seven."
```

becomes

```
    msg = "This must be one argument, not seven."
    log_error "$msg"
```

which ends up looking something like:

```
    msg = ‘catexstr mycat.cat 1 15 \
       "This must be one argument, not seven."‘
    log_error "$msg"
```

After running the first pass of `catexstr`, search the message list file for any occurrence of a back-quote character. Any such occurrence, as mentioned above, must be changed. This may be done by either modifying the original source and re-running the first pass of `catexstr`, or by modifying the new source file after running the second pass of `catexstr`.

After running both passes of `catexstr`, edit the new source file and examine each call to `catgets`, to make sure that it makes sense. One particular optimization that can frequently be made is, for example, to change

echo 'catgets mycat.cat 1 16 "Hello, world."'

to

catgets mycat.cat 1 16 "Hello, world."

EXAMPLES

The following examples show uses of `catexstr` to convert a C program.

Assume that the file `hw.c` contains:

```
main()
{
    printf("This is an example\n");
    printf("Hello world!\n");
    printf("This is the %s string (number %d)\n", "third", 3);
}
```

`catexstr hw.c > hw.strings` produces the following output in the file `hw.strings`:

```
hw.c:3:8:20:::"This is an example\n"
hw.c:4:8:14:::"Hello world!\n"
hw.c:5:8:35:::"This is the %1$s string (number %2$d)\n"
hw.c:5:47:5:::"third"
```

The file `hw.strings` can be edited as described above.

The `catexstr` utility can now be invoked with the `-r` option to replace the strings in the source file by calls to the message retrieval function `catgets`().

`catexstr -r hw.c <hw.strings >hw.new.c` produces the following output (the indentation has been modified to fit on this manual page):

```
#include <nl_types.h>
main()
{
printf(catgets(catd, 1, 1, "This is an example\n"));
printf(catgets(catd, 1, 2, "Hello world!\n"));
printf(catgets(catd, 1, 3, "This is the %1$s string (number %2$d)\n"), \
catgets(catd, 1, 4, "third"), 3);
}
```

This new source file must be edited to include a declaration of `catd` (as type nl_catd), a call to `catopen`, and possibly calls to `setlocale` and `catclose`. You may also wish to break the long line:

```
#include <nl_types.h>
#include <locale.h>
static nl_catd catd;

main()
{
    (void) setlocale (LC_ALL, "");
    catd = catopen ("hw.cat", 0);
    printf(catgets(catd, 1, 1, "This is an example\n"));
    printf(catgets(catd, 1, 2, "Hello world!\n"));
    printf(catgets(catd, 1, 3, "This is the %1$s string (number %2$d)\n"),
            catgets(catd, 1, 4, "third"), 3);
        catclose (catd);
    }
```

The `catexstr -r` command above also produces a message file, `hw.msg`:

```
$quote "
$set 1
1   "This is an example\n"
2   "Hello world!\n"
3   "This is the %1$s string (number %2$d)\n"
4   "third"
```

This message file may be replicated and translated into other natural languages.

The following command is used to compile the message catalog:

```
rm hw.cat; gencat hw.cat hw.msg
```

The resulting message catalog (`hw.cat`) must be installed in the appropriate directory. Normally, this would be a subdirectory of `/usr/lib/nls/msg`.

Multiple Source Files

Programs that consist of more than one source file should be handled as follows. First, `catexstr` is called with all the source files as arguments:

```
catexstr foo1.c foo2.c > foo.strings
```

Second, the message list file (`foo.strings`) is edited as described above.

Third, `catexstr -r` is called once for each source file, to create new source files and message (`.msg`) files:

```
catexstr -r foo1.c < foo.strings > foo1.new.c
catexstr -r foo2.c < foo.strings > foo2.new.c
```

Fourth, `gencat` is called to compile the message catalog:

```
rm -f foo.cat
gencat foo.cat foo1.msg foo2.msg
```

FILES

`/usr/lib/nls/msg/`*locale*/*catalog*`.cat`   files created by `gencat`(1)

ENVIRONMENT VARIABLES

`NLSPATH`	specification of directory containing the locale-specific message catalog directories.
`LANG`	locale name.

DIAGNOSTICS

The error messages produced by `catexstr` are intended to be self-explanatory. They indicate errors in the command line or format errors encountered within the input file.

SEE ALSO

`catgets`(1), `gencat`(1),
`catopen`(3C), `catclose`(3C), `catgets`(3C), `printf`(3S), `setlocale`(3C) in the *Programmer's Reference for the DG/UX System*.
`environ`(5) in the *System Manager's Reference for the DG/UX System*.
`exstr`(1) — AT&T-style message facility.

NAME

`catgets` – print message from message catalog

SYNOPSIS

`catgets` *catalogname setnumber messagenumber defaultmessage*

where:

catalogname	Is the name of the compiled message catalog (e.g. `perror.cat`).
setnumber	Is the number of the message set within the message catalog.
messagenumber	Is the number of the message within the set.
defaultmessage	Is the default string to use if the message catlog is not available, or the specified message is missing. This string must be a single argument to `catgets`, which means that if it contains any space(s), it must be quoted.

DESCRIPTION

The `catgets` command performs a function very similar to the `catgets`(3C) subroutine — it extracts a message from an X/Open-style message catalog. The message catalog to use is selected on the basis of the *catalogname* argument, and the values of the NLSPATH and LANG environment variables (see `catopen`(3C)). The designated string is printed to `catgets`'s standard output. If the designated catalog is not available, or does not contain the message specified by *setnumber* and *messagenumber*, then the *defaultmessage* string is printed.

EXAMPLES

```
catgets lp.shl.cat 1 7 "Enter name of lp device:"
```

This command attempts to retrieve message number 7 from set number 1 in message catalog lp.shl.cat, in the directory specified by the NLSPATH and LANG environment variables. If that is successful, that message is printed; otherwise `Enter name of lp device:` is printed.

FILES

`/usr/lib/nls/msg/$LANG/`*catalogname* Default location of message catalog.

SEE ALSO

`gencat`(1),
`catopen`(3C), `catgets`(3C), `setlocale`(3C),
`environ`(5).
`gettxt`(1) — AT&T-style message facility.

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 (also $home in csh). 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 in csh). $CDPATH has the same syntax as, and similar semantics to, the PATH shell variable (path in csh). Cd must have execute (search) permission in *directory*.

Because a new process is created to execute each non-built-in command, cd would be ineffective if it were written as a normal command; therefore, it is a built-in command for both the Bourne shell and the C shell.

EXAMPLES

```
$ pwd
/usr/user1
$ cd work_dir
$ pwd
/usr/user1/work_dir
$
```

The above example changes your current directory to the directory named "work_dir." "Work_dir" is located below the directory /usr/user1. The string, "..", can be substituted to indicate the directory above the current directory. This string can be repeated on the same command line to go up several levels.

SEE ALSO

pwd(1), sh(1), csh(1).
chdir(2) in the *Programmer's Reference for the DG/UX System (Volume 1).*

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.

Valid options to chgrp are:

-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.

EXAMPLES

```
$ chgrp 1009 chapter
```

If you own a file **chapter**, the new group will be the group named by the numeric group ID **1009**. **1009** must be a valid group ID listed in the /etc/group file.

```
$ chgrp work *
```

This command changes the group for all the files you own in the current directory. The new group will be the group with the group name **work**. **work** must be a valid group name listed in the /etc/group file.

FILES

/etc/group

SEE ALSO

chmod(1), chown(1), groups(1), id(1), logname(1), ls(1).
group(4), passwd(4) in the *Programmer's Reference for the DG/UX System*.

NAME

chgtinfo – create a temporary version of a TERMINFO entry

SYNOPSIS

```
TERMINFO=`chgtinfo modifications`
export TERMINFO
```

DESCRIPTION

One of the touted drawbacks of TERMINFO has been that one could not create a temporary modification of a TERMINFO entry. `Chgtinfo` permits the user to make such modifications.

The *modifications* are actual `terminfo`(4) source statements and are passed on to `tic`(1M) for compilation. The new TERMINFO directory tree is printed out so that it may be assigned to `$TERMINFO`. Any programs run subsequent to this assignment will make use of the modified TERMINFO entry instead of the original TERMINFO entry.

EXAMPLE

```
TERMINFO=`chgtinfo xhp, smso=E[3m, smul@,`
export TERMINFO
```

This will add the `xhp` boolean variable, change or add the `smso` string variable, and remove the `smul` string variable from the current TERMINFO entry.

FILES

/usr/tmp/tinfo*	Temporary directories and files holding modified TERMINFO entry

SEE ALSO

`sh`(1), `tic`(1M), `terminfo`(4).

NAME

chkey – change your encryption key

SYNOPSIS

chkey

DESCRIPTION

NOTE: Secure RPC using DES Authentication is an additional feature that must be purchased separately from the DG/UX ONC/NFS package. You must have this feature to use this command.

chkey prompts the user for their login password, and uses it to encrypt a new encryption key for the user to be stored in the publickey(4) database.

SEE ALSO

keylogin(1), keyserv(1M), newkey(1M), publickey(4).

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), `fcntl`(2) in the *Programmer's Reference for the DG/UX System (Volume 1)*.

NOTES

`chmod` permits you to produce useless modes so long as they are not illegal (e.g., making a text file executable). `chmod` does not check the file type to see if mandatory locking is available.

NAME

chown – change file owner

SYNOPSIS

chown [-R] [-h] *owner 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.

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.

EXAMPLES

```
$ ls -l test_file
-rw-rw-rw-   1 intern   other        349 Nov 18 13:26 test_file
$ chown wilson test_file
$ ls -l test_file
-rw-rw-rw-   1 wilson   other        349 Nov 18 13:26 test_file
$
```

The original owner of test_file was intern. After the chown command was executed, the new owner becomes wilson. Only the current owner of a file or the superuser can change the owner name.

FILES

/etc/passwd

SEE ALSO

chgrp(1), chmod(1), id(1), logname(1), ls(1).
chown(2), passwd(4) in the *Programmer's Reference for the DG/UX System*.

NAME

`clear` – clear 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.

EXAMPLES

```
clear
```

Clears your screen and moves the cursor to the top of the screen.

FILES

`/usr/lib/terminfo` terminal information data base

SEE ALSO

`tput`(1)
`terminfo`(4), `environ`(5) in the *Programmer's Reference for the DG/UX System*.

NOTE

`Clear` is a shell script that calls `tput`(1).

NAME

`cmp` – compare two files

SYNOPSIS

`cmp [ -l ] [ -s ]` *file1 file2*

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.

Options

`-l` Print the byte number (decimal) and the differing bytes (octal) for each difference.

`-s` Print nothing for differing files; return codes only.

DIAGNOSTICS

Exit code `0` is returned for identical files, `1` for different files, and `2` for an inaccessible or missing argument.

SEE ALSO

`berk_diff`(1), `comm`(1), `diff`(1).

NAME

`col` – filter reverse line-feeds

SYNOPSIS

`col [ -bfpx ]`

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 the `tbl`(1) preprocessor.

Options are:

`-b` No backspacing. If two or more characters are to appear in the same place, only the last one read will be output.

`-f` Fine-adjust half-line motions. Although `col` accepts half-line forward motions in its input, it outputs them as full-line motions unless you specify `-f`. Reverse half-line motions (and all other reverse line motions) are still ignored.

`-x` Do not convert white space to tabs on output. Normally `col` converts blank areas to tab sequences.

`-p` Output escape sequences as regular characters. Without this option, `col` will ignore any unknown escape sequences found in its input. Don't use this option unless you know the textual position of the escape sequences.

`Col` assumes that the ASCII control characters SO (\016) and SI (\017) 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 to ensure that each character is printed in the correct 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.

SEE ALSO

`nroff`(1), `tbl`(1) in the Documenter's Toolkit. *Using the Documenter's Tool Kit on the DG/UX System*, *Documenter's Tool Kit Technical Summary for the DG/UX System*.

NOTES

The input format accepted by `col` matches the output produced by `nroff` with the `-Tlp` option. The `-T37` option is not implemented in the DG/UX system.

BUGS

`Col` cannot back up more than 128 lines.

It allows at most 800 characters, including backspaces, on a 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

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 filename – 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), berk_diff(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`

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.

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.

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

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), `rm`(1).

NAME

cpd – change or view the allocation limits for a control point directory

SYNOPSIS

`cpd [ -b` *blocks* `] [ -f` *file-nodes* `]` *dirname* ...

DESCRIPTION

If no options are given, `cpd` displays the current allocation and the maximum allocation of blocks and file nodes for each control point directory named on the command line. If the `-b` or `-f` option is given, the allocation limits of the control point directory are changed as described below:

`-b` *blocks* Set the maximum block allocation to *blocks*. This is the maximum number of blocks that can be allocated to this directory and all of its descendants. Alternatively, you may specify the maximum in bytes instead of blocks. To do this, append the appropriate suffix to the `-b` option value: `b` for bytes, `k` for kilobytes (1024 bytes), `m` for megabytes (1,048,576 bytes), and `g` for gigabytes (1,073,741,824 bytes). For example, `-b 5m` sets a limit of 5 megabytes on the amount of space that can be allocated for the directory and all its descendants. The letter suffix may be upper or lower case. Note that the byte size may be rounded down by `cpd` to be a multiple of the block size.

`-f` *file-nodes*
Set the file node allocation limit to *file-nodes*. This is the maximum number of file nodes that can be allocated to this directory and all of its descendants. *file-nodes* may include a "k", "m", or "g" suffix.

In order to change the allocation limits for a CPD, the user must have write permission in the parent directory (owning the CPD is not sufficient). In the case where the CPD is the root of a file system, only the superuser can change the limits.

The last component of *dirname* may not be "." or "..". Use an absolute pathname instead.

The limits for a CPD may be resized to any value between 0 and the system maximum. Note that it is not a requirement that either allocation limit be greater than the current allocation.

To create a control point directory, use the `mkdir`(1) command.

DIAGNOSTICS

`cpd` returns a non-zero status code if any of the *dirname*s does not exist, is not a CPD, is not on a local file system, or has a last component of "." or "..". Otherwise, 0 is returned.

SEE ALSO

`ls`(1), `mkdir`(1).

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio -i[bBcdfkmrsStuvV6] [-C *bufsize*] [-E *file*] [-H *hdr*] [-I *file* [-M *message*]] [-Q *query-file*] [-R *ID*] [*pattern* ...]

cpio -o[aABcLvV] [-C *size*] [-H *hdr*] [-O *file* [-M *message*]] [-Q *query-file*]

cpio -p[adlLmuvVY] [-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 unless you supply the -B option (for a 5120-byte block size) or the -C option (for a user-specified block size). You will acheive an improvement in performance by using a block size that is larger than the default.

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 partitions.
-b	Reverse the order of the bytes within each word. (Use only with the -i option.)
-B	Input/output is to be blocked 5120 bytes to the record. The default buffer size is 512 bytes when this and the -C options are not used. If you use the larger block size, the operation takes far less time. The smaller block size is

desirable only when you are dumping data that you may need on a system whose **cpio** command requires that data be in 512-byte blocks. (You cannot use the `-B` option with the *pass* option; `-B` is meaningful only with data directed to or from a character special device, for example, `/dev/rmt/0`.)

`-c` Read or write header information in ASCII character form for portability. The output of `cpio -oc` complies with the *extended cpio* format described in *POSIX* and *XPG3*. Always use this option (or the `-H` option) when the origin and the destination machines are different types (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/0`.) Some types of tape drives, including models 6577 (QIC-150 150MB 1/4" cartridge) and 6590 (8mm 2GB helical scan) require that *bufsize* be a multiple of 512.

`-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. Always use this option or the `-c` option when the origin and the 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 Data Interchange Standard header and format
`tar` or `TAR` – `tar` header and format
`odc` – ASCII header with small device numbers (the same as `-c`).
`asc` – the new "portable" format. This format is produced by the `-c` option on some other System V.4 systems.

`-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.

`-Q` *query-file*
Specify *query-file* as the file from which to read input from the operator. Normally, `cpio` writes operator messages to stderr and reads operator responses from /dev/tty. This option allows operator input to be read from an alternate source such as fifo-special file. This is very useful when running `cpio` from `cron`(1M) since `cron` jobs have no controlling tty. For example,

```
find /foo -print | cpio -o -Q /tmp/fifo >/dev/rmt/0 2>/dev/console
```

would backup directory foo to tape /dev/rmt/0. Error messages and operator queries would be written to the console, and operator input would be read from the fifo file /tmp/fifo. Running this command from a cron job would allow you to send `cpio` output to the console without having to take control of the console for input. Operator queries from `cpio` (such as requests for the next tape) could be answered by echoing responses to /tmp/fifo.

`-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.

`-Y` Create symbolic links instead of copying files. (This option can only be used with the `-p` option).

`-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 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/0 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), ar(4).

NOTES

An archive created with the -Hasc option on a Release 4.0 system cannot be read on System V Release 3.2 systems, or earlier.

System V Releases prior to Release 4.0 do not understand symbolic links. The result of copying in a symbolic link on an older release will be a regular file that contains the pathname of the referenced file.

Path names are restricted to 256 characters for the binary (the default) and -H odc header formats. Otherwise, path names are restricted to 1024 characters.

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.

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. The environment variable `EDITOR` determines which editor is invoked with the `-e` option. The default editor is 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

/etc/cron.d	main cron directory
/var/spool/cron/crontabs	spool area
/etc/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), ed(1).

cron(1M) in the *System Manager's Reference for the DG/UX System*.

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(1) to the other user's login before making any changes to the crontab file.

NAME

crypt – encode/decode

SYNOPSIS

crypt [*password*]

where:

password A key that selects a particular transformation

DESCRIPTION

Crypt, although documented here, is not distributed outside of the United States in accordance with Federal Export regulations. International versions of the DG/UX System do not include encryption mechanisms. crypt reads from the standard input and writes on the standard output. If no *password* is given, crypt demands a key from the terminal and turns off printing while the key is being typed in. crypt encrypts and decrypts with the same key:

```
crypt password <clear >cypher

crypt password <cypher | pr
```

will print the clear.

Files encrypted by crypt are compatible with those treated by the editor ed in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; and "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, they require a lot of work.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e., to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lowercase letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the crypt command, it is potentially visible to users executing ps(1) or a derivative. The choice of keys and key security are the most vulnerable aspect of crypt.

EXAMPLES

```
$ cat a_name
Don Ho
$ crypt 24 < a_name > encrypted_name
```

Crypt is passed a key and a file that contains a name to be encrypted. crypt puts the encrypted name into the file encrypted_name.

```
$ crypt 24 < encrypted_name
Don Ho
```

Crypt decrypts the contents of the encrypted file and displays the decrypted results on the screen.

FILES

/dev/tty	For typed key

SEE ALSO

`ed`(1), `makekey`(1), `stty`(1).

NOTES

If output is piped to `nroff` and the encryption key is *not* given on the command line, `crypt` can leave terminal modes in a strange state (see `stty`(1)).
If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

NAME

csh – invoke a shell (command interpreter) having a C-like syntax

SYNOPSIS

`csh [ -bcefinstvVxX ] [` *script* `] [` *arg* `... ]`

where:

script	The pathname of a file containing a C shell script.
arg	An argument to the script.

DESCRIPTION

The `csh` command invokes a C shell, a command interpreter, which was developed at the University of California at Berkeley. The C shell is both a command line interpreter and a programming language, which allows you to compose executable shell scripts.

The primary attributes of the C shell are job control, history, and aliasing. The C shell also has predefined variables to prevent accidental file overwrites and log offs, and to disable filename expansion.

Through the C shell, you can enable editread, an optional command line editor. The editread history facility is similar to the C shell's, but its implementation and use are different. Refer to `editread`(5) for more information.

This man page covers the following `csh` topics:

— Command Line Options

— Initialization and Termination

— Command Line Words

— Quoting Special Characters

— Predefined Variables

— Built-in Commands

— Filename Completion

— History Substitution

— Aliasing

— Job Control

— Pipes, Sequential Command Processes, and Command Groups

— Input/Output Redirection

— Variables and Variable Substitution

— Command Substitution

— File Substitution

— Expressions and Operators

— Signal Handling and Status Reporting

— Parsing Order and Execution

— International Features

Command Line Options

`-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`	Reads commands from the first argument (a filename), which must be present. Any remaining arguments on the command line are placed in the predefined shell variable `argv`, which stores the argument list.
`-e`	Exits if any invoked command terminates abnormally or yields a non-zero exit status.
`-f`	Fast start-up; does not search for or execute commands from the `.cshrc` file or the `.login` file (if in a login shell), thus reducing shell start-up time.
`-i`	Forced interactive; prompts for input from the terminal, even if standard input does not appear to be a terminal (for example, a special character device). If a shell's I/O devices are terminals, interactive operation is assumed without having to set this option.
`-n`	Parses but does not execute commands. This may aid in syntactic checking of shell scripts.
`-s`	Takes commands from standard input.
`-t`	Takes one line of input to read and execute. You can use a backslash (\) to escape each terminating new-line so that input can continue on the next line.
`-v`	Sets the `verbose` predefined variable, which echoes a command's input after history substitution but before command execution. See the section on *Predefined Variables* for more about the `verbose` predefined variable.
`-V`	Sets the `verbose` variable before `.cshrc` is executed.
`-x`	Sets the `echo` variable so that commands are echoed immediately before execution.
`-X`	Sets the `echo` variable before `.cshrc` is executed.

Initialization and Termination

When you log in to the system, a shell executes the commands in these files:

`/etc/login.csh`	Executes only during login to the system; is maintained by the system administrator. Typically contains environment and local shell variables.
`.cshrc`	If it exists in your home directory, executes at login and each time a shell executes a script or creates a subshell. Typically contains aliases and local shell variables.
`.history`	If it exists in your home directory, executes at login, reading in a list of saved history events into the current history list.
`.login`	If it exists in your home directory, executes following execution of `.cshrc` and only during initial login. Typically contains environment variables, the umask settings (default permissions assigned to user-created files), `stty` settings, and other commands to be executed at login only.
`.logout`	If it exists in your home directory, executes at logout only.

After you have successfully logged in, an interactive shell will usually begin reading commands from the terminal, prompting with *hostname*`%` (the default) for the ordinary user; *hostname*`#` for the superuser.

Command Line Words

The C shell splits input lines into *words* at blank(s), tab(s), and new-line(s). Regardless of surrounding space, the following special characters are also recognized as words:

space tab new-line	Command argument separator.
$	Variable identifier.
* [ ] ? { } ~	Filename expansion characters.
< > & !	Redirection symbols.
! ^	History characters.
|	Pipe.
;	Command separator.
( )	Command group.
\ ' "	Quoting.
`	Command substitution.
&	Background execution.

These words must be quoted (escaped) with a backslash (\) to inhibit their interpretation as special characters.

Quoting Special Characters

In addition to the backslash (\), you can also use the following characters to inhibit the interpretation of the special characters listed in the previous section:

\`*command*\`	Command substitution; see the section on *Command Substitution*.
'*string*'	String characters taken literally.
"*string*"	Allows command substitution and variable substitution.

Predefined Variables

The predefined variables in this section have special meaning to the shell. Of these, the shell automatically sets `argv`, `cwd`, `home`, `path`, `prompt`, `shell` and `status`. Except for `cwd` and `status`, the shell sets these variables only at login.

You can set a variable as an *environment variable* (variable is exported to subshells) or a *local shell variable* (variable is known only to the current shell) using the `set` and `setenv` commands (covered in the section on *Built-in Commands*). By convention, environment variables are set in uppercase characters, and local shell variables are set in lowercase characters. (See the section on *Variables and Variable Substitution* for more information on how to set variables.) Display local variables with the built-in command `set`. Display environment variables with the `printenv`(1) command.

The shell copies the environment variable `TERM` into `term`, `HOME` into `home`, and `PATH` into `path` and copies these back into the environment when a variable's value changes.

Except for those predefined variables that the shell sets automatically, you must set explicitly all other predefined variables. You set a variable by either declaring it or by assigning it a value, whichever is appropriate. The predefined variables follow:

`argv`	Contains the command line arguments supplied to the current shell. This variable contains the values for the positional parameters, referenced as `$0`, `$1`, `$2`, and so on, through `$9`. With `argv`, you can reference the first item on the command line with `argv[0]`, the second item with `argv[1]`, and so on, through `argv[9]`; you can reference all arguments with `argv[*]`, and the number of arguments with `#argv`.
`cdpath`	Change directory path; contains a list of alternate directory pathnames used by commands (such as `cd`, `chdir`, and `popd`) when searching for subdirectories.
`cshscript`	Causes shell scripts that start with a pound sign (#) to be interpreted by the C shell instead of the Bourne shell. If this option is not set, only scripts that start with `#! /bin/csh` will be interpreted by the C shell. Setting this option provides compatibility with other implementations of the C shell, but Bourne shell scripts that start with a pound sign (except for `#! /bin/sh`) will probably break.
`cwd`	Contains the full pathname of the current working directory.
`echo`	Causes each command and its arguments to be echoed just before it is executed. This is set when the `-x` command line option is given (see the `-x` option in the previous section on *Command Line Options*). This option can also be set with the `set echo` command.
`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 and the ESC character have special significance when typed in at the end of a terminal input line:
`histchars`	Identifies a two-character string used as history substitution metacharacters. The first character replaces the default history substitution character, !. The second character replaces the quick substitution character, ^.
`history`	Specifies the number of history events (commands issued from the command line) to be saved in the history list. A large number of events saved in the history list can exceed available shell memory. If the `history` variable is not set to a specific value, only the last executed command is saved in the history list.
`home`	Specifies the user's home directory. The filename expansion of ~ refers to the value of the `home` variable.
`ignoreeof`	If set, makes the shell ignore the end-of-file signal from terminal input devices. Setting `ignoreeof` prevents accidental logouts issued with `Ctrl-D`.
`mail`	Defines file location(s) where the shell checks for mail and the interval at which you are notified of the arrival of new mail. The variable is specified in this form:

`set mail =` [*n*] *mailfile-path* [*mailfile-path*]

The optional *n* value specifies the mail-checking interval in seconds and is used to override the default, 600 seconds.

If new mail arrives, you are alerted with the message, "`You have new mail`." If a command is being executed during the arrival of mail, the message is postponed until the prompt returns to the the terminal screen. If multiple mailfile paths are specified, you are alerted with the message, "`New mail in` *mailfile-path*."

`nobell`
: Suppress the bell during filename completion when asking the C shell to extend an ambiguous filename.

`noclobber`
: Restricts output redirection to ensure that files are not accidentally destroyed or "clobbered" (described in the section on *Input/Output Redirection*). It prevents you from overwriting an existing file when using the redirection symbol (>). Also, it prevents the creation of a new file when you attempt to append output to a nonexistent file when using the append output symbol (>>). Instead, error messages are displayed to alert you to the problem.

`noglob`
: If set, inhibits global filename expansion. Filename expansion metacharacters * ? [] ~ {} are not recognized and are treated as literal characters instead. Setting `noglob` in shell scripts is useful after filenames have been expanded or when filename expansion is not desired. (Refer to the section on *Filename Substitution* for more information on filename expansion and metacharacters.)

`nonomatch`
: If set, inhibits the display of error messages if commands containing filename expansion fail to locate a matching pattern. Malformed patterns, however, are considered errors for which error messages are displayed. For example, the command, `echo [`, returns an error.

`notify`
: If set, the shell notifies you asynchronously of a job's completion. The default is to issue a job's completion message before the prompt returns to the terminal screen.

`path`
: Specifies a list of directories that is searched for an executable command. If this variable is not set, then only full pathnames will execute. The default search path is (`. /bin /usr/bin`); however, the default varies from system to system. A null word specifies the current directory. For the superuser the default search path is (`/etc /bin /usr/bin`).

A shell command with neither the `-c` nor the `-t` options will normally hash the contents of the directories in the `path` variable after reading `.cshrc`, and each time the `path` variable is reset. If new commands are added to these directories while the shell is active, you may need to use the `rehash` command to update the command list.

`prompt`
: Defines the string used by an interactive shell as a prompt for your input. If a ! appears in the prompt string, it will be replaced by the current history event number (assigned to each command issued from the command line) unless a preceding \ is given. (Refer to the section on *History Substitution* for more information on the current event.) The default is % for the normal user, or # for the superuser.

`savehist`	Specifies the number of events in the history list saved in the `.history` file in your home directory when you log out. During shell startup, the shell reads the contents of `.history` into the history list. A large value for `savehist` slows down the shell during startup.
`shell`	Specifies the file in which the shell resides. The default is `/bin/csh`.
`status`	Contains the status returned by the last command. If a command terminates abnormally, then 0200 is added to the status. Built-in commands (those that do not execute as child processes) that fail will return exit status 1; all other built-in commands set status 0. (See the `status` command under *Built-in Commands*.)
`time`	Controls automatic timing of commands. If set, then any command requiring more than the specified number of CPU seconds will print a line when it terminates, giving user, system, and real times and a utilization percentage (the ratio of user plus system times to real time). (See the `time` command under *Built-in Commands*.)
`verbose`	Prints the words of each command after history substitution. This is set by the `-v` command line option.

Built-in Commands

C shell built-in commands are executed within the shell. If a built-in command is any component of a pipeline or a command group (except the last one), then it is executed in a subshell.

`alias` *name definition*

An alias is an alternate name you can assign to an existing DG/UX system command. This form of the `alias` command assigns the specified *definition* to the alias *name*. (See the section on *Aliasing* for more information.)

`bg`
`bg %`*job* ...

The first form (without an argument) moves the last suspended job to the background for continued execution. The second form puts the specified job into the background for continued execution. (Refer to the section on *Job Control* for information.)

`break`

Interrupts a `foreach` or `while` loop. `break` executes the remaining commands on the current line before it transfers control to the instruction following the `end` of the loop.

`breaksw`

Breaks from a `switch`, resuming after the `endsw`.

`case` *label*:

Specifies a label in a `switch` statement (discussed in a later paragraph about `switch`).

`cd`
`chdir`
`cd` *dir*
`chdir` *dir*

The first and second forms change from the C shell's working directory to the user's home directory. The third and fourth forms change the C shell's working directory to a directory named *dir*. If *dir* is not found as a subdirectory of the current directory (and does not begin with `/`, `./`, or `../`), then each component of the predefined variable `cdpath` is checked for a subdirectory named *dir*. If *dir* is a shell variable whose value begins with `/`, then the shell changes to this directory. The second and fourth forms (the `chdir` expression) are the same as the first and third forms, respectively.

`continue`
: Interrupts a `while` or `foreach` loop. `continue` executes the remaining commands on the line before it transfers control to the `end` statement, which then sends control back to the top of the loop.

`default:`
: Labels the default case in a `switch` statement. The default should follow all `case` labels.

`dirs`
: Prints the directory stack. The first directory in the stack is the current directory. With the -l argument, produce an unabbreviated printout; use of the ˜ notation is suppressed. (See also `pushd` and `popd` later in this section.)

`echo` *wordlist*
`echo -n` *wordlist*
: The specified words are written to the shell's standard output, separated by spaces, and terminated with a new-line unless the `-n` option is specified, in which case the new-line is suppressed.

`else`
`end`
`endif`
`endsw`
: See the upcoming descriptions of `foreach`, `if`, `switch`, and `while` .

`eval` *arg ...*
: The arguments are read as input to the shell and the resulting command(s) executed. This is used usually to execute commands generated as the result of variable or command substitution (see the sections on *Variables and Variable Substitution* and *Command Substitution*), because parsing occurs before these substitutions.

`exec` *command*
: Executes *command* in place of the current shell.

`exit`
`exit` *(expr)*
: Terminates the shell with either the value of the `status` variable (first form) or with the value of the specified *expr* (second form).

`fg`
`fg %`*job ...*
: Brings the current job (first form) or a specified *job* (second form) into the foreground for execution.

`foreach` *name* (*wordlist*)
  ...
`end`
: The variable *name* is set successively to each member of *wordlist*, and the sequence of commands between the `foreach` command and the matching `end` command are executed. (Both `foreach` and `end` must appear on separate

lines.)

`glob` *wordlist*
: Performs filename expansion on a *wordlist*. The `glob` command performs similarly to `echo` but no \ escapes are recognized. Words are delimited by null characters in the output.

`goto` *label*
: Unconditionally transfers control to a routine located in another part of the script which is identified with the specified *label*. A colon (:) follows the *label* to signify the contents of the routine. Program execution continues after the specified label.

`hashstat`
: Prints a statistics line indicating the internal hash table's effectiveness at locating commands (and avoiding execution of the `exec` command). Such an execution 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` *n*
: History enables you to recall and re-execute previously issued commands that are saved in a list. This form of the command lists *n* most recent items from the history list. (See the section on *History Substitution* for more information.)

`if` (*expr*) *command*
: If the specified expression evaluates to true, then the single *command* with arguments is executed. *Command* must be simple; it cannot be a pipeline, a command list (separated by semicolons), or a command group (surrounded by parentheses). Note that I/O redirection occurs even if *expr* is false and the command is not executed (this is a bug). (See the later section on *Pipes, Sequential Command Processes, and Command Groups*.)

`if` (*expr*) `then`
  ...
`else if` (*expr2*) `then`
  ...
`else`
  ...
`endif`
: If the specified *expr* is true, then the commands following `then` (up to the first `else if`) are executed; if *expr2* is true, then the commands following the second `then` (up to the second `else`) are executed, and so on. Any number of `else if` pairs can be used, but only one `else` (optional) and one `endif` (required) can be used. The words `else` and `endif` must appear at the beginning of input lines; the `if` must appear alone on its input line or after an `else`.

`jobs`
: Enables you to list the active jobs that you can control through the job control facility. (Refer to the section on *Job Control* for more information.)

`kill %`*job*
: Terminates an active job that is identified by a specific number preceded by a percent sign (%). (See the section on *Job Control* for more information.)

`limit`
`limit` *resource*
`limit` *resource maximum-use*

`limit -h` *resource maximum-use*

Limits resource consumption for each process and each of its forked processes to no more than *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

`-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 the following:

cputime	Maximum number of CPU-seconds to be used by each process.
filesize	Largest single file that can be created.
datasize	Maximum growth of the data and stack for the process beyond the end of text.
stacksize	Maximum size of the stack for the process.
coredumpsize	Size of the largest core dump file that will be created.
memoryuse	Maximum size that a process' resident set size may grow to.
descriptors	Maximum number of open files that a process may have at one time.

Maximum-use can be a number (floating point or integer) followed by a scale factor.

*n*k	(kilobytes); default for all limits other than *cputime* and *descriptors*.
*n*m	(megabytes); an alternative to kilobytes for all limits other than *cputime* and *descriptors*.
n	Default *cputime* limit in seconds.
*n*m	*n* minutes for *cputime*.
*n*h	*n* hours for *cputime*.
mm:ss	Minutes and seconds for *cputime*.

`login`

Terminates the current login shell, replacing it with an instance of `/bin/login`. This method of logging off is used for compatibility with the Bourne shell.

`logout`

Terminates a login shell, which is especially useful if `ignoreeof` is set.

`newgrp`

Changes the group identification of the caller; for details, see the `newgrp`(1) man page. `newgrp` executes a new shell so that the previous shell state is lost.

`nice`
`nice +` *number*
`nice` *command*
`nice +` *number command*

`nice` – *number command*

Executes a process at a lower priority (or a higher priority for superusers only), which reduces the demand that the process makes on the system. The "nice" number is the factor (4 by default) that is added to (or subtracted from) your job's priority. The higher the *nice* number, the lower the priority of a process. The `nice` priority values range from 0 to 39. The default priority is 20.

The first form sets the `nice` number for the current shell to 4 (the default), which means that the `nice` value would be 24. The second form sets the priority to 20 + *n*. The third form runs *command* at the default `nice` value. The fourth form runs *command* at a priority of `20 +` *n*. The final form (for the superuser only) runs *command* at a priority of `20 –` *number*. The maximum changes to the `nice` value are: `nice + 19` and `nice – 20`.

This `nice` command is not the same as the one documented in the `nice`(1) manual page. The `nice`(1) manual page documents the program `/usr/bin/nice`.

`nohup`
`nohup` *command*

The first form can be used in shell scripts to ignore hangups for the remainder of the script. The second form causes the specified *command* to run with hangups ignored. *Command* is always run in a subshell. All processes run in the background (commands appended with `&`) are effectively run without hangups.

`notify`
`notify %`*job ...*

If set, notifies you immediately when the status of the current job (first form) or a specified job (second form) changes; normally, notification is presented after a process has completed just before the prompt reappears on the screen. (See the `notify` variable under *Predefined Variables*.)

`onintr`
`onintr –`
`onintr` *label*

Controls the action of the shell on interrupts. The first form restores the default action of the shell, which is to terminate a shell script or to return to the terminal command input level. The second form, `onintr –`, causes all interrupts to be ignored. The final form causes the shell to execute a `goto` label when an interrupt is received or a child process terminates because it was interrupted.

If the shell is running in the background (detached) and interrupts are being ignored, no form of `onintr` has meaning. The shell and all invoked commands continue to ignore interrupts.

`popd`
`popd +`*n*

Pops the directory stack, returning to the new top directory. With an argument +*n*, the *n*th entry in the stack is discarded. The elements of the directory stack are numbered from 0, starting at the top.

`pushd`
`pushd` *name*
`pushd +`*n*

With no arguments, `pushd` (first form) exchanges the top two elements of the directory stack and changes the current directory to the top directory. Given a *name* argument, `pushd` (second form) changes to the new directory and pushes the old current working directory onto the directory stack. With a numeric argument, `pushd` +*n* (final form) rotates the *n* th argument of the directory stack to the top and changes to it. The members of the directory stack are numbered from the top, starting at 0.

`rehash`
: Recomputes the internal hash table of the contents of the directories in the `path` variable to account for new entries added while logged in. This action is necessary only if you add commands (or scripts) to the directories in the path.

`repeat` *count command*
: Repeats *command count* times. I/O redirections occur exactly once, even if *count* is 0.

`set` *variable* = *value*
: Assigns a *value* to shell *variable*. (Refer to the section on *Variables and Variable Substitution* for more information.)

`setenv` *variable value*
: Assigns a *value* to environment *variable*. (Refer to the section on *Variables and Variable Substitution* for more information.) Display your environment variables with the `printenv`(1) command.

`shift`
`shift` *variable*
: In the first form, the components of `argv` are shifted to the left, discarding `argv[1]`. It is an error for `argv` to be set to null or to have no words as a value. The second form performs the same function on the specified variable.

`source` *file*
`source -h` *file*
: The first form reads commands from *file*. These commands may be nested; if they are nested too deeply, however, the shell can run out of file descriptors. An error in a source at any level terminates all nested source commands. Commands read from a file will not be added explicitly to the history list. The `-h` option, however, (second form) will add the commands to the history list without being executed.

`stop %`*job ...*
: Stops the specified job that is executing in the background. (See the section on *Job Control* for more information.)

`suspend`
: Interrupts the shell temporarily (until you execute a command to handle the suspended shell), much as if it had been sent a stop signal with `<Ctrl-Z>`. This is most often used to stop shells started by `su`(1).

`switch` (*string*)
`case` *string1*:
commands
`breaksw`
`case` *string2*:
commands
`breaksw`
`default`:
commands

`breaksw`
`endsw`

Each case label (such as *string1* and *string2*) is successively matched against the specified *string*. The file metacharacters *, ? and [...] may be used in the case labels, which are resolved to a filename. If none of the labels match before a default label is found, then execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command `breaksw` continues execution after the `endsw`. If no label matches and there is no default, execution continues after `endsw`.

`time`
`time` *command*

With no argument (first form), prints a summary of time and system resources used by the current shell and subprocesses. If arguments are given (second form), the specified simple *command* is timed and a summary of time and system resources used is printed. The information is printed in seven fields. An explanation of a "zero consumption" case with each field description follows:

0.0u	User time, in seconds.
0.0s	System time, in seconds.
0:00	Real time, in minutes and seconds.
0%	Rough approximation of the percentage of CPU cycles used during real time, which is calculated by adding system and user times and dividing the sum by elapsed real time.
0+0k	Amount of shared and unshared memory-time, in kilobyte-seconds, each separated by +.
0+0io	Number of blocks input and output, each separated by +.
0pf+0w	Number of page faults (pf) and number of times the process was swapped out to disk (w).

If necessary, an additional shell is created to print the time statistic when the command completes. The `time` variable, discussed in the section on *Predefined Variables*, can be set to a threshold; thereafter, time use information (system, user, real) is printed whenever any program or command exceeds that threshold.

`umask`
`umask` *value*

Displays the three-digit octal mask value (first form) that identifies the access mode created for files and directories. By default, all files are created with a `umask` value of 666; for directories, 777. The first digit identifies the owner's permissions; the middle digit, the group's permissions; the last digit, permissions for all other users.

The second form is the octal mask value that the owner sets. Each specified digit removes a specific permission; a value of 1 removes execute permission, 2 removes write permission, and 4 turns off read permission. As an example, an owner could deny the permission for group and other with a umask of 022, which yields a umask of 644 (owner has read and write permission, group has only read permission, and other has only read permission). Values are additive; for example 6 turns off read and write permissions.

Note that `umask 000` is effectively `umask 111`; the shell creates files with a default permission of 666, which means that no one (not even the owner) has execute permission for the file. Use `chmod(1)` to add execute permission.

`unalias` *pattern*
: Deletes specified alias with a matching pattern. You can delete all aliases using the filename metacharacter `*`. For example, all aliases are removed with `unalias *`. (See the section on *Aliasing* for more information.)

`unhash`
: Disables the internal hash table.

`unlimit`
`unlimit` *resource*
`unlimit -h` *resource*
: If no *resource* is specified (first form), then all *resource* limitations are removed. (Refer to the `limit` command for information on *resource* names.) Removes the limitation on *resource* (second form).

  `-h` Remove corresponding hard limits. Only the privileged user may do this.

`unset` *name*
: Deletes specified variable.

`unsetenv` *name*
: Deletes specified environment variable. (Refer to the `setenv` command in this section and the `printenv(1)` command for information on setting and displaying environment variables.)

`wait`
: Delays some action until all background jobs are completed. If the shell is interactive, an interrupt can disrupt the wait, at which time the shell lists all jobs and associated numbers that are in the background, suspended, or stopped.

`while` (*expr*)
...
`end`
: While the specified *expr* is true (evaluates to non-zero), the commands between the `while` and the matching `end` are executed. The `while` *expr* and `end` must appear alone on a line each. (See `continue` and `break` for information on interrupting a loop.)

`%`*job*
`%`*job* `&`
: Brings the specified job into the foreground (first form); continues the specified *job* in the background (second form). (See the section on *Job Control* for more information.)

`@` *variable = expr*
: This form sets the specified *variable* equal to the value of *expr*. (See the section on *Variables and Variable Substitution* for more information.)

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.

History Substitution

History substitution allows you to recall, re-execute, and edit previously entered commands. With this facility, you can repeat commands, repeat arguments of a previous command in the current command, or edit a previous command (for example, to fix spelling mistakes). Command lines, known as *history events*, are saved in a history list, the length of which is controlled by the predefined `history` variable (refer to the section on *Predefined Variables*).

History substitutions begin with the character `!` and may begin anywhere in the input stream, as long as they do not nest. You can change the value to another character and store it in predefined variable `histchars`. The `!` can be escaped with a \ to prevent its special meaning; the `!` is not interpreted as a special character if it is followed by a blank, tab, new-line, =, or (. History substitutions can also be performed using the ^ character, discussed in the subsection on *Event Modifiers*. Any input line that contains history substitution is expanded and echoed on the terminal before it is executed.

`set history =` *n*
`history`
`history` *n*
`history -r` *n*
`history -h` *n*

The first form sets the number of history events to be contained in the history list. The second form prints a history list. The third form prints only the *n* most recent events. The fourth form reverses the order of the history list so that the more recent events are at the top of the list; the older events are at the bottom. The final form prints the history list without leading numbers, which produces files suitable for using the `-h` option to the `source` command.

An example of setting the history list length follows:

```
set history = 5
```

Regardless of the number of command lines entered, the five most recent commands are saved in the list. Command lines are numbered sequentially from 1. For example, event 6 would be maintained as the most recent event, and event 1 would be deleted, thus maintaining a constant length of 5.

The second form prints a history list. An example follows:

```
 8  ls -l
 9  write michael
10  vi write.c
11  cat oldwrite.c
12  diff write.c write.d
```

The commands are shown with their event numbers, which can be used in the prompt by preceding the prompt string with a `!`. An example follows:

```
% set prompt = '\! %'
```

Event Designators

An event designator is used to invoke an event from a history list.

!	Designates a history substitution, except when followed by a space, tab, new-line, =, or (.
!!	Recalls the previous event just executed.
!*n*	Recalls event *n* from the history list.
!-*n*	Recalls event −*n* relative to the current event.
! *string*	Recalls the most recent history event beginning with *string*.
!?*string*?	Recalls the most recent history event containing the embedded *string*.
!{*string1*}*string2*	Recalls the most recent history event matching *string1* and appends *string2* to it.

The following examples of event designators are based on the preceding history list given in this section.

```
13% !!
diff write.c write.d
14% !9
write michael
15% !v
vi write.c
16% !?old?
cat oldwrite.c
17% !{d} > save.file
diff write.c write.d > save.file
```

Word Designators

To select words from an event, follow the event designator by a colon (:) and a designator for the desired words. The words of an input line are numbered from 0, the first word being 0 (the command), the second word (first argument) being 1, and so on. The basic word designators are:

0	First word, which is always the command.
n	*n*th argument
^	Second word, which is the first argument.
$	Last argument.
%	Word matched by (immediately preceding) ?*string*? search.
x−*y*	Range of words.
−*y*	Abbreviates 0−*y*.
*	Abbreviates ^-$, or nothing if only one word in event.
*x**	Abbreviates *x*−$.
x−	Like *x**, but without word $.

The : separating the event designator from the word designator can be omitted if the word designator begins with a ^, $, *, − or %.

Examples of word designators are based on the following history event:

12% cd test ; ls | grep '\.h$'

Command	**Produces**
% `!12:0`	cd
% `!12:3`	ls
% `!?rep?:% Bob names`	grep Bob names
% `!12:0-1`	cd test
% `!12:3*`	ls | grep '\.h$'
% `echo !12:^ !$`	echo test '\.h$'

Word Modifiers

A sequence of modifiers can follow the optional word designator. Precede each with a :. The following modifiers are defined:

`h`	(head); removes a trailing pathname component, leaving the head.
`r`	(root); removes a trailing filename extension, leaving the root name.
`e`	(extension); removes all of the filename except the extension.
`s/`*str1*`/`*str2*`/`	(substitute); substitutes search pattern *str1* with replacement pattern *str2*.
`t`	(tail); removes all leading pathname components, leaving the tail.
`&`	(repeat); repeats the previous substitution.
`g`	(global); applies the change to the first match in each word on the line; precedes other word modifiers; for example, `g&`.
`p`	(print); prints the new command but does not execute it.
`q`	(quote); quotes the substituted words (no further substitutions).
`x`	Similar to `q`, but breaks into words at spaces, tabs and new-lines.

For substitutions, unless preceded by a `g`, only the first occurrence of the matched word *str1* is modified. An error results if no word is matched.

The search string, signified by *str2*, is expressed using a literal string; regular expressions cannot be used. Any character can replace `/` as a delimiter. A \ can be used to quote the substitution delimiters separating *str1* and *str2* to escape their meanings. The `&` character, which stores the value contained in *str1* is a legal value that can be used in *str2*. The `&` construct modifies rather than replaces *str2*. The `&` can be quoted with a \ to preserve its literal meaning. A null *str1* uses the previous string from *str2* or from a contextual scan string *str* in `!?`*str*`?`. The trailing delimiter `/` in a substitution and the trailing `?` in a contextual scan may be omitted if a new-line follows immediately.

The sequence ^*str1*^*str2*^ can be used as a shortcut for `!!:s/str1/str2/`.

Examples of word modifiers are given based on the following event:

```
4% ls /usr/della/test.1
```

Command	Prints and Executes
`% ^della^eunice`	`ls /usr/eunice/test.1`
`% !4:s/test.1/quiz`	`ls /usr/della/quiz`
`% !4:h`	`ls /usr/della`
`% !4:1:r`	`/usr/della/test`
`% !4:s/test.1/re&/`	`ls /usr/della/retest.1`
`% !4:p`	`ls /usr/della/test.1` (prints but doesn't execute)

Aliasing

You can assign alternate names to existing DG/UX system commands with the alias facility. Specifically, you can rename a command, supply default arguments to a command, or construct new commands from existing ones.

`alias`
`alias` *name*
`alias` *name definition*
`unalias` *pattern*

The first form of the command displays the current list of aliases. The second form lists the corresponding definition for the specified alias. The third form assigns a simple definition to the alias. The final form deletes an alias name matching the specified pattern.

Aliases can also take user-supplied arguments, which require the use of the history facility. An example follows:

```
% alias man 'man \!* | more'
```

The alias `man` is assigned a definition wherein the `man` command and a user-supplied argument, signified by `!*`, are piped through the `more` command. The single quotation marks enclose the *definition* to prevent shell expansion. Also, the \ escapes the `!` to prevent it from being interpreted as you define the alias.

When executing the `man` alias, you can enter an argument, which is placed automatically in the history list and is then retrieved and substituted into the `alias` command. The `!*` expression refers to the previous history event (`!`) and the first through final arguments in that event (`*`).

Job Control

The C shell associates a numbered *job* with each executed command line to keep track of all commands in the background and all commands suspended temporarily (with <`Ctrl-Z`>). In addition to starting a command in the background, with the shell job control facility, you can switch a job's processing between three states: foreground, background, and suspension. The form of a job started asynchronously with `&` follows:

% *command arg* [*arg2*] ...`&`
[*n*] *pid-number*

For example, if the *pid-number* is `[1]1234`, the number, 1, surrounded by square brackets would be the job number, which has one (top-level) process, and 1234 would be the process identifier.

Listing Jobs

`jobs`
`jobs -l`

The first form lists the active jobs. The `-l` option (second form) lists PIDs, the job numbers, corresponding command lines, and status (running or stopped). An example follows:

```
% jobs
[1] - Stopped   man jobs | more
[2] + Stopped   cat large.out
[3]   Running   /usr/bin/lp -w -s -t report &
```

The shell maintains a list of the current and previous jobs. The current job is marked with a + and the previous job with a -.

Manipulating Jobs

There are several ways to manipulate jobs:

`%`*n*	Refer to job number *n*.
`<Ctrl-Z>`	Suspend foreground job.
`<Ctrl-Y>`	Suspend foreground job when interactive read is attempted.
`bg`	Put job in background.
`fg`	Put job in foreground.
`kill`	Terminate job.
`stop`	Suspend background job.

`%  %+  %%`
`%-`
`%` *n*
`%?`*string*

Three ways of referring to the current job are specified in the first line. For example, the command `fg %` puts the currently stopped job in the foreground.

The second form (second line) refers to the previous job. The command `fg %-` puts the first previous job in the foreground.

The third form (third line) refers to an absolute job number. The command `fg %3` puts the third job in the foreground.

The fourth form (fourth line) specifies an unambiguous *string* occurring at the beginning of the line. The command `fg %?/usr` would put in the foreground the job containing the string "`/usr`" at the beginning of the line, which is the third job. To manipulate a stopped job, you specify the current job with a + command preceded by a percent sign (`%`). You specify the previous job with the - command preceded by a `%`. After the current job (indicated by +) completes or moves to the foreground, the previous job (indicated by -) becomes the current job and assumes a + status. If there were a third stopped job, it would become the previous job and assume a - status.

`<Ctrl-Z>`

Suspends temporarily a foreground job, which sends a STOP signal to the executing job. The C shell should indicate that the job has been stopped by displaying a `Stopped` message followed by another prompt. A `<Ctrl-Z>`

takes effect immediately and behaves like an interrupt; pending output and unread input are discarded when typed in.

`<Ctrl-Y>`
: The `<Ctrl-Y>` command does not generate a STOP signal until the executing job attempts an interactive read from the terminal. Thus, you can issue the `<Ctrl-Y>` command during such a job's execution and the job will continue to execute until an interactive read from the terminal is attempted. A background job stops if it tries to read from the terminal. Conversely, however, a background job normally is allowed to produce output, but this can be disabled with the command `stty tostop`.

`bg`
`bg %` *job...* The first form prints a list of jobs running in the background. The second form puts the currently stopped job or a specified job into the background for continued execution.

`fg`
`fg %` *job...* Brings the current job (first form) or a specified *job* (second form) into the foreground.

`kill` *-sig* `%`*job ...*
`kill` *pid*
`kill` *-sig pid ...*
`kill -l`
: Sends either the TERM (terminate) signal or the signal (*sig*) to either a specific process (*pid*) or a specific *job* number. Signals are given either by number or name (as given in `/usr/include/sys/signal.h,` stripped of the prefix `SIG`).

The `kill -l` command lists the signal names. Using `kill` alone does not send a signal to the current job. If the `kill` command sends either a TERM (terminate) or HUP (hangup) signal to a job or process, then it also sends a CONT (continue) signal.

`stop`
`stop %` *job*
: Stops the current job (first form) or the specified *job* (second form) that is executing in the background. Using `stop 0` in a login shell (the one that you log in to) will hang your terminal. Also, you must be using the Berkeley line discipline driver (`berk_stty`) for `stop` and `suspend` signals to be handled properly.

Pipes, Sequential Command Processes, and Command Groups

command | *command ...*
command ; *command* ; *command* . . .
(*command group*) . . .
: The first form uses a *pipe* symbol (|); the second uses a set of sequential commands (;); the third signifies a command group (( )).

The *pipe* symbol is used for connecting a series of simple commands to form a *pipeline*. The output of each command in a pipeline is connected to the input of the next command. To execute a sequence of pipelines without immediately waiting for the sequence to terminate, follow it with the `&` symbol, which executes the job in the background.

An example of a pipeline follows (note that the quote marks are back quotes):

```
% echo There are `who | wc -l` users on the system today.
```

The second form shows a sequence of commands separated by semicolons (`;`), which cause each command to be performed unconditionally from left to right in a sequential manner. Commands can also be separated by `||` or `&&`, which represent conditional execution. The expression on the right of one of these symbols is performed if the left expression (signified by `||`) is false (failure) or if the left expression (signified by `&&`) is true (success).

An example of conditional execution follows:

```
% grep "Using" filea && echo "I found it."
```

If an instance of "Using" is found in `filea`, the expression would be considered true (success) and the expression to the right of `&&` would be performed.

The third form illustrates a command group, which can be composed of a pipeline or command sequence, enclosed in parentheses to form a part of a larger pipeline or command sequence. A command group processes a set of commands in a subshell, establishing an environment separate from its parent, or puts a group of commands (processed sequentially) in the background.

Examples of command grouping follow:

```
% ( cat test1 ; cat test2 ) > bigtest.out
% ( grep micro *.me > micro.out ; lp micro.out ) &
```

Input/Output Redirection

The following special characters are used to redirect standard input, standard output, or standard error from a command to a file.

< *filename*
: Opens *filename* as standard input.

<< *word*
: Reads the shell's input up to a line that is identical to *word.* If *word* contains a special character (see section on *Command Line Words*), it will be interpreted unless escaped with a backslash (\). Commands that are substituted have all blanks, tabs, and new-lines preserved, except for the final new-line, which is dropped. The resultant text is placed in a temporary file that is given to the command as standard input.

> *filename* >& *filename* >! *filename* >&! *filename*
: Angle bracket (>) signifies a redirection of standard output to *filename.* If *filename* does not exist, then it is created. If the file exists, it is overwritten; its previous contents are lost (first form).

The second form redirects standard output and standard error (diagnostic output) to *filename.* The terminal is standard error by default. If *filename* exists, it is overwritten; its previous contents are lost.

If the `noclobber` variable is set when using either the first or second forms, it prevents the shell from redirecting output to an existing file and issues an error message instead. setting the `noclobber` variable prevents accidental file overwriting of files that already exist. Note that `noclobber` permits redirection to terminals and `/dev/null`.

Alternatively, it is possible to override the `noclobber` setting with the `!` symbol. The third form, `>!`, allows an existing file to be overwritten. The final form, `>&!`, overwrites an existing file with standard output and standard error.

`>>` *filename* `>>&` *filename* `>>!` *filename* `>>&!` *filename*

Appends standard output to a file (first form). It is similar to the `>` notation but appends to, rather than creates, a file. If the file does not already exist, it is created.

The second form appends standard output and standard error (diagnostic output) to *filename*.

If the `noclobber` variable is set when using either the first or second forms, and if a file does not already exist, an error message is issued and no output is appended to a file.

Alternatively, it is possible to override the `noclobber` setting with the `!` symbol. The third form, `>>!`, creates a file if one does not already exist. Likewise, the final form, `>>&!`, creates a file if one does not already exist.

Variables and Variable Substitution

There are two types of variables that the C shell maintains: predefined and user-defined. The C shell automatically sets some predefined variables; others you can set (refer to the previous section on *Predefined Variables* for more information). User-defined variables can be restricted to the current shell (*local variables*) or exported to the environment (*environment variables*). Also, user variables can be defined as string or numeric. All variables have a *name* and a corresponding *value* of one or more words. A user-defined variable name can consist of as many as 251 characters (alphabetic and numeric and the underscore character).

A reference to the value of a variable begins with `$`, which is a signal to the shell to interpret the dollar sign and the adjacent expression as a variable. The shell then expands the variable, effectively substituting the variable with its corresponding value.

You can suppress variable substitution by preceding the `$` with a backslash (\), except within double quotation marks ("), which allow variable substitution. Single quotation marks (') suppress variable substitution. A `$` is interpreted literally if followed by a space, tab, or new-line. Also, braces ({ }) can be used to insulate a variable name from subsequent adjacent characters (for example, `${VAR}`).

Metasequences introduce variable values into the shell input. Except as noted, referencing an unset variable is an error. Most of the metasequences covered in this section can be modified using a set of word modifiers introduced in the *Word Modifiers* subsection of the *History Substitution* section in this manual page. These modifiers (`:h`, `:gh`, `:t`, `:gt`, `:r`, `:gr`, `:q` and `:x`) can be applied to some variable metasequences. If braces { } appear in the command form, the modifiers must appear within the braces. Only one `:`*x* modifier is allowed on each `$` expansion. The following metasequences can be modified:

`$`*var*
`${`*var*`}`

In the first form, *var* is replaced by the word(s) of the value of variable *var,* each separated by a blank. Braces insulate *var* from following characters that would otherwise be part of it (second form). If *var* is not a local shell variable, but an environment variable, then that value is returned (but `:`*x* modifiers and the other forms given as follows are not available in this case).

`$`*var*`[`*index*`]`
`${`*var*`[`*index*`]}`
Selects only the specified word, represented by *index*, from *var*. The *index* value can be represented by a single number or two numbers separated by a dash (–) to indicate a range. The first word of a variable's value is numbered 1. If the first number of a range is omitted, it defaults to 1. If the last member of a range is omitted, it defaults to `$#`*var* (see the next item). The argument, `*`, selects all words. It is not an error for a range to be empty if the second argument is omitted or within range.

`$#`*var*
`$#{`*var*`}`
Gives the number of words in the variable, which is useful with wordlists.

`$`*number*
`${`*number*`}`
Is equivalent to `$argv[`*number*`]`. Substitutes the value corresponding to the positional parameter given in the command line. For example, `$0` refers to the command name, `$1` refers to the first argument, `$2` refers to the second argument, and so on.

`$*`
`${*}`
Is equivalent to `$argv[*]`, which refers to all arguments.

The following metasequences cannot be modified.

`$?`*var*
`${?`*var*`}`
Returns the value of 1 if *var* is set; 0 if *var* is not set.

`$?0`
Returns the value of 1 if the current input filename is known, 0 if it is not set.

`$$`
Substitutes the (decimal) process number of the (parent) shell.

`$<`
Substitutes a line from the standard input, with no further interpretation. A shell script can use this form to read from standard input.

Commands for Setting Variables

`set`
`set` *name ...*
`set` *name* `=` *word*
`set` *name*`[`*index*`]` `=` *word*
`set` *name* `=` `(`*wordlist*`)`
The first form of the command lists the values of all variables. Values that form a list of multiple words are surrounded with parentheses. The second form sets *name* equal to the null string. The third form sets *name* equal to the single *word*. The fourth form sets the *nth* component of *name* (specified by *index*) equal to *word*. Before you can assign a single index value to a variable, the variable must exist. The final form sets *name* equal to the list of space-separated words in *wordlist*. In all forms, the value is command- and filename-expanded before it is assigned.

These arguments may be repeated to set multiple values in a single `set` command. Note, however, that variables in arguments are expanded before they are set.

`setenv` *name value*
: Sets the value of environment variable *name* equal to *value*, a single word. Two commonly used environment variables, `TERM` and `PATH`, are automatically imported to and exported from the C shell local variables, `term` and `path`. Setting these variables in the environment is unnecessary.

@
@ *name* = *expr*
@ *name*[*index*] = *expr*

The first form lists the values of all shell variables. The second form sets the specified *name* equal to the value of *expr*. If *expr* contains one of these four characters: `<`, `>`, `&`, or `|`, then at least this part of the expression must be placed within parentheses ( ).

The third form assigns the value of *expr* equal to the argument specified by the *index* of *name*. Both *name* and the *index* argument must already exist.

Other assignment operators in addition to = can be used. (Refer to the section on *Expressions and Operators* for more information.)

`unset` *name* ...
: Deletes specified variable.

`unsetenv` *name* ...
: Deletes specified environment variable. (See also `printenv`(1) for more information.)

Command Substitution

The output produced by a command can be substituted as an argument to some other command on the same command line. The command to be substituted is surrounded by backquotes (``).

`*command*`

Executes the backquoted command (`*command*`) and substitutes the result in the command line before executing the entire command line.

Command substitution is performed in a subshell. The output from such a command is normally broken into separate words at spaces, tabs, and new-lines. Null words are discarded.

Within double quotation marks (" "), only new-lines force new words; spaces and tabs are preserved.

The single, final new-line does not force a new word. Note that it is thus possible for a command substitution to yield only a partial word, even if the command outputs a complete line.

An example follows:

```
% echo The date of today is:  `date`
The date of today is: Mon Aug 29 16:59:57 EST 1988
```

Filename Substitution

Words containing any of these characters (`*`, `?`, `[`, or `{`) or beginning with the character (~) are candidates for filename expansion. Such a word represents a pattern that is matched and consequently replaced with an alphabetically sorted list of filenames containing the matched pattern.

`*` Matches zero or more character(s) in a filename.

`?` Matches any single character in a filename. For example, the `ls oct.?` command may produce this result: `oct.1`, `oct.2`, and `oct.9`, but not `oct.12`.

`[...]`
Matches any single character from the enclosed list, which is referred to as a *character class*. A list can also be expressed as a range, which is designated by two characters separated by a hyphen (-). The elements forming the beginning and end of a range must follow the ASCII collation sequence; as examples, `a-z` and `1-4`. For example, the `ls *.[123]` may produce this result: `oct.1`, `oct.2`, and `oct.3`, but not `oct.4`.

`~` Expands to the user's home directory, which is defined by the `home` shell variable. When the tilde is followed by a name consisting of alphabetic and numeric characters or the hyphen (-), the shell searches for the user's home directory and substitutes it in place of `~`. For example, `~ken` might expand to `/usr/ken` and `~ken/chmach` to `/usr/ken/chmach`. If the `~` is followed by a character that is not alphabetic or numeric, or a hyphen, or if the `~` does not appear at the beginning of a word, it is left undisturbed.

`{...}` Rather than having to type multiple full pathnames that contain a common component, you can type the commonly shared component only once and enclose the unique filename fragments in brackets. A comma follows each filename fragment.

This expression `a{b,c,d}e` is shorthand for `abe ace ade`. Left-to-right order is preserved. This construct may be nested. Thus,

```
~source/s1/{oldls,ls}.c
```

may expand to

```
/usr/source/s1/oldls.c  /usr/source/s1/ls.c
```

Successful filename expansions do not imply that the expanded filenames exist.

As a special case `{`, `}` and `{}` are passed undisturbed.

Expressions and Operators

Some of the built-in commands (see the section on *Built-in Commands*) take expressions as arguments, in which the operators are similar to those of the C language, with the same precedence. These expressions appear in the `@`, `exit`, `if`, and `while` commands, which are used to control the flow of executing commands.

Strings beginning with 0 are considered octal numbers. Null or missing arguments are considered 0. The results of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word. In most cases these characters (`&`, `|`, `<`, `>`, `(`, and `)`) should be surrounded by spaces.

The following operators are grouped in order of precedence.

`(...)` Change precedence.

`~` Perform 1's complement.

`!` Logically negate.

`* / %`
Multiply, divide, modulo.

`+  -`	Add, subtract.
`<<  >>`	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 match, filename substitution pattern mismatch. The operators, `=~` and `!~`, check for a match between the string on the left of the expression and a filename substitution pattern on the right of the expression. These two operators reduce the need for use of the `switch` statement in shell scripts when pattern-matching between strings is all that is needed.
`&`	Bitwise AND.
`^`	Bitwise XOR (exclusive or).
`|`	Bitwise inclusive OR.
`&&`	Logical AND.
`||`	Logical OR.

The assignment operators are given as follows:

`++`	Increment.
`--`	Decrement.
`=`	Assign.
`*=`	Multiply left side by right side and update left side.
`/=`	Divide left side by right side and update left side.
`+=`	Add left side to right side and update left side.
`-=`	Subtract left side from right side and update left side.
`^=`	Exclusive OR left side to right side and update left side.
`%=`	Modulo; divide left side by right side and update left side with remainder.

File status operators are available in the form *-l filename*, where *l* is one of the following values:

`-r` *filename*	Read access.
`-w` *filename*	Write access.
`-x` *filename*	Execute access.
`-e` *filename*	Existence.
`-o` *filename*	Ownership.
`-z` *filename*	Zero size.
`-f` *filename*	Plain file.
`-d` *filename*	Directory.

The specified filename is command- and filename-expanded and then tested for the file's status with regard to the user (for example, do you have write access?). If the file does not exist or is inaccessible, then all inquiries return a false value (0).

You can use a command as a boolean component in an expression. The status returned by the executing command determines the value of the expression. Successful command execution returns a true (0) value; failure returns a false (non-zero) value. The command name should appear in braces delimited by spaces (like `{`

/bin/m88k }).

An example of using a file status expression in a shell script follows:

```
echo "Enter a filename:"
set file = $<
if (-r $file && -w $file && -x $file) then
      echo "$file is readable, writable, and executable."
else
      echo "$file doesn't have correct permissions."
endif
```

Signal Handling and Status Reporting

The shell normally ignores quit signals. Background jobs running either by the & symbol, the bg command, or %*job*& command are immune to signals generated from the keyboard, including hangups. Other signals have values that the shell inherits from its parent. The shell's handling of interrupt and terminate signals in shell scripts can be controlled by the onintr commands. Log-in shells catch the terminate signal; otherwise, this signal is passed to children from the state in the shell's parent. Interrupts are not allowed when a login shell is reading the file .logout.

The shell learns immediately whenever a process changes state. It normally informs you when a job becomes blocked and prevents further progress, but only just before it prints a prompt so as to avoid disturbing your work. However, if the shell variable notify is set, the shell notifies you immediately of status changes in background jobs. There is also a shell command notify that marks a single process so that its status changes will be reported immediately. By default, notify marks the current process; after starting a background job, you can issue the notify command to mark it.

If you try to exit the shell while jobs are stopped, the shell displays a message indicating that there are stopped jobs. You can issue the built-in jobs command to list the currently stopped jobs. If you again attempt to terminate the shell while jobs are stopped, the shell will not issue a second warning, but will terminate the stopped jobs.

Parsing Order and Execution

The order in which the shell deals with special characters on the command line affects the result produced. Here are the steps of the parsing process:

History substitution	Expands the ! and ^ history metacharacters.
Line parsing	Divides the command line into words, numbering the first item 0; the second item, 1, and so on.
History update	Places the command line in the history list, allowing access to specific events and words.
Word parsing	Performs six separate steps, as follows:

' and " resolution

Both forms quote (prevent interpretation of) special characters for filename expansion, redirection, background execution, and pipes. Only single quotation marks (' ') quote variable expansion and command substitution special characters. Also, to prevent interpretation of the history metacharacter, ! must be quoted within single quotation marks.

Alias substitution
: Substitutes an alias with its assigned executable command.

`<`, `>`, `<<`, `>>`, `&`, and | recognition
: Recognizes these metacharacters but does not process them.

Variable substitution
: Expands variables to assigned values.

Command substitution
: Executes any command within backquotes (\` \`) and substitutes the result in standard output on the command line.

Filename substitution
: Expands filenames according to the filename metacharacters used.

Command Execution — After parsing occurs, the shell executes commands (informing the DG/UX system of the command to locate and the files to use for input and output).

International Features

`csh` can process characters from supplementary code sets in addition to ASCII characters. Characters from supplementary code sets can be used for command arguments, as values of variables, as alias name, and in comments and pipes.

Strings used in history substitution can contain characters from supplementary code sets.

Searches and pattern matching using metacharacters are performed in character units, not bytes.

?	Matches an character from supplementary code sets.
*	Matches any 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 specified in the range is matched.

C-shell scripts may also contain characters from supplementary code sets.

FILES

`/etc/login.csh`	Is read at login.
`~/.cshrc`	Is read at beginning of execution by each shell.
`~/.login`	Is read by login shell, after `~/.cshrc` at login.
`~/.logout`	Is read by login shell, at logout.
`/bin/sh`	Identifies the standard shell (for shell scripts not starting with `#! /bin/csh`)
`/tmp/sh*`	Identifies the temporary file for `<<`.
`/etc/passwd`	Specifies the source of home directories for *~username*.
`~/.history`	Contains the current history list saved at logout.

LIMITATIONS

A word cannot exceed 1024 characters. The DG/UX system limits an argument list to 10,240 characters. The number of arguments to a command that involves filename expansion is limited to 1/6 the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the C shell restricts the number of alias substitutions on a single line to 20.

SEE ALSO

`printenv`(1), `sh`(1), `stty`(1), `access`(2), `exec`(2), `fork`(2), `killpg`(2), `pipe`(2), `umask`(2), `wait`(2), `jobs`(3), `a.out`(4), `editread`(5), `environ`(5), and `ttcompat`(7).

NOTES

When a command is restarted from a stop, the shell prints the directory it started in if different from the current directory; this can be misleading, since the job may have changed directories internally.

If a process reading from a pipe ends before the process writing to the pipe, a harmless "Broken Pipe" error message is printed. Example: `ls -l | head` in a large directory.

Shell built-in functions cannot be stopped and then restarted. Command sequences of the form "*a ; b ; c*" also are not handled gracefully when you attempt to stop them. If you suspend *b*, the shell will then immediately execute *c*. This is especially noticeable if this expansion results from an alias. You can place the sequence of commands in parentheses to force it to a subshell. For example, "*(a ; b ; c)*" will suffice.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by `?`, are not placed in the history list. Control structure should be parsed rather than be recognized as built-in commands. Such a change would allow you to place the control commands anywhere, to combine control structure with pipes (`|`), and to use with commands executed in the background (`&`) and sequentially (`;`).

It should be possible to use the `:` *word* modifiers on the output of command substitutions. Furthermore, multiple modifiers (at least more than one) should be allowed on variable substitutions.

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 files labeled 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+1: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a – then standard input is used.

Options are:

`-s` Suppresses the printing of all character counts. `csplit` normally prints the character counts for each file created.

`-k` Leaves previously created files intact. `csplit` normally removes created files if an error occurs.

`-f` *prefix* The created files are named *prefix*`00` . . . *prefixn*. The default is `xx00 . . .  xx`*n*.

The arguments (*arg1* . . . *argn*) to `csplit` can be a combination of the following:

/rexp/ Create a file for the section from the current line up to (but not including) the line containing the regular expression *rexp*. The current line becomes the line containing *rexp*. This argument may be followed by an optional + or – some number of lines (e.g., `/Page/-5`).

%rexp%
The same as */rexp/*, but no file is created for the section.

linenum Create a file from the current line up to (but not including) *linenum*. The current line becomes *linenum*.

{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 *linenum*, the file will be split every *linenum* 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.

International Features

`csplit` can process characters from supplementary code sets. In regular expressions, searches are performed on characters, not bytes.

Option:

`-f`*prefix*
Characters from supplementary code sets can be used for *prefix*.

EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, `cobol00` ... `cobol03`. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file  100  {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The `-k` option retains the created files if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c  '%main(%'  '/^}/+1'  {20}
```

Assuming that `prog.c` follows the normal `C` coding convention of ending routines with a `}` at the beginning of the line, this example will create a file containing each separate `C` routine (up to 21) in `prog.c`.

DIAGNOSTICS

Self explanatory except for:

```
arg - out of range
```

This means that the given argument did not reference a line between the current position and the end of the file.

SEE ALSO

`ed`(1), `sh`(1).
`regexp`(5) in the *Programmer's Reference for the DG/UX System*

NAME

ct – spawn login to a remote terminal

SYNOPSIS

ct [-w*min*] [-x*lev*] [-h] [-v] [-s*speed*] *telno* ...

where:

min	The maximum number of minutes that ct is to wait for a line
lev	The debugging level, a single digit in the range 0-9
speed	The baud rate; default = 1200
telno	A telephone number comprising up to 31 characters: 0 thru 9, – (delay), = (secondary dial tone), *, and #

DESCRIPTION

Ct dials the telephone number of a modem that is attached to a terminal and spawns a login process to that terminal. 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. If there are no free lines, ct asks whether 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.

Options

-w Override the dialogue asking for the number of minutes to wait.

-x Produce (for debugging) a detailed output of the program execution on stderr. -x9 is the most useful value.

-h Prevent ct from hanging up the current line (the default, so the line can answer the incoming call), and wait for the termination of the ct process before returning control to the user's terminal.

-v Send a running narrative to the standard error output stream.

-s Set the data rate.

Destination Terminal Logout

After the user on the destination terminal logs out, ct prompts, Reconnect? If the response begins with the letter n, the line will be dropped; otherwise, login will be started again and the login: prompt will be printed.

Note that the destination terminal must be attached to a modem that can answer the telephone.

FILES

/etc/uucp/Devices
/usr/adm/ctlog

SEE ALSO

cu(1), ttymon(1M), login(1), uucp(1).

NOTES

For a shared port, one used for both dial-in and dial-out, the ttymon program running on the line must have the -b option specified (see ttymon(1M)).

NAME

cu − call another UNIX system

SYNOPSIS

cu [-s *speed*] [-b7|*8*] [-l *line*] [-h] [-t] [-d] [-o|-e] [-n] *telno*
cu [-s *speed*] [-b7|*8*] [-h] [-d] [-o|-e] -l *line*
cu [-b7|*8*] [-h] [-d] [-o|-e] *systemname*

DESCRIPTION

cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files.

cu accepts the following options and arguments:

-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.
-b7|*8*	Forces cu to use 7 or 8 bit characters. The default value depends on the current settings of ISTRIP and CS. If your modem is set for 8 bit operation, you must use 8 bit characters to achieve proper results.
-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. 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. The specified device is generally a directly connected asynchronous line (e.g., /dev/tty*ab*) in which case 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. Use of this option with *systemname* rather than *telno* will not give the desired result (see *systemname* below).
-h	Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.
-t	Used to dial an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.
-d	Causes diagnostic traces to be printed.
-o	Designates that odd parity is to be generated for data sent to the remote system.
-n	For added security, will prompt the user to provide the telephone number to be dialed rather than taking it from the command line.
-e	Designates that even parity is to be generated for data sent to the remote system.
telno	When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.
systemname	A uucp system name may be used rather than a telephone number; in this case, cu will obtain an appropriate direct line or telephone number

from `/etc/uucp/Systems`. Note: the *systemname* option should not be used in conjunction with the `-l` and `-s` options as `cu` will connect to the first available line for the system name specified, ignoring the requested line and speed.

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 (^s/^q) is used to control input from the remote so the buffer is not overrun. Lines beginning with ~ have special meanings as described below:

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.
	For both `~%take` and `put` commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
`~~` *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`(7) structure variables for the user's terminal (useful for debugging).
`~l`	prints the values of the `termio`(7) structure variables for the remote communication line (useful for debugging).
`~%nostop`	toggles between DC3/DC1 (^s/^q) input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output redirection to a file when a line from the remote begins with ~.

Data from the remote is redirected (or appended, if >> is used) to *file* on the local system. A trailing ~> marks the end of the redirection.

The use of `~%put` requires `stty`(1) and `cat`(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of `~%take` requires the existence of `echo`(1), `cat`(1) and `test`(1) on the remote system. `~%Take` will not work if you are logged in to a C shell on the remote system; the C shell does not support the `test` command. 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.

International Features

`cu` sets the input and output conversion mode to on or off, as appropriate, to avoide a character conversion on *LOCAL* system when accessing the *REMOTE* system.

On the *REMOTE* system, the input and output conversion should be set manually, as `cu` cannot know whether input conversion is required or not. In most cases, *REMOTE* systems can be used with input conversion on, however when transfering files, this should be set to off before invoking the file transfer command in order to avoid unexpected conversion of the file contents.

EXAMPLES

To dial a system whose telephone number is 9 1201 555 1212 using 1200 baud (where dialtone is expected after the 9):

```
cu  -s1200  9=12015551212
```

If the speed is not specified, "Any" is the default value.

To login to a system connected by a direct line:

```
cu  -l  /dev/ttyXX
```

or

```
cu -l ttyXX
```

To dial a system with a specific line and a specific speed:

```
cu  -s1200  -l  ttyXX
```

To dial a system using a specific line associated with an auto dialer:

```
cu  -l  culXX  9=12015551212
```

To use a system name:

```
cu  systemname
```

FILES

`/etc/uucp/Systems`
`/etc/uucp/Devices`
`/usr/spool/locks/LCK..`*(tty-device)*

DIAGNOSTICS

Exit code is zero for normal exit, otherwise, one.

SEE ALSO

`cat`(1), `ct`(1), `echo`(1), `stty`(1), `uname`(1), `uucp`(1).

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.

There is an artificial slowing of transmission by `cu` during the `~%put` operation so that loss of data is unlikely.

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut -c*list* [*file1 file2* ...]
cut -f*list* [-d*char*] [-s] [*file1 file2* ...]

DESCRIPTION

Use cut to cut out columns from a table or fields from each line of a file. In database parlance, cut 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). Either the -c or -f option must be specified. Cut can be used as a filter; if no files are given, the standard input is used.

Options are:

list	A comma-separated list of integer field numbers (in increasing order), with optional - to indicate page ranges, e.g., 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field).
-c*list*	The *list* following -c (no space) specifies character positions (e.g., -c1-72 would pass the first 72 characters of each line).
-f*list*	The *list* following -f is a list of fields assumed to be separated in the file by a delimiter character (see -d); e.g., -f1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless -s is specified.
-d*char*	The character following -d is the field delimiter (-f option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.
-s	Suppresses lines with no delimiter characters in case of -f option. Unless specified, lines with no delimiters will be passed through untouched.

International Features

cut can process characters from supplementary code sets.

Options:

-c*list*	Positions *list* must be specified as column positions rather than characters. 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.
-d*char*	The field delimiter *char* can be a character from a supplementary code set.

EXAMPLES

```
$ who | cut -c1-11
nespole
hoopes
wadsworth
carpenter
simmons
degeorge
parnagian
eydenberg
rosenberger
```

Usually, the `who` command gives username, tty number, and date and time that the user logged on the system. This information can be piped through the cut command, and the result is a list of users currently on the system.

Hints

Use `grep(1)` to make horizontal "cuts" (by context) through a file, or `paste(1)` to put files together 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

`line too long` A line can have no more than 1023 characters or fields or the Newline is missing.

`bad list for c/f option`
Missing `-c` or `-f` option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

`no fields` The *list* is empty.

`no delimeter` Missing *char* on `-d` option.

`cannot handle multiple adjacent backspaces`
Adjacent backspaces cannot be processed correctly.

`cannot open` *filename*
Either *filename* cannot be read or does not exist. If multiple filenames are present, processing continues.

SEE ALSO

`grep(1)`, `paste(1)`.

NAME

`date` – print and set the date

SYNOPSIS

`date` [`-u`] [`+` *format*]
`date` [`-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).

`-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.

`-u`
: Display (or set) the date in Greenwich Mean Time (GMT—universal time), bypassing the normal conversion to (or from) local time.

mm	is the month number
dd	is the day number in the month
HH	is the hour number (24 hour system)
MM	is the minute number
cc	is the century minus one
yy	is the 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 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

International Features

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*.

EXAMPLE

The command

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

generates as output:

```
DATE: 08/01/76
TIME: 14:45:05
```

DIAGNOSTICS

`No permission`	You are not the super-user and you try to change the date.
`bad conversion`	The date set is syntactically incorrect.

SEE ALSO

`sysadm`(1M), `strftime`(4), `environ`(5).

NOTES

Should you need to change the date while the system is running multi-user, use the `System->Date->Set` option in `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.

NAME

dc – desk calculator

SYNOPSIS

dc [*file*]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but you can specify an input base, output base, and a number of fractional digits to be maintained. (See bc(1), a preprocessor for dc that provides infix notation and a C-like syntax that implements functions. bc also provides reasonable control structures for programs.) The overall structure of dc is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. 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 and possibly, extended digits, for radices greater than 10. Extended digits, e.g. A-F in base 16, must be specified as capital letters only. A number may be preceded by an underscore (_) to input a negative number. Base 10 numbers may contain decimal points.

+ – / * % ^
: The top two values on the stack are added (+), subtracted (–), multiplied (*), or divided (/). The percent sign (%) shows the remainder when the two values are divided and the caret (^) shows the result when one of the two numbers is used as an exponent of the other. 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 at zero. If the l is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. P interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack are printed.

q Exits the program. If executing a string, the recursion level is popped by two. If q is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x Treats the top element of the stack as a character string and executes it as a string of dc commands.

X Replaces the number on the top of the stack with its scale factor.

[...]
: Puts the bracketed ASCII string onto the top of the stack.

<*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 DG/UX 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.
`; :`	Used by `bc` for array operations.

EXAMPLES

```
$ cat dc.infile1
10sa2sb
lad+p
lalb/p
q
$ dc dc.infile1
20
5
$
```

In this example, the `dc` command uses `dc.infile1` for its input. The first line of `dc.infile1` causes the a and b registers to be loaded (using "l" for load) with the values 10 and 2, respectively. The next 2 lines use the a and b register values (using "l" for load again) to perform some arithmetic. The second line pushes the a register on the stack, duplicates it and then adds the two numbers together. The third line pushes the b register and the a register on the stack and then divides b by a. In both cases the result is pushed on the stack, and the p causes the result to be printed to the output file. The 20 is the result of 10+10 and the 5 is the result of 10/2.

```
$ cat dc.infile2
[3 5 * p]
x

$ dc dc.infile2
15
<Ctrl-D>
$
```

In this example, `dc.infile2` is used for the input for `dc`. The first line of the input pushes the strings of commands onto the stack. The `x` command treats the top of the stack as a string of commands and executes them. The `p` in the string of commands causes the result (15) to be written as output. You press `Ctrl-D` to end execution.

```
$ cat dc.infile3
[6 4 + p] sa
5
5
=a
q

$ dc dc.infile3
10
$
```

In this example, `dc` uses `dc.infile3` for its input. The first line of the input causes register a to be loaded with the given string of commands ([6 4 + p]). The next two lines cause 2 5s to be pushed on the stack. The =a command then checks the top 2 values on the stack to see if they are equal. Since they are, the commands in the a register are executed. The result (10) is written as output.

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.

SEE ALSO

bc(1).

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 size 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 (default 512)
bs=*n*	Set both input and output block size, superseding ibs and obs ; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	Conversion buffer size
skip=*n*	Skip *n* input blocks before starting copy
iseek=*n*	Seek *n* blocks from beginning of input file before copying
oseek=*n*	Seek *n* blocks from beginning of output file before copying
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
lcase	Map alphabetics to lowercase
ucase	Map alphabetics to uppercase
swab	Swap every pair of bytes
noerror	Do not stop processing on an error
sync	Pad every input block to ibs
. . . , . . .	Several comma-separated conversions
files=*n*	Allows concatenation of multiple input files for tape
conv=*block*	Convert to blocked files
conv=*unblock*	Convert to unblocked files

Where sizes are specified, the 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 a product.

Cbs is used only if ascii or ebcdic conversion is specified. In the former case, cbs characters are placed into the conversion buffer, converted to ASCII, trailing blanks trimmed, and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output block of size cbs.

After completion, dd reports the number of whole and partial input and output blocks.

EXAMPLES

To convert a file from lowercase to uppercase:

```
$ cat file1
this file contains only lower case letters.
$ dd if=file1 of=file2 conv=ucase
$ cat file2
THIS FILE CONTAINS ONLY LOWER CASE LETTERS.
$
```

To write `file0`, `file1`, and `file2` to magnetic tape:

```
$ dd if=file0 of=/dev/rmt/0n conv=sync
$ dd if=file1 of=/dev/rmt/0n conv=sync
$ dd if=file2 of=/dev/rmt/0 conv=sync
```

DIAGNOSTICS

f+p `blocks in(out)`
Numbers of full and partial blocks read(written)

SEE ALSO

cp(1).

NOTES

The ASCII/EBCDIC conversion tables are taken from the 256-character standard in the CACM Nov, 1968. These do not always correspond to certain IBM® print train conventions. There is no universal solution.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

`deblock` – change blocking size

SYNOPSIS

`deblock` [-i] [-o] [-f]

DESCRIPTION

`Deblock` reads from standard input and writes to standard output. This utility is used to change the blocking factor between input and output. An intermediate buffer stores bits of information to be sent to output. The user selects the size of this buffer. If this buffer is sufficiently large, limited streaming is possible on streamer tapes.

The available options for `deblock` are as follows:

`-i`*size*	Input buffer (record) size in bytes. The default is 5120.
`-o`*size*	Output buffer (record) size in bytes. The default is 1024.
`-f`*size*	The factor used to determine the intermediate buffer size. The intermediate buffer size is determined by multiplying the size of the input buffer by the value of `-f` and then adding the size of the output buffer.

To facilitate streaming of tapes, reads with buffer size specified by the "i" option are performed until the intermeditate buffer is filled. When the intermediate buffer fills, writes with buffer size specified by the "o" option are performed until the intermediate buffer is empty. The "f" option sets the intermediate buffer size.

EXAMPLE

The following command sequence reads a file, compresses it, and writes it to a tape in cpio format. If the tape is a streaming tape, it will stream during the time that the intermediate buffer is being output. The output buffer size is selected for streaming of B media. The intermediate buffer size is 540,672 (512 * 1024+16384).

```
echo foo|cpio -ob|compress|deblock -i512 -o16384 -f1024
```

SEE ALSO

dd(1).

NAME

deroff – remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [-m *x*] [-w] [*files*]

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), tbl(1) descriptions, and pic descriptions, 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 or l. The -mm option causes the macros to be interpreted so that only running text is output (i.e., no text from macro lines.) The -ml option forces the -mm option and also causes deletion of lists associated with the mm macros.

If the -w option is given, the output is a word list, one word per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. 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). *Using the Documenter's Tool Kit on the DG/UX System.*

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.
Erroneous results can occur if eqn(1) constructs are not closed.
The -ml option does not handle nested lists correctly.

NAME

`dg_kill` – test for or terminate a process

SYNOPSIS

`dg_kill` [`-lnp`] [*-signo* | *-signame*] [`--`] *name* | *PID* ...

DESCRIPTION

`Dg_kill` tests for the existence of the specified processes and, optionally, sends a specified signal to those processes. Sending a signal will normally kill processes that do not catch or ignore the signal.

A process can be selected by specifying its simple filename, *name*, or its process number. The *name* is the command name by which the process was invoked with any leading directory components omitted. Regular expressions, as used with `ed`(1), may be used in the *name*. All regular expression patterns are anchored as if specified with a leading "^" and followed by a "$".

The process number, *PID*, of each asynchronous process started with `&` is reported by the shell unless more than one process is started in a pipeline. If more than one process is started in a pipeline, the number of the last process in the pipeline is reported. Process numbers can also be found by using `ps`(1).

If a negative process number is specified, all processes in the process group to which the positive process number belongs will be signalled. If process number 0 is specified, all processes in the current process group are signalled. The signalled process must belong to the current user unless he or she is the superuser. See `kill`(2) for more information.

If a signal number, *signo*, or signal name, *signame*, preceded by – is given, that signal is sent to the process. (see `signal`(2) or `/usr/include/sys/signal.h`). Signal number 9 (as in `dg_kill -9 . . .`) is a sure kill. Signal names may include or omit the leading "SIG". Use the `-l` option to obtain a list of acceptable names.

If no signal number or signal number 0 is specified, no signal will be sent and `dg_kill` will merely test to see if the any of the selected processes are running. The exit code will indicate whether any are or not.

Options

`-l`	List all signal numbers and names, and then exit.
`-n`	Invert the exit code, effectively testing for "is not running" instead of "is running". `dg_kill` will exit with a zero exit code if none of the specified processes are running, or if there was any error. A non-zero exit code will result if any of the specified processes are running.
`-p`	Print the PID and command name for all processes found to be running. This option is ignored when specifying processes by process number.
-sig	The number or name of the signal to be sent to all selected processes.

EXAMPLES

```
$ if dg_kill lpsched
> then
>     echo lpsched is running
> fi
```

In this example, `dg_kill` is used to determine if a command by the name of `lpsched` is running.

```
$ dg_kill -SIGTERM 'rpc..*'
```

In this example, the `dg_kill` command is used to terminate all processes with names matching the regular expression pattern "`rpc..*`". Note the use of quotes to avoid shell interpretation of special characters.

EXIT CODES

The following are the normal exit values:

0	At least one process that matched the selection criteria was found to be running.
1	No running processes match the selection criteria.
2	An error occurred obtaining process information.
3	There was a syntax error in the command line.

When the `-n` option is used, the non-zero exit codes are replaced by zero and 1 is returned if any process was found to be running.

No message is displayed when a specified process number or name does not match a running process. The exit code is the only indication given.

SEE ALSO

`csh`(1), `ed`(1), `kill`(1), `ps`(1), `sh`(1).
`kill`(2), `signal`(2) in the *Programmer's Reference for the DG/UX System (Volume 1)*.

NOTES

Unlike `kill`(1), `dg_kill` does not signal selected processes unless a signal number is supplied.

Use care when specifying process names, especially when using regular expression patterns, because other users may be running commands with names similar to the names you are trying to select. Note that regular expression patterns are not the same as shell wildcard characters.

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`

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

SEE ALSO

`bdiff`(1), `cmp`(1), `comm`(1), `ed`(1), `pr`(1).

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 that occurs 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 are suppressed.

`-e` Produce a script for the editor `ed`(1) that will incorporate into *file1* all changes between *file2* and *file3*, i.e., 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), `berk_diff`(1), `berk_diff3`(1).

NOTES

Text lines that consist of a single `.` will defeat `-e`.
Files longer than 64K bytes will not work.

NAME

`dircmp` – compare two directories

SYNOPSIS

`dircmp` [`-d`] [`-s`] [`-l`*x*] [`-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 filenames common to both directories have the same contents.

`-d`	Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in `diff`(1).
`-s`	Suppress messages about identical files.
`-l`*x*	Change the page length of the output to *x* lines. The default length is 66.
`-w`*n*	Change the width of the output line to *n* characters. The default width is 72.

International Features

Characters from supplementary code sets can be used for directory names, and the specified directory can contain files with names using supplementary characters.

Options:

`-w`*n*	The width of the output line *n* is in columns, not in characters or bytes. Multibyte characters spanning over the specified width are not displayed.

EXAMPLES

```
dircmp SOURCE SOURCE.2
```

Compares the two directories, `SOURCE` and `SOURCE.2`. Prints the names of files that are unique to each directory, and identifies files that have the same name and identical or different contents.

```
dircmp -s SOURCE SOURCE.2
```

Compares the files in the directories `SOURCE` and `SOURCE.2`. Lists the files that are unique to each directory, and the files with the same name but different contents. Suppresses the printing of file names that have identical contents.

```
dircmp -d -w80 SOURCE SOURCE.2
```

Compares the files in `SOURCE` and `SOURCE.2`. Lists the files that are unique to each directory, files with the same name but identical or different contents, and creates scripts to make different files identical. `dircmp` truncates the output lines so they are only 80 characters long.

SEE ALSO

cmp(1), `diff`(1).

NAME

dispgid – display 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

SEE ALSO

dispuid(1), groups(1).

NAME
`dispuid` – display 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

SEE ALSO
`dispgid`(1), `listusers`(1).

NAME

domainname – set or display name of the current NIS domain

SYNOPSIS

domainname [*name-of-domain*]

DESCRIPTION

Without an argument, domainname displays the name of the current domain, which typically encompasses a group of hosts under the same administration. As such, the name of an NIS domain is normally also a valid Internet domain name, and can be used in conjunction with sendmail(1C) and the name server named(1M).

Only the superuser can set the name of the domain by giving domainname an argument; this is usually done in the startup script /usr/sbin/init.d/rc.ypserv using an argument defined in /etc/nfs.params.

SEE ALSO

named(1M), sendmail(1C), ypinit(1M).

NAME

download – download host resident PostScript fonts

SYNOPSIS

`/usr/lib/lp/postscript/download` [`-f`] [`-p` *printer*] [`-m` *name*] [`-H` *dir*] [*file*] ...

where:

printer	The (directory) name for a particular printer.
name	A font map table.
dir	The host font directory.
file	A PostScript input file.

DESCRIPTION

`download` prepends needed host resident fonts to each input *file* and writes the results on the standard output. If no *file* is specified, or if – is given for any *file* argument, `download` reads from its standard input. `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.

Options are:

`-f` Force a complete scan of each input *file*. Without the `-f` option, the default scan stops immediately after the PostScript header comments, unless the following explicit comment appears in the header to point `download` to the end of the file:

```
%%DocumentFonts: (atend)
```

`-p` *printer* Check the list of printer-resident fonts in `/etc/lp/printers/`*printer*`/residentfonts` before downloading. Any fonts found in this list are assumed to be on the printer already.

`-m` *name* Use *name* as the font map table. A *name* that begins with `/` is taken as 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/share/lib/hostfontdir`.

Requested fonts are named in a comment (marked with `%%DocumentFonts:`) in the input *file*s. Available fonts are listed in the map table selected using the `-m` option.

The map table consists of space-separated 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 assumed to be relative to the host font directory. Blank lines and comments are allowed in the map table file. Comments are introduced by `%` (as in PostScript) and extend to the end of the current 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 cannot 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
```

FILES

/usr/lib/lp/postscript	Default host resident font directory
/etc/lp/printers/*/residentfonts	Host resident font tables for specific printers

DIAGNOSTICS

An exit status of 0 is returned if each *file* was successfully processed.

SEE ALSO

dpost(1), postdaisy(1), postdmd(1), postio(1), postmd(1), postprint(1), posttek(1).

NOTES

The facilities provided by download 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.

NAME

`dpost` – `troff` postprocessor for PostScript printers

SYNOPSIS

`/usr/lib/lp/postscript/dpost` [`-cemnopwxyFHLOT` *arg*] [*files*]

where:

arg	An argument to an option, described below
files	The name(s) of one or more input 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` d*ir*	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/postscript/dpost.ps
/usr/lib/postscript/color.ps
```

```
/usr/lib/postscript/draw.ps
/usr/lib/postscript/forms.ps
/usr/lib/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

du − summarize disk usage

SYNOPSIS

du [-amrs] [*name* ...]

where:

name The name of a directory or regular file; default = .

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *name* argument. The block count includes the indirect blocks of the file.

A file with two or more links is counted only once.

Options are:

-a	Generate an entry for each file.
-m	Traverse only those subdirectories that are within the same file system (i.e., don't cross file system mount points). Normally, du traverses all subdirectories within each directory when calculating disk usage.
-r	Generate a message for each directory that cannot be read, each file that cannot be opened, etc. By default du is silent in such instances.
-s	Give only the grand total for each of the specified directories. Absence of either -a or -s generates an entry for each directory only.

EXAMPLES

```
du | egrep '^[1-9][0-9][0-9]+'
```

Displays the directories and sub-directories which have a larger block count than 100. The output of du is piped to egrep, and egrep searches for strings that start with the number 100 or greater.

SEE ALSO

df(1M).

NOTES

If the -a option is not used, non-directories given as arguments are not listed.
If there are too many distinct linked files, du will abort.
Files with holes in them will get an incorrect block count.

NAME

echo – echo arguments

SYNOPSIS

echo [*arg*] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \. Arguments are as follows:

\b	Backspace
\c	Print line without new-line
\f	Form-feed
\n	New-line
\r	Carriage return
\t	Tab
\\	Backslash
\n	The 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number *n*, which must start with a zero.
\v	Vertical tab

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

International Features

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 (\).

EXAMPLES

```
$ echo Hello World
```

Prints the string "Hello World" on the standard output.

```
$ echo *
```

Prints the names of the files and subdirectories in the current working directory. This output is similar to that of ls(1), but the file and directory names are listed in one long line. It is usually quicker than ls(1).

```
$ if test ! -r book
> then echo "file is not readable"
> fi
```

Tests if you do *not* have read permission for the file "book". Echo prints "file is not readable" if the result of the "test ! -r book" command is true.

SEE ALSO

sh(1), csh(1).

NAME

ed, red – text editor

SYNOPSIS

ed [-s] [-p *string*] [-x] [-C] [*file*]

red [-s] [-p *string*] [-x] [-C] [*file*]

DESCRIPTION

ed is a 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.

-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) (U.S. versions of the DG/UX software only; encryption mechanisms are not included in international distributions of DG/UX software). 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 (e.g., 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 RE. The REs allowed by ed are constructed as follows:

The following "*one-character*" *RE*s match a *single* character or collation element (see below):

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE 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* RE (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* RE (see 4.2 below).

  d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the g command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([ ]) is a "one-character" RE that matches any one character or one collating symbol in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character or collating symbol *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 (-) can indicate a range of consecutive characters or collating symbols. For example, in most locales, [0-9] is equivalent to [0123456789]. In an expression such as [*c*-*d*], if both *c* and *d* are single-byte characters (i.e. they occupy only one byte in a multibyte string), then the ordering of characters is defined by the collation order specified in the LC_COLLATE table for the current locale (see *environ*(5) and *setlocale*(3C)). In the C locale, the collation order is defined by the numerical (ASCII) value of the character, and by definition all characters are single-byte.

Multibyte characters may be included in a [ ] range. Constructs such as [*mm*-*nn*], where *mm* and *nn* represent multibyte characters, are allowed, as long as the two characters are in the same code page. In this case, collation order information is not used, and the range is handled by simple numeric comparisons of the character values.

Multi-character collating symbols (e.g. the Spanish "ch") can be included in a [ ] range either explicitly, as described below, or implicitly. For example, in the Spanish locale, [c-d] would match "c", "ch", or "d", while in the English locale, the same expression would match only "c" or "d". Any given [ ] range may contain either multi-character collation symbol(s) (either explicitly or implicitly), or it may contain multibyte character(s), but not both.

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); e.g., `[]a-f]` matches either a right square bracket (]) or one of the letters `a` through `f` inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

Within square brackets, the following constructs have special meanings:

a. Collating symbols are expressed as `[.`*cd*`.]`. This can be used to represent any multi-character collating symbol that is declared in the collation table, as if it were a single character.

b. Equivalence classes are expressed as `[=`*c*`=]`. Such a construct matches any character or collating symbol that has the same relative ordering in the current collation sequence as *c*. *c* must be a single-byte character.

c. Character class expressions are expressed as `[:`*classname*`:]`. The character classes are defined by the LC_CTYPE category of the current locale (see *ctype*(3C)). The acceptable values of *classname* are:

classname	Meaning
alpha	an alphabetic letter
upper	an upper-case alphabetic letter
lower	a lower-case alphabetic letter
digit	a decimal digit
xdigit	a hexidecimal digit
alnum	an alphabetic letter or a decimal digit
space	a character that produces white space in displayed text
punct	a punctuation character
print	a printing character
graph	a character with a visible representation
cntrl	a control character

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by `\{`*m*`\}`, `\{`*m,*`\}`, or `\{`*m,n*`\}` is a RE that matches a *range* of occurrences of the one-character RE. 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 RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences `\(` and `\)` is a RE that matches whatever the unadorned RE 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 RE. 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 RE may be constrained to match words.

3.1 `\<` constrains a RE 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 RE must be a digit, underscore, or letter.

3.2 `\>` constrains a RE to match the end of a string or to precede a character that is not a digit, underscore, or letter.

An *entire RE* 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 RE constrains that RE to match an *initial* segment of a line.

4.2 A dollar sign (`$`) at the end of an entire RE constrains that RE to match a *final* segment of a line.

4.3 The construction `^`*entire RE* `$` constrains the entire RE to match the entire line.

The null RE (e.g., `//`) is equivalent to the last RE 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 RE enclosed by slashes (`/`) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before FILES below.
6. A RE enclosed in question marks (`?`) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before FILES below.
7. An address followed by a plus sign (`+`) or a minus sign (`-`) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. 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; e.g, `-5` is understood to mean `.-5`.

9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so −− refers to the current line less 2.

10. For convenience, a comma (`,`) stands for the address pair `1,$`, while a semicolon (`;`) stands for the pair `.,$`.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (`,`). They may also be separated by a semicolon (`;`). In the latter case, the 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 `a`ppend 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 `c`hange 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 `d`elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

`e` *file*

The `e`dit command 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 `E`dit command is like `e`, except that the editor does not check to see if any changes have been made to the buffer since the last `w` command.

`f` *file*

If *file* is given, the `f`ile-name command changes the currently remembered file name to *file*; otherwise, it prints the currently remembered file name.

`( 1 , $ )g/`*RE*`/`*command list*

In the `g`lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with 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/*RE*/

In the interactive Global command, the first step is to mark every line that matches the given RE. 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 (and optionally the error message restricted shell; see the h and H commands below).

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 (e.g., *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 the current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

`( . , . )s`*/RE/replacement/* or
`( . , . )s`*/RE/replacement/*`g` or
`( . , . )s`*/RE/replacement/n* *n* = 1-512
: The `substitute` command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator `g` appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number *n* appears after the command, only the *n*-th occurrence of the matched string on each addressed line is replaced. It is an error if the substitution fails on *all* addressed lines. Any character other than space or new-line may be used instead of `/` to delimit the RE 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 RE on the current line. The special meaning of `&` in this context may be suppressed by preceding it by `\`. As a more general feature, the characters `\`*n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between `\(` and `\)`. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of `\(` starting from the left. When the character `%` is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The `%` loses its special meaning when it is in a replacement string of more than one character or is preceded by a `\`.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by `\`. Such substitution cannot be done as part of a `g` or `v` command list.

`( . , . )t`*a*

This command acts just like the `m` command, except that a *copy* of the addressed lines is placed after address `a` (which may be 0); 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`*/RE/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 RE.

`( 1 , $ )V`*/RE/*

This command is the same as the interactive global command `G`, except that the lines that are marked during the first step are those that do *not* match the RE.

`( 1 , $ )w`*file*

The 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`

(Not available in international distributions of the DG/UX system; U.S.

versions only.) 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 RE or of a replacement string (e.g., `/`) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2          s/s1/s2/p
g/s1             g/s1/p
?s1              ?s1?
```

International Features

ed can process characters from supplementary code sets as well as ASCII characters.

ed supports internationalized regular expressions as specified by XPG3.

Searches and pattern matching with regular expressions are performed in character units, not in individual bytes.

A prompt string containing characters from supplementary code sets can be designated in *string* using the `-p` option.

Marks set by the `k` command must be ASCII lower case letters.

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.

SEE ALSO

`edit`(1), `ex`(1), `grep`(1), `sed`(1), `sh`(1), `stty`(1), `umask`(1), `vi`(1).
`fspec`(4), `regexp`(5) in the *System Manager's Reference for the DG/UX System*.

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 Encryption 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 RE does not match a new-line character.

If the editor input is coming from a command file (e.g., `ed` *file* `<` *ed_cmd_file*), the editor exits at the first failure.

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. 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`(1).

Options

`-r`	Recover *file* after an editor or system crash. If *file* is not specified, a list of all saved files will be printed.
`-x`	Encryption option; when used, the file will be encrypted when it is written and will require an encryption key to be read. When reading a file, `edit` makes an educated guess as to whether the file is encrypted or not. See `crypt`(1).
`-C`	Encryption option; same as `-x` except that when reading in a file, `edit` assumes that the file is encrypted.

Brief Introduction

To edit the contents of an existing file you begin with the command "edit name" to the shell. `Edit` makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and invoke `edit` with the filename. If no file by that name already exists, `edit` will create one and print `[NEW FILE]` on your screen.

`Edit` prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in `edit`'s buffer (its name for the copy of the file you are editing). Most commands to `edit` use its "current line" if you do not tell them which line to use. When you invoke `edit` for an existing file, the last line of the file is your current line. If you type `print` (which can be abbreviated `p`) and press Newline (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 typically the next line in the file. If you `delete` this last line, then the new last line becomes the current one. In general, after a `delete`, the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the `append` (`a`) command can be used. After you give this command (typing a Newline after the word `append`) `edit` will read lines from your terminal until you give a line consisting of just a "`.`", placing these lines after the current line. The last line you type then becomes the current line. The command `insert` (`i`) is like `append` but places the lines you give before, rather than after, the current line.

`Edit` numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then `edit` will print this first line. If you then give the command `delete` `edit` will delete the first line, line 2 will become line 1, and `edit` will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the `substitute` (`s`) command. You type "s/*old* /*new*/" where *old* is the old characters you want to get rid of and *new* is the new characters you want to replace *old* with.

The command `file` (`f`) will tell you how many lines there are in the buffer you are editing and will print "[Modified]" if you have changed the buffer. After modifying a file you can replace the file with the buffer text by giving a `write` (`w`) command. You can then leave the editor by issuing a `quit` (`q`) command. If you run `edit` on a file, but do not change it, it is not necessary (but does no harm) to `write` the file back. If you try to `quit` from `edit` after modifying the buffer without writing it out, you will be warned that there has been "No `write` since last change (:quit! overrides)" and `edit` will await another command. If you wish not to `write` the buffer out then you can issue another `quit!` command (note the exclamation point). The buffer is then irretrievably discarded, and you return to the shell.

By using the `delete` and `append` commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use `edit` more than a few times.

The `change` (`c`) command will change the current line to a sequence of lines you supply (as with `append` you add lines up to a line consisting of only a "."). You can tell `change` to change more than one line by giving the line numbers of the lines you want to change, e.g., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The `undo` (`u`) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a `substitute` command which does not do what you want, you can type `undo` and the old contents of the lines will be restored. You can also `undo` an `undo` command so that you can continue to change your mind. `Edit` will give you a warning message when your commands affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an `undo` and looking to see what happened. If you decide that the change is ok, then you can type `undo` again to get it back. Note that commands such as `write` and `quit` cannot be undone.

To look at the next line in the buffer you can just hit Newline. To look at a number of lines type ^D (the control key and, while it is held down, the D key, then let up both) rather than Newline. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by typing "´´". The `z` command can also be given other following characters: "z−" prints a screen of text (or 24 lines) ending at the current line; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.11" to get 11 lines total (your former current line is now in the center of the 11 lines, and the last line printed is your new current line). This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them. However, since the line numbers change when you insert and delete lines, this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form */text/* to search forward for *text* or *?text?* to search backward for *text*. If a search reaches the end of the file without finding the text it wraps around to the other end of the file and continues to search back to the line where you are. A useful feature here is a search of the form `/^text/` which searches for *text* at the beginning of a line. Similarly `/text$/` searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "`.`"; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$−5" is the fifth before the last, and ".+20" is 20 lines after the current.

You can find out which line you are at by typing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (e.g., 10 to 20). For a move you can then type "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. `Edit` has 26 such buffers named `a` through `z`. You can later get these lines back by doing "put a" to put the contents of buffer `a` after the current line. If you want to move or copy these lines between files you can give an `edit` (`e`) command after copying the lines, following it with the name of the other file you wish to edit, e.g., "edit chapter2". By changing `delete` to `yank` above you can get a command for copying lines. If the text you wish to move or copy is all within one file then you can just type "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

`ex`(1), `vi`(1).

NAME

egrep – search a file for a pattern using full regular expressions

SYNOPSIS

egrep [*options*] *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 extended ("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 full regular expressions as in ed(1), except for \(and \), and the other execptions mentioned below under **International Features**, with the addition of:

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 new-line 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 new-line.

If no files are specified, egrep assumes standard input. Normally, each line found is copied to the 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. This is valid for single byte characters only.

-h Suppress printing of filenames 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).

-v Print all lines except those that contain the pattern.

-e *special_expression*
Search for a *special expression* (*full_regular_expression* that begins with a -).

-f *file*
Take the list of full regular expressions from *file*.

International Features

egrep can process characters from supplementary code sets. In regular expressions, searches are performed on characters, not on individual bytes.

egrep does *not* support the following international features in regular expressions that are described in ed(1):

`[.`*ch*`.]`	multi-character collation symbol
`[=`*c*`=]`	collation-order equivalence class
`[:`*alpha*`:]`	character class

Moreover, character ranges such as `[`*a–j*`]` are interpreted by simply comparing the numeric values of the character bytes, not by using collation ordering information.

EXAMPLES

```
$ egrep fs2 /etc/passwd
```

Searches through the file "/etc/passwd" and prints all lines containing the pattern "fs2" on the standard output.

```
$ egrep -l -e -ooutfile src/*
```

Searches through all the files in the subdirectory "src" for all lines containing the regular expression "-ooutfile". Prints the names of the files containing the pattern.

```
$ egrep 'int|long' prog.c
```

Searches through the file "prog.c" in the current working directory for all lines containing the pattern "int" or the pattern "long". The "|" character stands for logical "or". Prints all of the lines that contain "int" or "long" on the standard output.

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 submitted 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 submitted 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* specified. This *reason* is reported by `lpstat -p`. *reason* must be enclosed in quotes if it contains blanks. The default reason is `unknown reason` for existing printers, and `new printer` for printers just added to the system but not yet enabled.

`-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 Features

Characters from supplementary code sets can be used for *value*, *command* and *args*.

SEE ALSO

sh(1).
exec(2), profile(4), environ(5) in the *Programmer's Reference for the DG/UX System*

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) in the *Programmer's Reference for the DG/UX System.*
`ldterm`(7), `streamio`(7) in the *System Manager's Reference for the DG/UX System.*

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.

SEE ALSO

`crypt`(1), `ed`(1), `edit`(1), `grep`(1), `sed`(1), `sort`(1), `vi`(1).
`curses`(3X) in the *Programmer's Reference for the DG/UX System*.
`term`(4), `terminfo`(4) in the *System Manager's Reference for the DG/UX System*.
Using the DG/UX System.
Using the DG/UX Editors.
`curses` (ETI) and `terminfo` chapters of *Programmer's Guide: Character User Interface (FMLI and ETI)*.

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, 2s 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.

International Features

expr can process characters from supplementary code sets in addition to ASCII characters. In regular expressions, pattern searches are performed on characters, not bytes.

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
```

Here is a better 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 : '.*'
```

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

SEE ALSO

ed(1), sh(1).

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 data base. The data base used depends on the run-time 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. A second example is that a string could be in a form of an escape sequence which could not 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)

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.

SEE ALSO

gettxt(1), mkmsgs(1), printf(1), srchtxt(1).
gettxt(3C), printf(3S), setlocale(3C), environ(5) in the *Programmer's Reference for the DG/UX System*.

NAME

`factor` – factor a number

SYNOPSIS

`factor` [*number*]

where:

number
: An integer from 1 to 100,000,000,000,000

DESCRIPTION

When `factor` is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 10^{14} it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If `factor` is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to $\sqrt{n}$ and occurs when n is prime or the square of a prime.

EXAMPLES

```
$ factor 12
12
        2
        2
        3
```

The above command lists the prime factors for the number 12.

DIAGNOSTICS

The message "Ouch!" is displayed for input out of range or for garbage input.

SEE ALSO

`expr(1)`.

NAME

`fez` – display file element sizes

SYNOPSIS

`fez` *filename*...

DESCRIPTION

For each *filename*, `fez` produces a line of output containing the

a. *filename*

b. the data element size for *filename* (in blocks)

c. the data element size for *filename* (in bytes, Kbytes, Mbytes, or Gbytes)

d. the index element size for *filename* (in blocks)

e. the index element size for *filename* (in bytes, Kbytes, Mbytes, or Gbytes)

DIAGNOSTICS

`fez` fails and returns a non-zero exit code if *filename* does not exist. Otherwise, 0 is returned.

SEE ALSO

`file`(1), `ls`(1).

NAME

fgrep − search a file for a character string

SYNOPSIS

fgrep [*options*] *string* [*file* ...]

DESCRIPTION

fgrep (fast grep) seaches files for a character string and prints all lines that contain that string. fgrep is different from *grep(1)* and *egrep(1)* 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. Since 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 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.
-h	Suppress printing of filenames when searching multiple files.
-i	Ignore upper/lower case distinction during comparisons. This is valid for single byte characters only.
-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).
-v	Print all lines except those that contain the pattern.
-x	Print only lines matched entirely.

-e *special_string*
    Search for a *special string* (*string* begins with a -).

-f *file*
    Take the list of *strings* from *file*.

International Features

fgrep can process characters from supplementary code sets.

Searches are performed on characters, not on individual bytes.

EXAMPLES

```
$ ps -af | fgrep -x -f expfile
```

Searches through the list of active processes for lines that entirely match the lines in "expfile".

```
$ find . -exec | fgrep -l attachment {} \;
```

Prints the names of all files under the current working directory that contain the string "attachment".

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.

International Features

`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

EXAMPLES

$ file file1
```
file1:          commands text
```

Since **file1** contains DG/UX shell commands (i.e. it is a script file), the `file` command gives the type as "commands text".

$ file -f filenames
```
file1:          commands text
```

```
file2:          ascii text
file3:          c program text
file4:          DG m88k pure executable
```

The file **filenames** contained the names of four files.

FILES

`/etc/magic`

DIAGNOSTICS

If the `-h` option is specified and *arg* is a symbolic link, `file` prints the error message:

`symbolic link to` *arg*

SEE ALSO

`filehdr(4)` in the *System Manager's Reference for the DG/UX System*.

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*.

Arguments that are normally numeric are terminated by the first non-numeric character in the argument, and the remaining characters are ignored. If the argument's first character is not numeric, the argument is considered equivalent to 0. Valid expressions are:

`-name` *pattern*	True if *pattern* matches the current file name. Normal shell file name 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.
`-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* is indicated by the space-semicolon. To keep the shell from interpreting this semicolon as the end of your command line, you should precede it with a backslash (see example). A command argument `{ }` is replaced by the current path name.
`-ok` *cmd*	Like `-exec` except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing `y`.
`-print`	Always true; causes the current path name to be printed.
`-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` block special file `c` character special file `d` directory `p` fifo (named pipe) `f` plain file `l` symbolic link file `s` socket files in the AF _UNIX domain
`-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 yields 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).

Only those primaries necessary to establish whether the expression is true or false are evaluated. Thus, for an expression concatenating two primaries, the second primary is evaluated only if the first primary is true. For an expression alternating two primaries, the second primary is evoked only if the first primary is true.

Note that when you use `find` in conjunction with `cpio`, if you use the `-L` option with `cpio` then you must use the `-follow` expression with `find` and vice versa. Otherwise there will be undesirable results.

International Features

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*.

Expressions:

-name *file*
: Characters from supplementary code sets can be used in *file*.

-exec *cmd*

-ok *cmd*
: Characters from supplementary code sets can be used in *cmd*.

EXAMPLES

```
$ find . -perm 777 -print
./a.out
./ed_script
./shell_script
```

The above example searches the working directory and all directories below the working directory for files with read, write, and execute permission for owner, group, and others.

```
$ find . -print -perm 0777 -exec rm {} \;
```

The name of every file below '.' is printed. Those files whose permissions are 0777 are deleted. Note the backslash that precedes the semicolon terminating the rm command.

```
$ find . -perm 0777 -o -print
```

The name of each file whose permissions are not 0777 is printed.

```
$ find . -name a.out -print
./a.out
```

The above example searches the working directory and all directories below it for a specific file.

```
$ find / -name passwd -print
/bin/passwd
/etc/passwd
/usr/etc/yp/src/passwd
```

The above example searches for the passwd file starting from the root directory. In this case, three different passwd files were found in different directories.

```
$ find $HOME \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

The above example removes all files in your home directory named a.out or *.o that have not been accessed for a week.

```
$ find . -name SCCS -prune -o -print
```

The above example recursively prints all file names in the current directory and below, but skipping SCCS directories.

```
$ find . -print -name SCCS -prune
```

The above example recursively prints all file names in the current directory and below, skipping the contents of SCCS directories, but printing out the SCCS directory name.

FILES

`/etc/passwd`, `/etc/group`.

SEE ALSO

`chmod`(1), `cpio`(1), `sh`(1), `test`(1).
`stat`(2), `cpio`(4), `fs`(4) in the *Programmer's Reference for the DG/UX System*.

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.

WARNING

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 (512-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), 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)

Options

`-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

fmt – simple text formatter

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

`fmt` can handle input lines up to 4094 bytes long. Input lines greater than 4095 bytes will be truncated at the 4095th byte and the remainder of the line will be discarded.

The `fmt` command accepts a *-width* option for BSD compatibility, but it may go away in future releases.

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 an 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.

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:138
```

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`.

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.

SEE ALSO

`addseverity`(3C), `fmtmsg`(3C) in the *Programmer's Reference for the DG/UX System (Volume 2)*.

NAME

`fold` – fold long lines for finite width output device

SYNOPSIS

`fold` [*–width*] [*file* ...]

DESCRIPTION

`Fold` is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using `col`(1) before `fold` processes the input.

SEE ALSO

`cat`(1), `col`(1), `newform`(1).

NOTES

`Fold` may distort underlining.

NAME

`ftp` – use file transfer program

SYNOPSIS

`ftp [ -v ] [ -d ] [ -i ] [ -n ] [ -g ] [` *host* [*port*]]

DESCRIPTION

The `ftp` program is the user interface to the Internet standard File Transfer Protocol (FTP). The program lets a user transfer files to and from a remote network site.

You may specify the client *host* with which `ftp` is to communicate on the command line. The `ftp` program will then try to establish a connection to an FTP server on that host and enter the command interpreter. Otherwise, `ftp` will enter its command interpreter and await instructions.

Options

The `-v` (verbose on) option forces `ftp` to show all responses from the remote server, as well as report on data transfer statistics. If `ftp` is invoked from the terminal, the verbose option is on by default.

The `-n` option keeps `ftp` from attempting auto-login upon initial connection. You must use the `user` command. If auto-login is enabled, `ftp` will check the `.netrc` file in the user's home directory for an entry listing a login, password, and account for the remote machine. (Because the `.netrc` file contains your username and passwords, `ftp` requires you to restrict access to this file. To restrict access to yourself only, set your access mode to 600.) The sample `.netrc` entry

```
machine remote1 login gerry password fastcar
```

with the username `gerry` and the password `fastcar` will allow you to auto-login to `remote1`.

If no entry exists in the `.netrc` file, `ftp` will provide as a default the user name associated with the real user ID on the local machine as the user identity on the remote machine. For example, if you had used `su(1)` to become root, `ftp` would provide root as the default name rather than your login name. Then, `ftp` will prompt for a password (if required) and, optionally, will prompt for an account with which to log in.

The `-i` option turns off interactive prompting during multiple file transfers.

The `-d` option enables debugging.

The `-g` option disables filename globbing.

When `ftp` is awaiting commands from the user, it shows a prompt: `ftp>`. If you omit one or more arguments to a command, `ftp` will generally either prompt for the arguments one at a time or print a "help" message that explains the correct way to use the command. The `ftp` program recognizes the following commands:

`abort`
: Abort the previous file transfer command. If `abort` is invoked when a data transfer has been interrupted, output from the transfer is aborted. The data connection closes and a reply is sent to the user indicating that the service request terminated abnormally.

`account` *account-number*
: Send an account number for a system logon or access to a specific process. During the login procedure, `ftp` automatically prompts you for your account number if one is needed.

`append` *local-file* [*remote-file*]
: Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local filename is used in naming the remote file. If *remote-file* does not exist, it will be created. File transfer uses the current settings for *type*, *mode*, and *structure*.

`bell`
: Sound a bell after some of the file transfer commands are completed.

`binary`
: Set the file transfer type to binary. You should use binary type for any non-text files such as executable programs or object files, compressed files, and tar or cpio files to ensure data integrity during transfer. This command is the same as the `type binary` and `type image` commands.

`bye`
: Terminate the file transfer session with the remote server and exit `ftp`.

`cd` *remote-directory*
: Change the working directory on the remote machine to *remote-directory*.

`cdup`
: Change the working directory on the remote machine to the parent directory.

`close`
: Terminate the file transfer session with the remote server and return to the local command interpreter.

`delete` *remote-file*
: Delete the file *remote-file* on the remote machine.

`debug` [*debug-value*]
: Toggle debugging mode. If you specify an optional *debug-value*, it is used to set the debugging level. Setting the debug level to zero turns debugging off; setting it to any other value turns debugging on. When debugging is on, `ftp` prints each command sent to the remote machine, preceded by the string -->.

CAUTION: We cannot guarantee that all commands will function normally in debug mode. Expect some unusual results.

`dir` [*remote-directory* [*local-file*]]
: Print a listing of the directory contents in the directory *remote-directory* and, optionally, place the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal. If the remote directory does not exist, nothing is returned.

`disconnect`
: A synonym for `close`.

`exit`
: Abruptly terminate the FTP session and exit.

`get` *remote-file* [*local-file*]
: Retrieve the *remote-file* and store it on the local machine. If the local filename is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *mode*, and *structure* are used while transferring the file.

`glob`
: Toggle filename globbing. With filename globbing enabled, each local file or pathname is processed for `csh(1)` metacharacters. These characters include *?[]~{}. Remote files specified in multiple item commands, e.g., `mget`, are globbed by the remote server. With

globbing disabled all files and pathnames are treated literally.

`hash` Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 2048 bytes.

`help` [*command*]
Print an informative message about the meaning of *command*. If no argument is given, `ftp` prints a list of the known commands.

`lcd` [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

`ls` [*remote-directory* [*local-file*]]
Print an abbreviated listing of the contents of a directory that is on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If *local-file* is specified, the listing is put there; otherwise, the output is sent to the terminal.

`mdelete` *remote-files*
Delete the specified files on the remote machine. If globbing is enabled, the specification of remote files will first be expanded using `ls`.

`mdir` *remote-files local-file*
Obtain a directory listing of multiple files on the remote machine and place the result in *local-file*.

`mget` [*remote-files*]
Retrieve the specified files from the remote machine and place them in the current local directory. If globbing is enabled, the specification of remote files will first be expanded using `ls`. If no files are specified, `mget` prompts for them.

`mkdir` [*directory-name*]
Make a directory on the remote machine. If *directory-name* is not specified, `mkdir` prompts you. If the directory already exists, `mkdir` tells you. It will not overwrite an existing directory.

`mls` [*remote-files* [*local-file*]]
Obtain an abbreviated listing of multiple files on the remote machine and place the result in *local-file*. If no files are specified, `mls` prompts you. If *local-file* does not exist, it is created.

`mode` [*mode-name*]
Set the file transfer *mode* to *mode-name*. If you do not specify a *mode-name*, `mode` displays the current mode. Three modes are available: *block*, *compressed*, and *stream*. The following table defines the available modes:

Mode	**Meaning**
`stream`	Transmits data as a stream of bytes without any restrictions on the type used. This is the default mode.
`block`	File is transmitted as a series of data blocks, each preceded by three header bytes. You can use record structures and any representation type in this mode.

`compress`
: Sends regular data, compressed data, and control information. Regular data is sent in a byte string, compressed data is sent in replications or fillers, and control information is sent in a two-byte sequence.

`mput` [*local-files*]
: Transfer multiple local files from the current local directory to the current working directory on the remote machine. If you do not specify *local-files*, `mput` prompts you for them. If a file does not exist, `mput` will give you an error message and continue.

`open` *host* [*port*]
: Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case `ftp` will try to contact an FTP server at that port. If the `auto-login` option is on (default), `ftp` will also try to automatically log the user in to the FTP server (see above).

`prompt`
: Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to let the user selectively retrieve or store files. If prompting is turned off, any `mget` or `mput` will transfer all files without interruption.

`put` *local-file* [*remote-file*]
: Store a local file on the remote machine. If *remote-file* is left unspecified, the local filename is used in naming the remote file. File transfer uses the current settings for *type*, *mode*, and *structure*. If *remote-file* already exists, it is overwritten.

`pwd`
: Print the name of the current working directory on the remote machine.

`quit`
: A synonym for `bye`.

`quote` *arg1 arg2 ...*
: Specified arguments are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return. This command is usually used for debugging or for working around local restrictions.

`recv` *remote-file* [*local-file*]
: A synonym for `get`.

`reinit`
: Terminate the user and reinitialize the command connection. Resets all transfer parameters to their default values. The command connection remains open.

`remotehelp` [*command-name*]
: Request help from the remote FTP server. If a *command-name* is specified, a more informative message about the *command-name* is given. If no *command-name* is specified, the remote server displays a list of known commands.

`rename` [*from*] [*to*]
: Rename the file *from* on the remote machine, to the file *to*. If no names are specified, `rename` prompts you for them. If *from* does not exist, an error is reported; if *from* is specified but *to* is not, `rename` shows you a syntax description. If the *to* file already exists, it is overwritten.

`restart` Restart the last transfer aborted by a system crash. The transfer restarts where it was aborted.

CAUTION: Only files transferred in `compress` or `block` transfer mode can be restarted. You must use `restart` before you begin any other data transfer.

`rmdir` *directory-name*
Delete a directory on the remote machine.

`runique` Toggle the use of unique naming of files transferred from other systems. When on, if a file on the local machine has the same name as the file transferred, a number is appended to the filename of the transferred file. The numbers assigned per transfer run consecutively from 1-99. `runique` is off by default.

`send` *local-file* [*remote-file*]
A synonym for `put`.

`sendport` Toggle the use of `port` commands. By default, `ftp` tries to use a `port` command when establishing a connection for each data transfer. If the `port` command fails, the default data port will be used. When `port` commands are disabled, no attempt will be made to use them for each data transfer.

`site` Display information about the remote system. The DG/UX system supplies the following format:

Operating System:	DG/UX
Storage Structure:	File
Storage Representation Type:	ascii
Storage Filler:	NULL
Local byte size allowed:	multiple of 8 bits
Default Page size:	2048 bytes
can be changed using PAGE command	
Default EOR Delimiter:	<NL>
can be changed using SEOR command	

`status` Show the current status of the local and remote environments. Displays the current values for the transfer parameters (*mode*, *type*, *format*, and *structure*) and modes (`verbose`, `bell`, `prompt`, `hash`, `globbing`, `sendport`, `runique`, and `sunique`).

`struct` [*s-name*]

Set the file transfer *structure* to *s-name*. The default structure is *file*. The table below lists the file transfer structures and what they mean:

Structure	**Meaning**
`file`	There is no internal structure. The file is a continuous sequence of bytes.
`page`	The file is made up of independent indexed pages.
`record`	The file is made up of sequential records.

The DG/UX operating system does not support `record` structured files. If you specify `record` structure, all EOR delimiters will be converted to `<NL>` for storage.

`page` structure will be accepted with only the Local Byte type and is supported only in the stream mode (see `type` command below).

`sunique` Toggle the use of unique naming of files transferred to other systems. When on, if a file on the remote machine has the same name as the file transferred, a number is appended to the filename of the transferred file. The numbers assigned per transfer run consecutively from 1-99. `sunique` is off by default.

`type` [*t-name* [*vertical-format*]]

Set the file transfer *type* to *t-name*. If no type is specified, the current type is printed. The default type is network ASCII. If you include the *t-name*, the type is set to *t-name*. *T-name* is the character transfer type.

Some transfer types have formats. Choose the formats by substituting a string for *vertical-format*. See the following table for the transfer types and available vertical-formats:

Type	**Vertical-formats**
`ascii`	[`non-print` | `telnet` | `carriage-control`]
`ebcdic`	[`non-print` | `telnet` | `carriage-control`]
`binary`	
`image`	
`local-byte`	[*byte_size*]

NOTE: *byte_size* must be a multiple of 8 bits.

The `binary` transfer type is the same as the `image` transfer type and the `binary` command.

You should not use `ascii` type to transfer binary files because data may be lost due to the carriage-return and newline translation between the server (`ftpd`) and the client (`ftp`). You should use `binary` type for for any non-text files such as executable programs or object files, compressed files, and `tar` or `cpio` files to insure data integrity during the transfer.

The *vertical-format* determines the vertical controls and how the information is represented on a printing device. The default *vertical-format* is *non-print*. The following list defines the *vertical-formats*:

Vertical-format	**Description**
`non-print`	The file need not contain vertical format information. A printer process can assume standard values for spacing and margins. Typically, this format is used with files that will be stored or processed.
`telnet` format controls	The file contains ASCII/EBCDIC vertical format controls, such as <CR>, <LF>, and <FF>, that the printer process can interpret. The sequence <CRLF> denotes the end-of-line.
`carriage-control`	The file contains American National Standards Institute (ANSI) FORTRAN vertical format control characters. If lines and records are formatted according to the ANSI standard, vertical format controls are read in before the data is printed.

`user` *username* [*password* [*account*]]
Identify yourself to the remote FTP server. If the password is not specified and 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 user will be prompted for it. Unless `ftp` is invoked with auto-login disabled (`-n` option), this process is done automatically on initial connection to the FTP server.

`verbose` Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. If verbose is on when a file transfer completes, statistics on the efficiency of the transfer are reported. Verbose mode is on by default when `ftp` is invoked from a terminal.

`?` [*command*] A synonym for `help`.

`!` Invoke a shell on the local machine.

Command arguments that have embedded spaces may be enquoted with quote (") marks.

File Naming Conventions

Files specified as arguments to `ftp` commands are processed according to the following rules:

1) If the filename "-" is specified, `stdin` (for reading) or `stdout` (for writing) is used.

2) If the first character of the filename is a vertical line (|), the rest of the argument is interpreted as a shell command. `ftp` then forks a shell, using `popen(3S)` with the argument supplied, and reads or writes from the `stdout` or `stdin`, respectively. If the shell command includes spaces, the argument must be enquoted; e.g., "`| ls -lt`".

3) Failing the above checks, if globbing is enabled, local filenames are expanded according to the rules used in `csh(1)`.

File Transfer Parameters

Several parameters control the transmission and the representation of data as the data is transferred. These transfer parameters are *mode*, *structure*, and *type*. *Mode* defines how the data bits are transferred, while *structure*, and *type* define how the data is represented as it is being transferred. For more information about these parameters, see the commands `mode`, `type`, and `struct` above.

If you want a transferred file to be identical to the original file, make sure the transfer parameters are appropriately set before transferring the file.

Interrupting a File Transfer

FTP allows you to interrupt a file transfer that is in progress. To interrupt a file transfer, enter the interrupt process character. The interrupt character can be different from system to system (for DG/UX, the interrupt character is usually ^C). The interrupt character suspends the data transfer and displays a menu on the screen. The menu lists the commands available. All menu commands except `help` exit the menu upon completion.

The following table lists and explains the available commands.

Command	**Function**
interrupt character	Terminates the FTP-user process.
`abort`	Aborts data transfer, closes data connection, but leaves command connection open.
`quit`	Completes data transfer, closes data connection, terminates user, and closes command connection.
`reinit`	Completes data transfer, terminates user, but leaves the command connection open.
`status`	Displays status information and continues data transfer.
`continue`	Continues the transfer.
`help` [*command*]	Displays available commands or syntax for one of the available commands.

NOTE: If you enter the interrupt character when no data is in transfer, your `ftp` user process will terminate.

SEE ALSO

`rcp`(1C), `tftp`(1C), `inet`(3N), `hosts`(4).

BUGS

Many FTP server implementations that you might connect with do not support experimental operations such as print working directory; they also may not work correctly if data transfer is interrupted.

Errors are not handled consistently, especially in the `mdelete`, `mdir`, `mget`, and `mput` commands. Before executing a command, check to see that the files you want to transfer exist. Also, after executing a command, check to see that the file transfer was successful.

NAME

`gencat` – generate a formatted message catalogue

SYNOPSIS

`gencat` [`-l`] *catfile [msgfile ...]*
`gencat` [`-a`] *catfile [msgfile ...]*
`gencat -d` *catfile*

DESCRIPTION

The `gencat` utility processes X/Open-style message catalogs. `gencat` 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 `-l` option can be used to list the messages as they are read in from the catfile.

The `-a` option generates an AT&T style message catalog that can be accessed by the *gettxt(3C)* function. If this option is selected, then the source message file should contain only one set which is numbered 1. The *catopen(3C)*, *catgets(3C)*, and *catclose(3C)* functions cannot be used on catalogs that are generated with the *gencat −a* option.

With the `-d` option, `gencat` writes to its standard output a text version of *catfile* in the format of a *msgfile*. This file can then be modified and reprocessed with `gencat` (without the `-d` option). If no file by the name *catfile* can be found, the environment variable NLSPATH is used to locate it, using the same algorithm as *catopen(3C)*.

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
```

SEE ALSO

`catexstr`(1), `catgets`(1), `catopen`(3C), `catgets`(3C), `catclose`(3C), `nl_types`(5) in the *Programmer's Reference for the DG/UX System*.
`mkmsgs`(1), `gettxt`(3C) — AT&T-style message facility.

NAME

getopt – parse command options

SYNOPSIS

set -- \`getopt *optstring* $*\`

where:

optstring
: 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 can be separated from it by white space. The special option -- delimits 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.

DESCRIPTION

Getopt organizes options in command lines for easy parsing by shell procedures and to check for legal 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 to process the arguments for a command that can take the options a, b, or o, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)       FLAG=$i; shift;;
        -o)            OARG=$2; shift 2;;
        --)            shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SEE ALSO

sh(1), getopt(3C).

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 legal 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`(1) command. (See the **CAUTIONS**, 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 the option letter by white space.

Each time it is invoked, `getopts` will place 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`.

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 in *optstring*.

By default, `getopts` parses the positional parameters. If extra arguments (*arg* ...) are given on the `getopts` command line, `getopts` will parse them instead.

`/usr/lib/getoptcvt` reads the shell script in *file*, converts it to use `getopts`(1) instead of `getopt`(1), and writes the results on the standard output.

`-b` the results obtained by running `/usr/lib/getoptcvt` will be portable to earlier releases of the DG/UX 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`(1) or `getopt`(1).

So all new commands will adhere to the command syntax standard described in `intro`(1), they should use `getopts`(1) or `getopt`(3C) to parse positional parameters and check for options that are legal for that command (see **CAUTIONS**, below).

International Features

Characters from supplementary code sets can be read as the argument to *optstring*.

EXAMPLES

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 will accept 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
```

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

SEE ALSO

intro(1), sh(1).
getopt(3C) in the *Programmer's Reference for the DG/UX System*.

CAUTIONS

Although the following command syntax rule (see intro(1)) exceptions are permitted under the current implementation, they should not be used because they may not be supported in future releases of the operating system. As in the **EXAMPLES** section above, a and b are options, and the option o requires an option-argument:

cmd -aboxxx file (Rule 5 violation: options with option-arguments must not be grouped with other options)
cmd -ab -oxxx file (Rule 6 violation: there must be white space after an option that takes an option-argument)

Changing the value of the shell variable OPTIND or parsing different sets of arguments may lead to unexpected results.

NAME

gettxt – retrieve a text string from a message data base

SYNOPSIS

gettxt *msgfile*:*msgnum* [*dflt_msg*]

DESCRIPTION

gettxt retrieves a text string from an AT&T-style 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 in the directory /usr/lib/locale/*locale*/LC_MESSAGES to retrieve *msgnum* from. The name of *msgfile* can be up to 14 characters in length, but may not contain either \0 (null) or the ASCII code for / (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-null, then it will display the value of *dflt_msg*; if *dflt_msg* is not present or is null, then it will display the string Message not found!!.

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/UX/*msgfile*. 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) in the *Programmer's Reference Manual*.
gencat(1), catgets(1), catopen(3C), catgets(3C) — X/Open-style message facilities.

NAME

glossary – definitions of common terms and symbols

SYNOPSIS

[help] glossary [*term*]

DESCRIPTION

The DG/UX system Help Facility command glossary provides definitions of common technical terms and symbols.

Without an argument, glossary displays a menu screen listing the terms and symbols that are currently included in glossary. A user may choose one of the terms, or the user may quit and exit to the shell by typing q. When a term is selected, its definition is retrieved and displayed. By selecting the appropriate menu choice, the list of terms and symbols can be redisplayed.

A term's definition may also be requested directly from shell level (as shown above), causing a definition to be retrieved and the list of terms and symbols not to be displayed. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following is a table that lists the symbols and their escape sequences.

SYMBOL	SH(1) ESCAPE SEQUENCE
""	\"\"
’’	\’\’
.	\\.
[]	\\[\\]
‘‘	\‘\‘
#	\#
&	\&
*	*
\	\\\\
|	\\|

When invoking glossary(1) from the csh, the escape sequences are the same as in the sh(1) except for the square brackets ([ ]). The escape sequence for [ ] is \[\].

From any screen in the Help Facility, a user may execute a command via the shell (sh(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt of the Help facility. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), set the shell environment variable SCROLL to no. In the Bourne shell, sh, do this by adding the following line to your .profile file [see profile(4)]:

```
SCROLL=no; export SCROLL
```

In the csh, add the following line to your .login file:

```
setenv SCROLL no
```

If you later decide that you prefer scrolling, set SCROLL to yes.

Information on each of the Help Facility commands (`starter`, `locate`, `usage`, `glossary`, and `help`) is located on their respective manual pages.

SEE ALSO

`help`(1), `helpadm`(1M), `locate`(1), `csh`(1), `sh`(1), `starter`(1), `usage`(1).
`term`(5) in the *Programmer's Reference for the DG/UX System*.

NOTES

If the shell variable `TERM` (see `sh`(1)) is not set in the user's `.profile` file or `.login`, `TERM` will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to `term`(5).

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 internationalized simple ("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. This is 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 Features

`grep` can process characters from supplementary code sets, as well as ASCII characters. Searches are performed on characters, not individual bytes.

Within `[ ]` expressions, `grep` recognizes international regular expression constructs such as:

`[.`*ch*`.]`	multi-character collation symbol
`[=`*c*`=]`	collation-order equivalence class
`[:`*alpha*`:]`	character class

These constructs are described in *ed*(1).

EXAMPLES

```
$ grep root /etc/passwd
```

Prints the lines in the file "/etc/passwd" that contain the login name "root".

```
$ who | grep "xyz"
```

Prints the name, terminal number, and time that the user with login name "xyz" logged in if "xyz" is logged in. If "xyz" is not logged in, this command line prints nothing.

```
$ grep rsh /etc/passwd|cut -d: -f5
```

Searches the "/etc/passwd" file for users who run a restricted shell, `rsh`(1). Then cut(1) prints the fifth field of every line that grep identifies. The fifth field contains the users' names.

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).

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, grep will only match up to the first null; if it matches, it will print the entire line.

NAME

groups – show group memberships

SYNOPSIS

groups [*user*]

where:

user	A user name listed in /etc/passwd or /etc/group

DESCRIPTION

The groups command shows the groups to which you or the optionally specified user belong. 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 that owns it, you are granted group access to the file.

FILES

/etc/passwd
/etc/group

SEE ALSO

chgrp(1), newgrp(1), setgroups(2), group(4).

NAME

head – give the first few lines

SYNOPSIS

head [*–count*] [*file* ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted, it defaults to 10.

EXAMPLES

```
$ cat example.txt
Bob Jackson      533-5645
Terry Johnson    443-9845
Mary Stanley     243-4837
Paul Davis       441-9384
Mary Wadsworth   435-5832
Karen Hoopes     564-6345
Susan Walbridge  352-5353
Claudine Dumont  463-8383
Carla Nespole    535-4325
Max Thompson     334-8453
Jennnifer Mas    334-3522
Melanie Jones    394-4857
```

The above example shows the contents of the example file.

```
$ head example.txt
Bob Jackson       533-5645
Terry Johnson     443-9845
Mary Stanley      243-4837
Paul Davis        441-9384
Mary Wadsworth    462-5832
Karen Hoopes      564-6345
Susan Walbridge   352-5353
Claudine Dumont   463-8383
Carla Nespole     535-4325
Max Thompson      334-8453
```

The above example of the command with no switches displays the first ten lines of the example file.

```
$ head -5 example.txt
Bob Jackson       533-5645
Terry Johnson     443-9845
Mary Stanley      243-4837
Paul Davis        441-9384
Mary Wadsworth    462-5832
```

The above example of the command with the switch displays the desired number of lines from the example file.

SEE ALSO

tail(1).

NAME

help – help facility

SYNOPSIS

```
  help
[ help ] starter
[ help ] usage [ -d ] [ -e ] [ -o ] [ command_name ]
[ help ] locate [ keyword1 [ keyword2 ] ... ]
[ help ] glossary [ term ]
  help sccs_command | sccs_message number
```

DESCRIPTION

The DG/UX system Help Facility provides on-line assistance for DG/UX system users, whether they desire general information about DG/UX or specific assistance for use of the Source Code Control System (SCCS) commands.

Without arguments, `help` prints a menu of available on-line assistance commands with a short description of their functions. The commands and their descriptions are:

COMMAND	DESCRIPTION
`starter`	Information about the DG/UX system for the beginning user.
`locate`	Locate DG/UX system commands using function-related keywords.
`usage`	DG/UX system command usage information.
`glossary`	Definitions of DG/UX system technical terms.

The user may choose one of the above commands by entering its corresponding letter (given in the menu), or may quit and exit to the shell by typing `q`.

With arguments, `help` directly invokes the named on-line assistance command, bypassing the initial `help` menu. The commands `starter`, `locate`, `usage`, and `glossary`, optionally preceded by the word `help`, may also be specified at shell level. When executing `glossary` from shell level, some of the symbols listed in the glossary must be escaped (preceded by one or more backslashes, "\'') to be understood by the Help Facility. For a list of symbols and how many backslashes to use for each, refer to the `glossary`(1) manual page.

From any screen in the Help Facility, a user may execute a command via the shell (`sh`(1)) by typing a `!` and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), set the shell environment variable `SCROLL` to `no`. In the Bourne shell, `sh`, do this by adding the following line to your `.profile` file (see `profile`(4)):

```
SCROLL=no; export SCROLL
```

In the `csh`(1), add the following line to your `.login` file:

```
setenv SCROLL no
```

If you later decide that you prefer scrolling, set `SCROLL` to `yes`.

Information on each of the Help Facility commands (`starter`, `locate`, `usage`, `glossary`, and `help`) is located on their respective manual pages.

The Help Facility can be tailored to a customer's needs by use of the `helpadm`(1M) command.

If the first argument to `help` is different from `starter`, `usage`, `locate`, or `glossary`, `help` assumes information is being requested about the SCCS facility. The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type1	Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, `ge3` for message 3 from the `get` command).
type2	Does not contain numerics (for example, a command, such as `get`).
type3	Is all numeric (for example, `212`).

If the first argument is not a true SCCS command or message number, the `help` main menu appears.

EXAMPLES

```
$ help get

get:
   get [-rSID] [-ccutoff] [-ilist] [-xlist] [-aseq_no]
    [-k] [-e] [-lp] [-p] [-m] [-n] [-s] [-b] [-g] [-t] file ...
$
```

`help` with an argument that is an SCCS command will display usage of that command.

```
$ help ge3

ge3:
"e not allowed with m"
You can't use both the -e and -m arguments on the same get command.
$
```

`help` with an error number from an SCCS module will display text associated with that error number.

SEE ALSO

`glossary`(1), `helpadm`(1M), `locate`(1), `csh`(1), `sh`(1), `starter`(1), `usage`(1). `admin`(1), `cdc`(1), `comb`(1), `delta`(1), `get`(1), `prs`(1), `rmdel`(1), `sact`(1), `sccsdiff`(1), `unget`(1), `val`(1), `vc`(1), `what`(1), `profile`(4), `sccsfile`(4), `term`(5) in the *Programmer's Reference for the DG/UX System*.

NOTES

If the shell environment variable `TERM` (see `sh`(1) and `csh`(1)) is not set in the user's `.profile` file or `.login` file, `TERM` will default to the terminal value type 450 (a hard-copy device). For a list of valid terminal types, refer to `term`(5).

NAME

`hostid` – set or print identifier of host system

SYNOPSIS

`hostid` [*identifier*]

DESCRIPTION

The `hostid` command (without an argument) prints the identifier of the current host in hexadecimal. This numeric value is expected to be unique across all hosts and is normally set to the host's Internet address. The superuser can set the `hostid` by specifying *identifier* as an argument. The parameter required at boot time is defined in

```
/etc/tcpip.params
```

and is used in

```
/usr/sbin/init.d/rc.tcpipport.
```

EXAMPLE

To specify a host with Internet address 128.211.10.4:

```
hostid 80d30a04
```

SEE ALSO

`hostname`(1C), `gethostid`(2), `sethostid`(2), `hostname`(4).

NOTE

Most programs look up the host name in the `hostname` database, rather than use `hostid`.

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [*nameofhost*]

DESCRIPTION

The hostname command (without an argument) prints the name of the current host. The superuser can set the *nameofhost* by specifying an argument. When the *nameofhost* is specified the command also sets the nodename for the system. The nodename can also be displayed with the uname command and is typically used with communication protocols other than TCP/IP, such as, uucp. The parameter used at boot time is defined in

```
/etc/tcpip.params
```

and is used in

```
. /usr/sbin/init.d/rc.tcpipport
```

SEE ALSO

uname(1), gethostname(2), sethostname(2), uname(2).

NAME

iconv – code set conversion

SYNOPSIS

iconv -f *fromcode* -t *tocode* [-m *mode*] [-v] [*file(s)*]

DESCRIPTION

Iconv converts the encoding of characters in *file(s)* 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*. The option *-v* verifies that the database is organised correctly; no action is taken. If no *file(s)* arguments are specified on the command line, iconv reads the standard input.

Iconv uses the database file /usr/lib/kbd/iconv_data. This file has 4 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.

Iconv matches the required arguments *fromcode* and *tocode* and the optional argument *mode* to the corresponding fields in the database.

The field *mode* does not have to be uniformly included or excluded from the database, i.e. it may be included in some rows and not in others. If the argument *mode* 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, as *kbdpipe* assumes that any *file* in the "-f *file*" argument that does not begin with "/" will be found in /usr/lib/kbd.

The codeset conversions supported in the supplied database are given in the table below.

Code Set Conversions Supported			
fromcode	tocode	modes	comment
ASCII	88591	d b e p	
88591	ASCII	d b e p	
6937	88591		Teletext
88591	6937		Teletext
646	88591	d	US Ascii
646DE	88591	d	German
646DK	88591	d	Danish
646GB	88591	d	English Ascii
646ES	88591	d	Spanish
646FR	88591	d	French
646IT	88591	d	Italian
646NO	88591	d	Norwegian
646SE	88591	d	Swedish
646PT	88591	d	Portugese
646YU	88592	d	Serbo Croation
88591	646	d b e p	7 bit Ascii
88592	646	d b e	
88591	646DE	d b e p	German
88591	646DK	d b e p	Danish
88591	646GB	d b e p	English Ascii
88591	646ES	d b e p	Spanish
88591	646FR	d b e p	French
88591	646IT	d b e p	Italian
88591	646NO	d b e p	Norwegian
88591	646SE	d b e p	Swedish
88591	646PT	d b e	Portugese
88592	646YU	d	Serbo Croatian
PC437	88591	d b e p	
88591	PC437	d b e p	
PC850	88591	d b e	
88591	PC850	d b e p	
PC860	88591	d b e	
88591	PC860	d b e	
PC863	88591	d b e	
88591	PC863	d b e	
PC865	88591	d b e	
88591	PC865	d b e	

Code Set Conversions Supported

fromcode	tocode	modes	comment
PC437	ASCII	d b e	
PC850	ASCII	d b e	
PC860	ASCII	d b e	
PC863	ASCII	d b e	
PC865	ASCII	d b e	
ASCII	88591	d b e	
ASCII	EBCDIC	d	
EBCDIC	ASCII	d	
ASCII	IBM_EBCDIC	d	
PC437	EBCDIC	d	
EBCDIC	PC437	d	
PC437	IBM_EBCDIC	d	
88591	ROMAN8	d	HP LaserJet II
88591	VT220	d b e	
VT220	88591	d b e	
88592	646	d b e	
88593	646	d b e	
88594	646	d b e	
88595	646	d b e	
88597	646	d b e	
dgi	88591	d b e p	
88591	dgi	d b e p	

The *fromcodes* and *tocodes* 88591, 646 and 6937 correspond to the International Standards ISO 8859-1, ISO 646 and ISO 6937 respectively.

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
Characters are where possible, mapped to the closest approximation of that character but always without expansion, ie., 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 mapping are performed by the STREAMS-module in the TTY sub-system then such mappings will be transparent and the application will have no knowledge that these mappings take place.]

e best fit with expansion
Characters of the source code set are, where possible, 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 some 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 below, with the following fields:

fromcode	tocode	table	file	mode
(the above field names are not included in the database).				
88591	6937	88591.6937.b	pubfile	b
88591	6937	pubtable	88591.6937.d.t	d
646	646DE	togerman	/mydir/togerman	

Using the above database, the following converts the contents of files *mail1* and *mail2* from code set *88591* to *6937* using *b* mode and stores the results in file *mail.local.*

```
iconv -f 88591 -t 6937 -m b mail1 mail2 > mail.local
```

The following will accomplish the same result as above, as the *b* mode from code set *88591* to *6937* will be the first row found containing the correct match.

```
iconv -f 88591 -t 6937 mail1 mail2 > mail.local
```

FILES

`/usr/lib/kbd/iconv_data` default database

EXIT CODES

The exit status will be set with 0 upon successful completion, 1 otherwise.

SEE ALSO

`kbdcomp`(1M), `kbdset`(1), `kbdload`(1M), `kbdpipe`(1), `att_kbd`(7).

NAME

id – print the user name and ID, and group name and ID

SYNOPSIS

`id [ -g | -u ] [ -nra ]`

DESCRIPTION

`id` writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

Also, if there are any supplementary group affiliations, they will also be printed.

Options are:

`-g`	Output only the effective group ID. This may be followed by any supplementary group IDs that are different from the effective group ID.
`-u`	Output only the effective user ID.
`-r`	Used with `-g` or `-u`, output the real ID instead of the effective ID.
`-n`	Used with `-g` or `-u`, output the name instead of the numeric ID.
`-a`	The `-a` option reports all the groups to which the invoking user or process belongs.

EXAMPLES

```
$ id
uid=3000(intern) gid=1(other) groups=0(sys),5(mail)
$
```

Displays your user ID, username, group ID and group name on the standard output. Also, in this example, you will notice there are two other supplementary groups that are valid for this user.

SEE ALSO

groups(1), logname(1), getuid(2), setgroups(2), initgroups(3).

NAME

`idc` – interface description compiler

SYNOPSIS

`idc` [*description-file* ...]

where:

description-file	An `idl`(4) source file.

DESCRIPTION

`Idc` is a compiler for interface description files. The description file or files given on the command line are compiled into a special format for later use an input to `idi`(1). This special format allows `idi` to process the compiled files much faster than regular interface description files.

If no *description-file*s are given on the command line, standard input is read.

OUTPUT

The compiled description file is written to standard output.

DIAGNOSTICS

Descriptions of syntax errors in the *description-file*s are written to standard error.

SEE ALSO

`idi`(1), `idl`(4).

NAME

`idi` – interface description interpreter

SYNOPSIS

`idi` *program-name interface-name description-file* ... [*interface-driver-options* ...]

DESCRIPTION

`Idi` is an interpreter for interface description language files. The format of interface description files is described in `idl`(4).

The command line argument *program-name* is used in the title string for screens and in error or help messages. The *interface-name* argument is the name of a supported interface driver to use. This must be one of `ascii` or `motif`.

The *description-files* describe the hierarchy of menus and queries to be presented to the user. These files (or files included by them) must define a menu called `main`. This menu is used as the root of the hierarchy.

Any options beginning with "–" are *interface-driver-options* and are interpreted differently by each interface driver. For example, the Motif driver allows the standard X Window System toolkit options (see `X`(1)).

The interface driver presents the top level menu and allows you to navigate through the top level menu and all its sub-menus. Eventually, you may choose an operation to perform.

If no more information is needed for the operation (no queries), the operation is dispatched to the shell for execution. Otherwise, the queries are presented before the operation is dispatched. After the operation is complete, you may resume menu navigation.

ASCII DISPLAY DRIVER

The ASCII display driver presents a line-oriented interface suitable for use on any terminal or terminal emulator.

The following optional *interface-driver-options* are allowed:

`–m` *menu-name*
: Specify *menu-name* as the first menu to be shown.

`–o` *operation-name*
: Specify that the *operation-name* operation should be performed on *menu-name*. Note that when `–o` is used, `–m` must also be used.

`–s`
: Force the display driver into "sub-program" mode. In this mode, the user may exit the program by entering "`q`" at any menu prompt (exit code is 1) or by entering "^" at the top menu prompt (exit code is 0).

Menu Navigation

The items in a menu are presented as a numbered list of choices. To navigate through the menus, enter one of the following:

number Select the menu item numbered *number*.

name Select the menu item named *name*. The *name* may be a partial match, in upper or lower case.

name:*sub-name*...
: Select the menu item named *sub-name* from the *name* menu. Any number of *name*:*sub-name* constructs may be used. For example, the entry

```
options:verbosity:set
```

would select the "Options" menu, then the "Verbosity" menu, then the "Set" operation.

A `:` at the beginning of the entry indicates the root, or top level, menu.

`^`	Return to the previous menu.
`!`	Create a shell, according to the value of the user's `SHELL` environment variable.
`!`*command*	Execute the shell *command* and return control to `idi`.
`?`	Print a help message about this menu.
number`?`	Print a help message about menu item *number*.
`q`	Exit the program.

If the menu item selected is a menu, it becomes the new menu, and the sub-menus of the selected menu are displayed. If the menu item selected is an operation, the queries for the operation (if any) are displayed, and the operation is performed.

Menu item names which end in `->` are menus with sub-menus. Menu item names which end in `...` are operations with related queries. Menu item names with no suffix are operations with no related queries.

Queries

Many operations have corresponding queries which gather information before performing a command. The ASCII Display Driver does not display screens or query-groups, but displays only the queries under them.

At any query, you can enter any of these:

return	Accept the default value for the query.
`^`	Return to the previous query for this operation.
`<`	Restart the operation at the first query.
`!`	Create a shell, according to the value of the user's `SHELL` environment variable.
`!`*command*	Execute the shell *command* and return control to `idi`.
`c`	Cancel this operation and return to the menu.
`?`	Print a help message about this query.

Below is a description of how the ASCII Display Driver presents each of the four types of queries:

text query The driver displays text queries as follows:

```
prompt: [default]
```

To enter a new value, simply type the value.

boolean query
The driver displays bool queries as follows:

```
prompt? [default]
```

Enter either "yes" or "no" (or any abbreviation).

select query

The driver displays select queries as follows:

```
prompt: [default]
```

Enter `?` to see the list of possible values for the selectquery. In most cases, the possible values are numbered, and you may select a value by entering the number, or by entering the value itself (or any abbreviation of the value). If the possible values are numbered, your entry is first interpreted as the number of your selection; if the number is invalid, it is then interpreted as a value from the possible value list.

range query

The driver displays range queries as follows:

```
prompt: (lower-upper) [default]
```

Enter a number between the lower and upper limits.

Skill Levels

The current skill level, as indicated by the value of the `idl` variable, `SkillLevel`, affects certain aspects of the ASCII Display Driver.

The `Novice` and `Intermediate` skill levels are essentially the same. When the skill level is `Expert` several aspects of the interface are different:

* the prompts for menu navigation are shorter
* help messages for queries do not contain information about the valid characters for the query
* the confirmation message usually presented before operations are started is not given

MOTIF DISPLAY DRIVER

The OSF/Motif display driver presents a window-based interface to be used on devices running the X Window System and the OSF/Motif window manager, mwm(1).

Menu Navigation

The top level menu is presented in a Motif menu bar. Selecting a menu causes a pull-down menu which contains the sub-menus to appear.

Menu item names which end in `->` are menus with sub-menus. Menu item names which end in `...` are operations with related queries. Menu item names with no suffix are operations with no related queries.

Queries

The Motif driver presents screens as dialog boxes. Each querygroup of a screen is separated from the other querygroups by a horizontal separator bar.

At the bottom of each dialog box are four or more buttons. If there is only one screen, these buttons are shown: `Ok`, `Reset`, `Cancel`, and `Help`. If the operation has more than one screen, the first screen has these buttons: `Next`, `Reset`, `Cancel`, and `Help`; the middle screens have these buttons: `Next`, `Prev`, `Reset`, `Cancel`, and `Help`; and the last screen of a multi-screen operation has these buttons: `Ok`, `Prev`, `Reset`, `Cancel`, and `Help`.

The meanings of the buttons are as follows:

`Ok` Validate the values of all queries, and perform the operation.

`Next` Validate the values of all queries, and go on to the next screen for this operation.

`Prev` Return to the previous screen.

`Reset` Change the values of all queries in this screen to the values the queries had when the screen was first entered.

`Cancel` Terminate this operation.

`Help` Present the help message for the current operation. This is the same as hitting Function Key 1 (F1).

Below is a description of how the Motif Display Driver presents each of the four types of queries:

text query The driver displays text queries as follows:

```
         +---------------------------------+
prompt:  | default                         +
         +---------------------------------+
```

boolean query
The driver displays text queries as follows:

```
[ ] prompt
```

A darkened box is interpreted as affirmative, and a non-darkened box is interpreted as negative.

select query
The driver displays select queries in one of three ways: as a selection box, as a set of radio buttons, or as an option menu.

A selection box looks like this:

```
prompt:
+------------------------------------+-+
| selection one                      |^|
| default                            | |
| selection two                      | |
| selection three                    |v|
+------------------------------------+-+

Selection:
+--------------------------------------+
| default                              |
+--------------------------------------+
```

You may select one of the choices in the box (using the scroll bars if necessary), or you may enter a value into the text box.

A set of radio buttons looks like this:

```
prompt:
+--------------------------------+
| [ ] selection one              |
| [#] default                    |
| [ ] selection two              |
+--------------------------------+
```

The button for the selected item is darkened. Under some circumstances, more than one item may be selected.

An option menu looks like this:

```
         +-----------------+
prompt:  | default         |
         +-----------------+
```

Selecting the menu causes a list of choices to appear:

```
         +-----------------+
         | selection one   |
prompt:  | default         |
         | selection two   |
         | selection three |
         +-----------------+
```

range query
The driver displays range queries as follows:

```
         +---------+
prompt:  | default |
         +---------+

       ++
+------||--------------------------------+
+------||--------------------------------+
       ++
```

The sliding bar indicates the relative position of the current value in the range. You may enter a new value into the text area, or you may slide the bar to the new value.

Note that you may enter F1 when the keyboard focus is on a query to receive help for that query.

Skill Levels

The Motif Display Driver does not change the appearance based on your level of expertise.

Widget Names

This section lists the names of widgets used by the Motif driver and gives a brief description of each.

You may customize the appearance of `idi` widgets by modifying attributes of these widgets. See X(1) for a discussion of how to set widget attributes; see the manual entries for each widget class for the names of valid attributes.

As an example, in order to change the background color of help dialog boxes to be "wheat", put the following line into a resource file (such as `$HOME/.Xdefaults`):

```
Idi*helpBox*background: wheat
```

Widget Names

Widget Name	Widget Class	Description
mainWindow	XmMainWindow	Main window
menuBar	XmRowColumn	Main menu bar
logTextSW	XmScrolledWindow	Window for main text
logText	XmText	Main text area
mainDescription	XmLabelGadget	Description area
menuPullDown	XmRowColumn	Holds names of sub-menus
menuCascadeButton	XmCascadeButtonGadget	Button to show menu items
menuPushButton	XmPushButtonGadget	Button for operation
menuSeparator	XmSeparatorGadget	Separator between menu items
messageDialogShell	XmDialogShell	Holder for message boxes
confirmBox	XmMessageBox	Confirmation box
helpBox	XmMessageBox	Help box
unimpBox	XmMessageBox	Unimplemented feature box
errorBox	XmMessageBox	Error message box
warningBox	XmMessageBox	Warning box
okB	XmPushButtonGadget	Ok button in screenHolder
nextB	XmPushButtonGadget	Next button in screenHolder
prevB	XmPushButtonGadget	Prev button in screenHolder
resetB	XmPushButtonGadget	Reset button in screenHolder
cancelB	XmPushButtonGadget	Cancel button in screenHolder
helpB	XmPushButtonGadget	Help button in screenHolder
screenHolder_popup	XmDialogShell	Parent of screenHolder
screenHolder	XmForm	Container for querygroups
queryGroupSeparator	XmSeparatorGadget	Separator between querygroups
textAreaLabel	XmLabelGadget	Container for static text
querySeparator	XmSeparatorGadget	Separator between queries
booleanButton	XmToggleButton	Boolean query
rangeLabel	XmLabelGadget	Prompt in range query
rangeText	XmTextField	Text entry box in range query
rangeQuery	XmScale	Scale bar in range query
selectionBox	XmSelectionBox	Selection box
radioLabel	XmLabelGadget	Prompt for radio buttons
radioBox	XmRowColumn	Holder for radio buttons
radioButton	XmToggleButtonGadget	Each radio button
optionPullDown	XmRowColumn	Holder for option menu items
optionPushButton	XmPushButtonGadget	Button within option menu
optionMenu	XmRowColumn	Option menu with prompt

textLabel	XmLabelGadget	Prompt in text query
textText	XmText	Multi-line text entry box in text query
textText	XmTextField	Single line text entry box in text query
reportBox	XmForm	Holder for report output
reportClose	XmPushButtonGadget	Close button for reports
reportSep	XmSeparatorGadget	Separator after report text
reportTextSW	XmScrolledWindow	Window for text of report
reportText	XmText	Text of report

Additional Resources

The following additional resources are provided by the Motif driver to allow further modification of the driver's appearance.

Additional Resources

Name	Class	Value Type	Default
iconImage	IconImage	file	""

The additional resources are described below:

`iconImage` (class `IconImage`)

This is the name of a file containing a `X` bitmap to be used as the icon image for the program.

ENVIRONMENT

The `LANG` environment variable is checked [by `setlocale`(3C)] to determine the appropriate locale.

DIAGNOSTICS

Descriptions of syntax errors in the *description-file*s are written to standard error.

SEE ALSO

`idc`(1), `idl`(4), `X`(1).

NAME

`idi_tools: idi_confirm, idi_echo, idi_error, idi_log, idi_warning` – tools for use with the interface description interpreter

SYNOPSIS

`idi_confirm` [`-iny`] *string*
`idi_echo` [*string* ...]
`idi_error` [*string* ...]
`idi_log` [*string* ...]
`idi_warning` [*string* ...]

DESCRIPTION

The `idi_tools` are used by commands to communicate with `idi(1)`, the interface description interpreter. The `idi_tools` commands function in very similar ways, but are described separately below.

idi_confirm

The `idi_confirm` program presents a *string* using the appropriate `idi` interface mechanism. If `idi_confirm` is invoked as a child of an `idi` process, the interface of the `idi` program is used. Otherwise, the *string* is written to standard error and the user's response is read from standard input.

If the *string* is confirmed, `y` is written to the standard output; otherwise, `n` is written to the standard output.

The following options may be passed to `idi_confirm`:

`-i` Present the *string* only if the command is not a child of some `idi` process. If the command is a child of some `idi` process, write `y` to the standard output and exit.

This is for use when the calling program is invoked interactively.

`-n` Use negative ("no") as the default response.

`-y` Use affirmative ("yes") as the default response.

If neither `-n` or `-y` is given, affirmative ("yes") is used as the default response.

idi_echo

The `idi_echo` command displays report or listing output using the appropriate `idi` interface mechanism. For the `idi` ASCII display driver, the *string* is written to standard output. For the `idi` Motif display driver, the *string* is presented in a separate "report" window. If there is no parent `idi` process, the *string* is written to standard output.

This command uses `idi`'s builtin `:Echo` command.

idi_error

The `idi_error` command displays error output using the appropriate `idi` interface mechanism. For the `idi` ASCII display driver, the *string* is written to standard error with an "Error: " prefix. For the `idi` Motif display driver, the *string* is presented in a separate "error" window. If there is no parent `idi` process, the *string* is written to standard error.

This command should be used for messages describing conditions which cause the invoking program to exit with a non-zero exit code.

This command uses `idi`'s builtin `:Error` command.

idi_log

The `idi_log` command displays informational output using the appropriate `idi` interface mechanism. The *string* is also appended to `idi`'s log file. For the `idi` ASCII display driver, the *string* is written to standard error. For the `idi` Motif display driver, the *string* is written to the main window's text area. If there is no parent `idi` process, the *string* is written to standard error.

This command should be used for informative messages such as "The host 'mynewhost' has been added.". Any listing or report output should be displayed with the `idi_echo` command, not by `idi_log`.

This command uses `idi`'s builtin `:Log` command.

idi_warning

The `idi_warning` command displays warning messages using the appropriate `idi` interface mechanism. For the `idi` ASCII display driver, the *string* is written to standard error with a "Warning: " prefix. For the `idi` Motif display driver, the *string* is written to a separate "warning" window. If there is no parent `idi` process, the *string* is written to standard error.

This command should be used when the invoking program encounters an unusual or unexpected condition which does not affect the program's ability to complete its task. A command which invokes `idi_warning` is still expected to exit with a zero exit code (unlike commands which invoke `idi_error`).

This command uses `idi`'s builtin `:Warning` command.

Output

`Idi_confirm` may write the *string* to standard error (if there is no parent `idi` process), and writes either `y` or `n` to standard output.

`Idi_echo` may write the *string* to standard output (if there is no parent `idi` process). `Idi_error`, `idi_log`, and `idi_warning` may write the *string* to standard error (if there is no parent `idi` process).

EXAMPLE

The shell code below demonstrates how these tools might be used:

```
done=no
while [ $done = "no" ]
do
    confirmed=` idi_confirm "Is the tape mounted in ${tape}?" `
    case ${confirmed} in
        y)  done=yes ;;
        n)  stop=` idi_confirm -n "Quit?" `
            case ${stop} in
                y)  idi_error "Operation aborted."; exit 1  ;;
                n)  continue;;
            esac
            ;;
    esac
done

if cpio -ivtBc < ${tape} > ${file}
then
    : nothing
else
    idi_error "Unable to read ${tape}."
    exit 1
```

```
fi

if set_file_permissions ${file} root other 0644
then
    : nothing
else
    idi_warning "Unable to set permissions for ${file}."
fi

idi_echo "
Mode   Owner    Size  Modification Time     File Name
----   -----    ----  -----------------     ---------
`cat ${file}`"

idi_log "Read ${tape}."
```

SEE ALSO

idi(1), idl(4).

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.

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 options use the argument *n*. This argument should be a 1 or a 2 referring to either *file1* or *file2*, respectively. Options are:

-a*n*	In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
-e *s*	Replace empty output fields by string *s*.
-j*n m*	Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with 1.
-o *list*	Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
-t*c*	Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

International Features

join can process characters from supplementary code sets, as well as ASCII characters.

Options:

-e *s*	The string *s* to be replaced can contain supplementary characters.
-t *c*	The separator *c* can be a character from the supplementary code sets.

EXAMPLE

The following command lines sort the passwd and group files on the key fields and join the sorted passwd and group files, matching the numeric group ID, and outputting the login name, the group name, and the login directory.

```
sort +3 -4 -t: /etc/passwd >/tmp/passwd.sort

sort +2 -3 -t: /etc/group >/tmp/group.sort

join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /tmp/passwd.sort /tmp/group.sort
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

With default field separation, the collating sequence is that of `sort -b`; with `-t`, the sequence is that of a plain sort.

The conventions of `join`, `sort`, `comm`, `uniq` and `awk`(1) are incongruous.

Numeric filenames may cause conflict when the `-o` option is used right before listing filenames.

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 `att_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 pre-pended 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.

The essentially similar `iconv`(1) program is considerably faster than `kbd_pipe`.

FILES

`/usr/lib/kbd`	directory containing system standard table files

SEE ALSO

`kbdload`(1M), `kbdset`(1), `att_kbd`(7).

NAME

kbdset – attach to att_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 att_kbd *STREAMS* module [see kbdcomp(1M) and att_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 att_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 `att_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.

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), `att_kbd`(7).

NOTES

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.

NAME

`keylogin` – decrypt and store secret key

SYNOPSIS

`keylogin`

DESCRIPTION

NOTE: Secure RPC using DES Authentication is an additional feature that must be purchased separately from the DG/UX™ ONC™/NFS® package. You must have this feature to use the command described in this manual page.

`keylogin` prompts the user for their login 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 services, such as NFS.

Normally, `login`(1) does this work when the user logs onto the system, but running `keylogin` may be necessary if the user did not type a password to `login`(1).

SEE ALSO

`chkey`(1), `login`(1), `keyserv`(1M), `newkey`(1M), `publickey`(4).

NAME

`kill` – terminate a process by default

SYNOPSIS

`kill` [-*signal*] *pid* ...
`kill` -*signal* -p*gid* ...
`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 signalled 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) in the *Programmer's Reference for the DG/UX System*.

NAME

ksh, rksh – KornShell, a standard/restricted command and programming language

SYNOPSIS

ksh [±aefhikmnprstuvx] [±o *option*] ... [-c *string*] [*arg* ...]
rksh [±aefhikmnprstuvx] [±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 (i.e., 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.

`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 file-name generation (see *File Name Generation* below).

`if` *list* `;then` *list* [`elif` *list* `;then` *list*] ... [`;else` *list*] `;fi`

The *list* following `if` is executed and, if it returns a zero exit status, the *list* following the first `then` is executed. Otherwise, the *list* following `elif` is executed and, if its value is zero, the *list* following the next `then` is executed. Failing that, the `else` *list* is executed. If no `else` *list* or `then` *list* is executed, 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 word*s and must be typed 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). The `{` must be followed by a space.

`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 path-name 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.

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.

`${`*parameter*`%`*pattern*`}`
`${`*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.

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. This path is searched when a function with the `-u` attribute is referenced and when a command is not found. If an executable file is found, then it is read and executed in the current environment.

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).

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 matches the pattern `*r*sh`, 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, 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* (i.e., 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 (i.e., 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 (i.e. *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`. Function can be undefined with the `-f` option of the `unset` special command.

Ordinarily, functions are unset when the shell executes a shell script. The `-xf` option of the `typeset` command allows a function to be exported to scripts that are executed without 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.

The INT and QUIT signals for an invoked command are ignored if the command is followed by `&` and job `monitor` option is not active. Otherwise, signals have the values inherited by the shell from its parent (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. 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 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 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 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.

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 `$a`.
[*count*]`c`*motion* `c`[*count*]*motion*	Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is `c`, the entire line will be deleted and input mode entered.
`C`	Delete the current character through the end of line and enter input mode. Equivalent to `c$`.
`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.
#	Sends the line after inserting a `#` in front of the line. Useful for causing the current line to be inserted in the history without being executed.
=	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*. If *n* is omitted then the exit status is that of the last command executed. An end-of-file will also cause the shell to exit 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 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.

`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. 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*, etc., 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 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* to return to the invoking script with the return status specified by *n*. 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` [±`aefhkmnpstuvx`] [±`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. 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.

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 the 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.

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, i. e., 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. 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 `ksh`, 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 (i.e., `/usr/rbin`) that can be safely invoked by `rksh`.

EXIT CODES

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), `paste`(1), `stty`(1), `test`(1), `umask`(1), and `vi`(1).
`dup`(2), `exec`(2), `fork`(2), `ioctl`(2), `lseek`(2), `pipe`(2), `signal`(2), `umask`(2), `ulimit`(2), `wait`(2), and `rand`(3C) in the *Programmer's Reference for the DG/UX System*.
`newgrp`(1M), `a.out`(4), `profile`(4), and `environ`(4) in the *System Manager's Reference for the DG/UX System*.

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

`last` – indicate last user or terminal logins

SYNOPSIS

`last` [`-n` *number* | *-number*] [`-f` *filename*] [*name* | *tty*] ...

DESCRIPTION

The `last` command looks in the `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 tty10`. 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 the `wtmp` file. 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 `/etc/wtmp`.

FILES

`/etc/wtmp`	accounting file

SEE ALSO

`who`(1).
`utmp`(4) in the *System Manager's Reference for the DG/UX System*.

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.

EXAMPLES

This example is a shell program that gathers information about users.

```
echo "ENTER YOUR NAME"
NAME=`line`
echo "ENTER YOUR DEPARTMENT NUMBER:"
DEPT=`line`
echo "$NAME $DEPT" >> personnel
```

First, the program queries you for your name. Because line is surrounded by grave accents, the program executes it, and line reads the name and assigns the name to the variable "NAME". Then the program asks for your department and office extension. The program assigns these values to "DEPT" The last line appends "NAME"and "DEPT" to the file "personnel".

SEE ALSO

sh(1).
read(2) in the *Programmer's Reference for the DG/UX System*

NAME

`listusers` – list user login information

SYNOPSIS

`listusers` [`-g` *groups*] [`-l` *logins*]

DESCRIPTION

Executed without any options, this command lists all user logins sorted by login. The output shows the login ID and the account field value 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.

SEE ALSO

`groups`(1), `id`(1), `logname`(1), `passwd`(4).

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 only be listed once, even if they belong to more than one of the selected groups.

NAME

`ln` – link files

SYNOPSIS

`ln [ -f ] [ -n ] [ -s ]` *file1* [*file2...*] *target*

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 *target*.

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.

The following options are recognized:

`-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.

`-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 `-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.

NOTES

The `ln` command cannot be used to create a hard link to a file that is in a control point directory.

SEE ALSO

`chmod`(1), `cp`(1), `mv`(1), `rm`(1), `link`(2), `readlink`(2), `stat`(2), `symlink`(2).

NAME

locate – identify a command using keywords

SYNOPSIS

[help] locate
[help] locate [*keyword1* [*keyword2*] ...]

DESCRIPTION

The locate command is part of the DG/UX system Help Facility and provides on-line assistance with identifying DG/UX system commands.

If locate is entered without arguments, the initial locate screen is displayed. A user may enter keywords and receive a list of DG/UX system commands whose functional attributes match those in the keyword list, or the user may quit and return to the shell by typing q. For example, if you wish to print the contents of a file, enter k for keywords then the keywords print and file. The locate command will then print the names of all commands related to the keywords.

Keywords may also be entered directly from the shell as arguments to the locate command (see above). In this case, the initial screen is not displayed, and the resulting command list is printed.

More detailed information on a command in the list produced by locate can be obtained by entering d (description), e (example), or o (options).

From any screen in the Help Facility, a user may execute a command via the shell (sh(1)) by typing a ! followed by the command to be executed. The screen will be redrawn if the command that was executed was entered at the initial locate screen. If entered at any other menu screen, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen cleared before printing the data (non-scrolling), set the shell environment variable SCROLL to no. If you use the Bourne shell, sh(1), do this by adding the following line to your .profile file (see profile(4)):

```
SCROLL=no ; export SCROLL
```

If you use the C shell, csh(1), do this by adding the following line to your .login file:

```
setenv SCROLL no
```

If you later decide that you want scrolling, set SCROLL to yes instead.

Information on each of the Help Facility commands (starter, locate, usage, glossary, and help) is located in their respective manual pages.

EXAMPLES

```
$ locate
```

Prints a menu screen that prompts you to enter a set of keywords related to the command you want to find.

```
$ locate print file format
```

Prints a screen containing a list of DG/UX system commands whose functions are related to the keywords print file and format. It also prints a menu that allows you to enter new keywords or request detailed information about any command.

SEE ALSO

glossary(1), help(1), sh(1), csh(1), starter(1), usage(1).
term(5) in the *Programmer's Reference for the DG/UX System*.

NOTES

If the shell variable TERM [see sh(1)] is not set in the user's .profile file, TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to term(5).

NAME

logger – make entries in the system log

SYNOPSIS

`logger [ -t` *tag* `] [ -p` *pri* `] [ -i ] [ -f` *file* `] [` *msg* `... ]`

where:

tag	Tag with which to mark every line in the log.
pri	Message priority (see description of `-p` option below).
file	Name of file containing lines to be logged.
msg	The message to log; the default is to log the file specified by `-f`, or standard input.

DESCRIPTION

`Logger` provides a program interface to the `syslog`(3C) system log module.

You can specify a message on the command line, or you can specify a file containing lines that you want logged.

You set the location of the system log file in `syslog.conf`(5).

Options

`-t`	Mark every line in the log with the specified *tag*.
`-p`	Enter the message with the specified priority. The priority may be specified numerically or as a "facility.level" pair. For example, `-p local3.info` logs the message(s) as *info*rmational level in the `local3` facility. The default is `user.notice`.
`-i`	Log the process id of the logger process with each line.
`-f`	Log the specified file.

EXAMPLES

```
logger System rebooted

logger -p local0.notice -t HOSTIDM
```

SEE ALSO

`syslogd`(1M), `syslog`(3C), `syslog.conf`(5).

NAME

login − sign on

SYNOPSIS

login [*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 will be invoked by the system when a connection is first established.

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.

If you make any mistake in the login procedure, the message

```
Login incorrect
```

is printed and a new login prompt will appear.

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, accounting files are updated, the /etc/profile script is executed, /etc/motd is printed, the user-ID, group-ID, supplementary group list, working directory, and command interpreter (usually sh) are initialized. If the initialized command interpreter is sh, login instructs sh to perform the procedure /etc/profile. In addition, if the file .profile exists in the working directory, sh executes it as well. in the working directory is executed, if it exists. These specifications are found in the /etc/passwd file entry for the user. 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 a chroot(2) is done to the directory named in the directory field of the entry making it the root directory. At that point login is re-executed at the new level which must have its own root structure, including /etc/login and /etc/passwd.

The basic *environment* 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

```
Ln=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.

The system administrator can modify the behavior of `login` by setting variables in the `/etc/default/login` file. The following variables are available:

`ALTSHELL`	If set to "YES" the `SHELL` environment variable containing the pathname of the user's shell will be declared as part of the basic initial environment.
`CONSOLE`	If set, superuser login is allowed only on the terminal specified. E.g., "`CONSOLE=/dev/console`" restricts superuser login to the console. If not set, no restrictions are placed on superuser login.
`HZ`	Default value for the HZ (hertz) environment variable. If not set, the value of HZ defaults to 100.
`PASSREQ`	If set to "YES" a password is required for all non-superusers on the system. If a new user account is added with no password, login will prompt for a password the first time the user attempts to log in.
`PATH`	Default value of `PATH` environment variable for all non-superuser logins on the system. If not set, the default is "`/usr/bin`".
`SUPATH`	Default value of `PATH` environment variable for all superuser logins on the system. If not set, the default is "`/sbin:/usr/sbin:/usr/bin:/etc`".
`TIMEOUT`	Maximum amount of time in seconds to wait on a successful login attempt before disconnecting. The maximum value allowed is 900 (15 minutes). If not set, timeout period defaults to 60 seconds. Setting TIMEOUT 0 disables the timeout feature.
`TIMEZONE`	Default value for the TZ (time zone) environment variable. If not set, the value of TZ defaults to "`EST5EDT`".
`ULIMIT`	Maximum size allowed for user files (in blocks). If ULIMIT is not set, no file size limit is enforced.
`UMASK`	Default umask for system users. If UMASK is not set the default umask will be 022.

FILES

`/var/adm/utmp`	accounting
`/var/adm/wtmp`	accounting
`/var/mail/`*your-name*	mailbox for user *your-name*
`/etc/default/login`	login system-wide default settings
`/etc/motd`	message-of-the-day
`/etc/passwd`	password file
`/etc/profile`	system profile
`.profile`	user's login profile

DIAGNOSTICS

`login incorrect`
This message appears if the user name or the password cannot be matched.

`No shell`, `cannot open password file`, or `no directory`
If these messages appear consult your system administrator.

`No utmp entry.  You must exec login from the lowest level sh.`
This message appears 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.

`Cannot open /dev/tty.`
This message appears if `login` is unable to open /dev/tty to read the password.

SEE ALSO

`mail`(1), `newgrp`(1M), `sh`(1), `su`(1M).
`loginlog`(4), `passwd`(4), `profile`(4), `environ`(5) in the *Programmer's Reference for the DG/UX System*.

NAME

`logname` – get login name

SYNOPSIS

```
logname
```

DESCRIPTION

`Logname` returns the contents of the environment variable `$LOGNAME`, the name under which you logged in. It is set when you log into the system, but can be reset. It is simply an environment variable; it does not affect the system's knowledge of you through your effective user ID.

EXAMPLES

```
$ logname
billcat
```

Returns the name of the user as known by the system at login time. If `$LOGNAME` (the shell variable) is not defined `logname` will still function.

```
$ logname > lognm
$ cat lognm
billcat
```

Puts the user name in the new file `lognm` and displays the contents of `lognm`.

FILES

`/etc/profile.`

SEE ALSO

`env`(1), `id`(1), `login`(1).
`logname`(3X), `environ`(5) in the *Programmer's Reference for the DG/UX System*

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-ID-list* [*printers*]

DESCRIPTION

The first form of the `lp` command arranges for the named *files* and associated information (collectively called a *request*) to be printed. If no file names are specified on the command line, the standard input is assumed. The standard input may be specified along with named *files* on the 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 LP print service associates a unique *request-ID* with each request and displays 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, and `lpstat`(1) for information about checking the status of a print request.]

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).

The `cancel` command allows users to cancel print requests previously sent with the `lp` command. The first form of `cancel` permits cancellation of requests based on their *request-ID*. The second form of cancel permits cancellation of requests based on the *login-ID* of their owner.

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 specified in any order. The following options are available for `lp`:

`-c` Make copies of the *files* before printing. 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. If *dest* is `any`, then the request will be printed on any printer which can handle it. Under certain conditions (availability 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)]. If the printer is located on a remote system that has a pre-5.4 version of the LP scheduler, the **lp** command submits the request through a remote shell using **remsh**(1C). Your system and the remote system must be configured to permit execution of a **remsh** command (by having your system's name in the remote system's **/etc/hosts.equiv** file).

`-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 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 is 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 one `-o` keyletter enclosed in double quotes (that is, `-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 may not 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 may not 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 may not 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 characters per inch) or `elite` (representing 12 characters per inch), or it can be `compressed` (representing as many characters as a printer can handle). There is no standard number of characters per inch for all printers; see the Terminfo database [`terminfo`(4)] for the default character pitch for your printer.

  This option may not be used with the `-f` option.

`stty='`*stty-option-list*`'`
: A list of options valid for the `stty` command; enclose the list with single 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. A priority limit may be assigned to individual users 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. Enclose *title* in quotes if it contains blanks.

`-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 may 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. The first form allows a user to specify one or more *request-IDs* of print jobs to be canceled. Alternatively, the user can specify one or more *printers*, on which only the currently printing job will be canceled.

The second form of `cancel` permits a user to cancel all of his or her own jobs on all printers. In this form the *printers* option can be used to restrict the printers on which the user's jobs will be canceled. Note that in this form, when the *printers* option is used, all jobs queued for those printers will be canceled. A printer class is not a valid argument.

Users without special privileges can cancel only requests associated with their own login IDs. The system administrator can cancel jobs submitted by any user. The *login-ID-list* must be enclosed in quotes if it contains blanks.

EXAMPLE

To print on a PostScript printer named `pslaser` an 8-bit-character document named `europe1` coded in ISO standard 8859.1 format:

```
lp -d pslaser -S iso-88591 europe1
```

FILES

/var/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1), postprint(1).
accept(1M), lpadmin(1M), lpfilter(1M), lpforms(1M), lpsched(1M), lpsystem(1M), lpusers(1M) in the *System Manager's Reference*.
terminfo(4) in the *Programmer's Reference*.

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.

NAME

`lpq` – examine the spool queue

SYNOPSIS

`lpq` [+[*n*]] [`-l`] [`-P`*printer*] [*job #* ...] [*user* ...]

DESCRIPTION

`lpq` examines the spooling area used by `lpd`(1M) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user. `lpq` invoked without any arguments reports on any jobs currently in the queue. A `-P` flag may be used to specify a particular printer, otherwise the default line printer is used (or the value of the PRINTER variable in the environment). If a + argument is supplied, `lpq` displays the spool queue until it empties. Supplying a number immediately after the + sign indicates that `lpq` should sleep *n* seconds in between scans of the queue. All other arguments supplied are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (i.e. invocation of `lpr`(1)) `lpq` reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to `lprm`(1) for removing a specific job), and the total size in bytes. The `-l` option causes information about each of the files comprising the job to be printed. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when `lpr`(1) is used as a sink in a pipeline) in which case the file is indicated as "(standard input)".

If `lpq` warns that there is no daemon present (i.e. due to some malfunction), the `lpc`(1M) command can be used to restart the printer daemon.

FILES

`/etc/termcap`	for manipulating the screen for repeated display
`/etc/printcap`	to determine printer characteristics
`/usr/spool/*`	the spooling directory, as determined from printcap
`/usr/spool/*/cf*`	control files specifying jobs
`/usr/spool/*/lock`	the lock file to obtain the currently active job

DIAGNOSTICS

Unable to open various files. The lock file being malformed. Garbage files when there is no daemon active, but files in the spooling directory.

SEE ALSO

`lpc`(1M), `lpd`(1M), `lpr`(1), `lprm`(1).

NOTES

Due to the dynamic nature of the information in the spooling directory `lpq` may report unreliably. Output formatting is sensitive to the line length of the terminal; this can results in widely spaced columns.

NAME

lpr – send print requests to a line printer spooler

SYNOPSIS

lpr [-P*printer*] [-#*num*] [-C *class*] [-J *job*] [-T *title*] [-i [*numcols*]] [-w*num*] [-plrmhs] [*name* ...]

DESCRIPTION

Lpr uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed. The -P option may be used to force output to a specific printer. Normally, the default printer is used (site dependent), or the value of the environment variable PRINTER is used.

The following single letter options are used to 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(1) to format the files (equivalent to print).

-l Use a filter which allows control characters to be printed and suppresses page breaks.

The remaining single letter options have the following meaning.

-r Remove the file upon completion of spooling or upon completion of printing (with the -s option).

-m Send mail upon completion.

-h Suppress the printing of the burst page.

-s Use symbolic links. Usually files are copied to the spool directory.

The -C option takes the following argument as a job classification for use on the burst page. For example,

```
lpr -C EECS foo.c
```

causes the system name (the name returned by hostname(1)) to be replaced on the burst page by EECS, and the file foo.c to be printed.

The -J option takes the following argument as the job name to print on the burst page. Normally, the first file's name is used.

The -T option uses the next argument as the title used by pr(1) instead of the file name.

To get multiple copies of output, use the -#*num* option, where *num* is the number of copies desired of each file named. For example,

```
lpr -#3 foo.c bar.c more.c
```

would result in 3 copies of the file foo.c, followed by 3 copies of the file bar.c, etc. On the other hand,

```
cat foo.c bar.c more.c | lpr -#3
```

will give three copies of the concatenation of the files.

The -i option causes the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.

The -w option takes the immediately following number to be the page width for pr.

The `-s` option will use `symlink`(2) to link data files rather than trying to copy them so large files can be printed. This means the files should not be modified or removed until they have been printed.

FILES

`/etc/passwd`	personal identification
`/etc/printcap`	printer capabilities data base
`/usr/lib/lpd*`	line printer daemons
`/usr/spool/*`	directories used for spooling
`/usr/spool/*/cf*`	daemon control files
`/usr/spool/*/df*`	data files specified in "cf" files
`/usr/spool/*/tf*`	temporary copies of "cf" files

DIAGNOSTICS

If you try to spool too large a file, it will be truncated. *If a user other than root prints a file and spooling is disabled,* `lpr` will print a message saying so and will not put jobs in the queue. If a connection to `lpd` on the local machine cannot be made, `lpr` will say that the daemon cannot be started. Diagnostics may be printed in the daemon's log file regarding missing spool files by `lpd`.

SEE ALSO

`lpc`(1M), `lpd`(1M), `lpq`(1), `lprm`(1), `pr`(1), `symlink`(2), `printcap`(5).

NOTES

Fonts for `troff` and `tex` reside on the host with the printer. It is currently not possible to use local font libraries.

NAME

lprm – remove jobs from the line printer spooling queue

SYNOPSIS

lprm [-P*printer*] [–] [*job #* ...] [*user* ...]

DESCRIPTION

Lprm will remove a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using lprm is normally the only method by which a user may remove a job.

Lprm without any arguments will delete the currently active job if it is owned by the user who invoked lprm.

If the – flag is specified, lprm will remove all jobs which a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the lpr command was invoked.

Specifying a user's name, or list of user names, will cause lprm to attempt to remove any jobs queued belonging to that user (or users). This form of invoking lprm is useful only to the super-user.

A user may dequeue an individual job by specifying its job number. This number may be obtained from the lpq(1) program, e.g.

```
% lpq -l

1st: ken                   [job #013ucbarpa]
   (standard input)        100 bytes
% lprm 13
```

Lprm will announce the names of any files it removes and is silent if there are no jobs in the queue which match the request list.

Lprm will kill off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The -P option may be usd to specify the queue associated with a specific printer (otherwise the default printer, or the value of the PRINTER variable in the environment is used).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current spooler and the job number of the currently active job

DIAGNOSTICS

"Permission denied" if the user tries to remove files other than his own.

SEE ALSO

lpd(1M), lpq(1), lpr(1).

NOTES

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

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 the user's 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 enclosed in quotes. For example:

```
-u "user1, user2, user3"
```

Specifying `all` after any keyletter that takes *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*. The keyletter `-o` may be omitted.

`-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.

`-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:

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

`-v` [*list*]
Reports the names of printers and the pathnames of the devices associated with them or remote system names for network printers: *list* is a list of printer names.

FILES

`/var/spool/lp/*`
`/etc/lp/*`

SEE ALSO

`enable(1)`, `lp(1)`.

NAME

`lptermprinter` – start printer session with 40014A Terminal Server

SYNOPSIS

`/usr/lib/lptermprinter -h` *hostname* [*filename*]

where:

hostname	Host name of a printer connected to a DG model 40014A Terminal Server
filename	Name of file from which to read input; default = standard input

DESCRIPTION

The `lptermprinter` program initiates a session with the specified host.

Refer to the "40014A Terminal Server / AViiON Release Notice" for further information.

FILES

`/usr/lib/lptermprinter`

SEE ALSO

`termprinter`(1).

NAME

ls – list contents of directory

SYNOPSIS

ls [-RadLCxmlnogrtucpFbqisf1%] [*names*]

ls [-lpF [%]] [*names*]

DESCRIPTION

For each directory argument, ls lists the contents of the directory; for each file argument, ls repeats its name and any other information requested. The output is sorted 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-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.
`-%`	When the `-F` or `-p` options are also given, put a percent (%) after each filename if that file is a control point directory. When the `-l` option is also given, put a percent (%) in the first character of the mode for each file that is a control point directory (instead of *d*). This option is useful only when used in combination with `-l`, `-p`, or `-F`.

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 (a.k.a. "named pipe") special file;
`-`	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 `long` 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 `l`ocking. 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.

International Features

`ls` can process directory names 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.

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

Unprintable characters in file names may confuse the columnar output options.

The total block count will be incorrect if there are hard links among the files.

NAME

`machid`: `dghost`, `m68k`, `m88k`, `i386`, `pdp11`, `u3b`, `u3b5`, `vax` – provide truth value about your processor type

SYNOPSIS

```
dghost

m68k

m88k

i386

pdp11

u3b

u3b5

vax
```

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

`dghost`	True if you are on a Data General MV-series computer.
`m68k`	True if you are on an M68000-based computer.
`m88k`	True if you are on an M88000-based computer.
`i386`	True if you are on a 386-based Data General computer.
`pdp11`	True if you are on a PDP-11/45 or PDP-11/70.
`u3b`	True if you are on a 3B 20S computer.
`u3b5`	True if you are on a 3B 5 computer.
`vax`	True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (non-zero) value. These commands are often used within `make`(1) makefiles and shell procedures to increase portability.

EXAMPLES

```
$ m88k
```

An exit code of 0 is returned if the statement is true.

SEE ALSO

`make`(1), `sh`(1), `test`(1), `true`(1).

NAME

`mail`, `rmail` – read mail or send mail to users

SYNOPSIS

Sending Mail

`mail` [`-tw`] [`-m` *message_type*] *recipient* . . .

`rmail` [`-tw`] [`-m` *message_type*] *recipient* . . .

Reading Mail

`mail` [`-ehpPqr`] [`-f` *file*]

Forwarding Mail

`mail -F` *recipient* . . .

Debugging

`mail` [`-x`*debug_level*] [*other_mail_options*] *recipient* . . .

`mail -T` *mailsurr_file recipient* . . .

DESCRIPTION

A *recipient* is usually a user name recognized by `login`(1). When *recipients* are named, `mail` assumes a message is being sent (except in the case of the `-F` option). It reads from the standard input up to an end-of-file (Ctrl-D) or, if reading from a terminal device, until it reads a line consisting of just a period. When either of those indicators is received, `mail` adds the *letter* to the *mailfile* for each *recipient*.

A *letter* is composed of some *header lines* followed by a blank line followed by the *message content*. The *header lines* section of the letter consists of one or more UNIX postmarks:

> `From` *sender date_and_time* [`remote from` *remote_system_name*]

followed by one or more standardized message header lines of the form:

> *keyword-name*`:` [*printable text*]

where *keyword-name* is comprised of any printable, non-white-space, characters other than colon (':'). A `Content-Length:` header line, indicating the number of bytes in the *message content* will always be present. A `Content-Type:` header line that describes the type of the *message content* (such as text, binary, multipart, etc.) will always be present unless the letter consists of only header lines with no message content. Header lines may be continued on the following line if that line starts with white space.

International Features

`mail` enables the retrieval, editing and processing of mail messages which include characters from supplementary code sets. Supplementary code set characters can be used in the *subject* line.

Sending Mail

The following command-line arguments affect SENDING mail:

`-m`	causes a `Message-Type:` line to be added to the message header with the value of *message_type*.
`-t`	causes a `To:` line to be added to the message header for each of the intended recipients.
`-w`	causes a letter to be sent to a remote recipient without waiting for the completion of the remote transfer program.

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If `mail` is interrupted during input, the message is saved in the file `dead.letter` to allow editing and resending. `dead.letter` is always appended to, thus preserving any previous contents. The initial attempt to append to (or create) `dead.letter` will be in the current directory. If this fails, `dead.letter` will be appended to (or created in) the user's login directory. If the second attempt also fails, no `dead.letter` processing will be done.

`rmail` only permits the sending of mail; `uucp`(1C) uses `rmail` as a security precaution. Any application programs that generate mail messages should be sure to invoke `rmail` rather than `mail` for message transport and/or delivery.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. There are numerous ways to address mail to recipients on remote systems depending on the transport mechanisms available to the local system. The two most prevalent addressing schemes are UUCP-style and Domain-style. With UUCP-style addressing, remote recipients are specified by prefixing the recipient name with the remote system name and an exclamation point (such as sysa!user). A series of system names separated by exclamation points can be used to direct a letter through an extended network (such as `sysa!sysb!sysc!user`). With Domain-style addressing, remote recipients are specified by appending an '@' and domain (and possibly sub-domain) information to the recipient name (such as `user@sf.att.com`). (The local system administrator should be consulted for details on which addressing conventions are available on the local system.)

Reading Mail

The following command-line arguments affect READING mail:

`-e`	causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
`-h`	causes a window of headers to be initially displayed rather than the latest message. The display is followed by the '?' prompt.
`-p`	causes all messages to be printed without prompting for disposition.
`-P`	causes all messages to be printed with *all* header lines displayed, rather than the default selective header line display.
`-q`	causes `mail` to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.
`-r`	causes messages to be printed in first-in, first-out order.
`-f` *file*	causes `mail` to use *file* (such as `mbox`) instead of the default *mailfile*.

`mail`, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. The default mode for printing messages is to display only those header lines of immediate interest. These include, but are not limited to, the UNIX `From` and `>From` postmarks, `From:`, `Date:`, `Subject:`, and `Content-Length:` header lines, and any recipient header lines such as `To:`, `Cc:`, `Bcc:`, etc. After the header lines have been displayed, `mail` will display the contents (body) of the message only if it contains no unprintable characters. Otherwise, `mail` will issue a warning statement about the message having binary content and `not` display the content. (This may be overridden via the `p` command. See below.)

For each message, the user is prompted with a `?`, and a line is read from the standard input. The following commands are available to determine the disposition of the message:

#	Print the number of the current message.
-	Print previous message.
<new-line>, `+`, or `n`	Print the next message.

`!`*command*	Escape to the shell to do *command*.
`a`	Print message that arrived during the `mail` session.
`d`, or `dp`	Delete the current message and print the next message.
`d` *n*	Delete message number *n*. Do not go on to next message.
`dq`	Delete message and quit `mail`.
`h`	Display a window of headers around current message.
`h` *n*	Display a window of headers around message number *n*.
`h a`	Display headers of all messages in the user's *mailfile*.
`h d`	Display headers of messages scheduled for deletion.
`m` [*persons*]	Mail (and delete) the current message to the named *person*(*s*).
n	Print message number *n*.
`p`	Print current message again, overriding any indications of binary (that is, unprintable) content.
`P`	Override default brief mode and print current message again, displaying all header lines.
`q`, or Ctrl-D	Put undeleted mail back in the *mailfile* and quit `mail`.
`r` [*users*]	Reply to the sender, and other *user(s)*, then delete the message.
`s` [*files*]	Save message in the named *file*(*s*) (`mbox` is default) and delete the message.
`u` [*n*]	Undelete message number *n* (default is last read).
`w` [*files*]	Save message contents, without any header lines, in the named *files* (`mbox` is default) and delete the message.
`x`	Put all mail back in the *mailfile* unchanged and exit `mail`.
`y` [*files*]	Same as save.
`?`	Print a command summary.

When a user logs in, the presence of mail, if any, is usually indicated. Also, notification is made if new mail arrives while using `mail`.

The permissions of *mailfile* may be manipulated using `chmod` in two ways to alter the function of `mail`. The other permissions of the file may be read-write (0666), read-only (0664), or neither read nor write (0660) to allow different levels of privacy. If changed to other than the default (mode 0660), the file will be preserved even when empty to perpetuate the desired permissions. (The administrator may override this file preservation using the `DEL_EMPTY_MAILFILE` option of `mailcnfg`.)

The group id of the mailfile must be `mail` to allow new messages to be delivered, and the mailfile must be writable by group `mail`.

Forwarding Mail

The following command-line argument affects FORWARDING of mail:

`-F` *recipients*
Causes all incoming mail to be forwarded to *recipients*. The mailbox must be empty.

The `-F` option causes the *mailfile* to contain a first line of:

```
        Forward to recipient. . .
```

Thereafter, all mail sent to the owner of the *mailfile* will be forwarded to each *recipient*.

An `Auto-Forwarded-From: ...` line will be added to the forwarded message's header. This is especially useful in a multi-machine environment to forward all a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded.

Installation and removal of forwarding is done with the `-F` invocation option. To forward all your mail to `systema!user` enter:

```
        mail -F systema!user
```

To forward to more than one recipient enter:

```
        mail -F "user1,user2@att.com,systemc!systemd!user3"
```

Note that when more than one recipient is specified, the entire list should be enclosed in double quotes so that it may all be interpreted as the operand of the `-F` option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

If the first character of any forwarded-to recipient name is the pipe symbol ('|'), the remainder of the line will be interpreted as a command to pipe the current mail message to. The command, known as a *Personal Surrogate*, will be executed in the environment of the recipient of the message (that is, basename of the *mailfile*). For example, if the mailfile is `/var/mail/foo`, `foo` will be looked up in `/etc/passwd` to determine the correct userID, groupID, and `HOME` directory. The command's environment will be set to contain only `HOME`, `LOGNAME`, `TZ`, `PATH` (= `/usr/usr/bin:`), and `SHELL` (= `/usr/bin/sh`), and the command will execute in the recipient's `HOME` directory. If the message recipient cannot be found in `/etc/passwd`, the command will not be executed and a non-delivery notification with appropriate diagnostics will be sent to the message's originator.

After the pipe symbol, escaped double quotes should be used to have strings with embedded white space be considered as single arguments to the command being executed. No shell syntax or metacharacters may be used unless the command specified is `/usr/bin/sh`. For example,

```
        mail -F "|/bin/sh -c \"shell_command_line\""
```

will work, but is not advised since using double quotes and backslashes within the shell_command_line is difficult to do correctly and becomes tedious very quickly.

Certain %keywords are allowed within the piped-to command specification and will be textually substituted for *before* the command line is executed.

`%R`	Return path to the message originator.
`%c`	Value of the `Content-Type:` header line if present.
`%S`	Value of the `Subject:` header line if present.

If the command being piped to exits with any non-zero value, `mail` will assume that message delivery failed and will generate a non-delivery notification to the message's originator. It is allowable to forward mail to other recipients and pipe it to a command, as in

```
mail -F "carol,joe,|myvacationprog %R"
```

Two UNIX System facilities that use the forwarding of messages to commands are `notify`(1), which causes asynchronous notification of new mail, and `vacation`(1), which provides an auto-answer capability for messages when the recipient will be unavailable for an extended period of time.

To remove forwarding enter:

```
mail -F ""
```

The pair of double quotes is mandatory to set a NULL argument for the -F option.

In order for forwarding to work properly the *mailfile* should have `mail` as group ID, and the group permission should be read-write.

`mail` will exit with a return code of `0` if forwarding was successfully installed or removed.

Debugging

The following command-line arguments cause `mail` to provide DEBUGGING information:

`-T` *mailsurr_file*	causes `mail` to display how it will parse and interpret the `mailsurr` file.
`-x`*debug_level*	causes `mail` to create a trace file containing debugging information.

The `-T` option requires an argument that will be taken as the pathname of a test `mailsurr` file. If NULL (as in `-T ""`), the system `mailsurr` file will be used. To use, type '`mail -T` *test_file recipient*' and some trivial message (like "testing"), followed by a line with either just a dot ('.') or a Ctrl-D. The result of using the `-T` option will be displayed on standard output and show the inputs and resulting transformations as `mailsurr` is processed by the `mail` command for the indicated recipient. Mail messages will never actually be sent or delivered when the `-T` option is used.

The `-x` option causes `mail` to create a file named `/tmp/MLDBG`*process_id* that contains debugging information relating to how `mail` processed the current message. The absolute value of *debug_level* controls the verboseness of the debug information. Zero implies no debugging. If *debug_level* is greater than zero, the debug file will be retained `only` if `mail` encountered some problem while processing the message. If *debug_level* is less than zero the debug file will always be retained. The *debug_level* specified via `-x` overrides any specification of `DEBUG` in `/etc/mail/mailcnfg`. The information provided by the `-x` option is esoteric and is probably only useful to system administrators. The output produced by the `-x` option is a superset of that provided by the `-T` option.

Delivery Notification

Several forms of notification are available for mail by including one of the following lines in the message header.

`Transport-Options:` [*/options*]

`Default-Options:` [*/options*]

`>To:` *recipient* [*/options*]

Where the "*/options*" may be one or more of the following:

`/delivery`	Inform the sender that the message was successfully delivered to the *recipient*'s mailbox.

`/nodelivery`	Do not inform the sender of successful deliveries.
`/ignore`	Do not inform the sender of unsuccessful deliveries.
`/return`	Inform the sender if mail delivery fails. Return the failed message to the sender.
`/report`	Same as `/return` except that the original message is not returned.

The default is `/nodelivery/return`. If contradictory options are used, the first will be recognized and later, conflicting, terms will be ignored.

FILES

`dead.letter`	unmailable text
`/etc/passwd`	to identify sender and locate recipients
`/etc/mail/mailsurr`	routing / name translation information
`/etc/mail/mailcnfg`	initialization information
`$HOME/mbox`	saved mail
`$MAIL`	variable containing pathname of *mailfile*
`/tmp/ma*`	temporary file
`/tmp/MLDBG*`	debug trace file
`/var/mail/*.lock`	lock for mail directory
`/var/mail/:saved`	directory for holding temp files to prevent loss of data in the event of a system crash.
`/var/mail/`*user*	incoming mail for *user*; that is, the *mailfile*

SEE ALSO

`chmod`(1), `login`(1), `mailx`(1), `notify`(1), `write`(1), `vacation`(1)

`mail_pipe`(1M), `mailsurr`(4), `mailcnfg`(4) in the *System Manager's Reference for the DG/UX System.*
User's Guide.

NOTES

The "Forward to recipient" feature may result in a loop. Local loops (messages sent to `usera`, which are forwarded to `userb`, which are forwarded to `usera`) will be detected immediately. Remote loops (mail sent to `sys1!usera1` which is forwarded to `sys2!userb`, which is forwarded to `sys1!usera`) will also be detected, but only after the message has exceeded the built-in hop count limit of 20. Both cases of forwarding loops will result in a non-delivery notification being sent to the message originator.

As a security precaution, the equivalent of a `chmod s+g` is performed on the *mailfile* whenever forwarding is activated via the `-F` option, and a `chmod s–g` is done when forwarding is removed via the `-F` option. If the setGID mode bit is not set when `mail` attempts to forward an incoming message to a command, the operation will fail and a non-delivery report with appropriate diagnostics will be sent to the message's originator.

The interpretation and resulting action taken because of the header lines described in the Delivery Notifications section above will only occur if this version of `mail` is installed on the system where the delivery (or failure) happens. Earlier versions of `mail` may not support any types of delivery notification.

Conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed; printing may be forced by typing a `p`.

NAME

`mailalias` – translate mail alias names

SYNOPSIS

`mailalias [ -s ] [ -v ]` *name* ...

DESCRIPTION

`mailalias` is called by `mail`. It places on the standard output a list of mail addresses corresponding to *name*. The mail addresses are found by performing the following steps:

1. Look for the file `/var/mail/`*name*. If found, print *name* and exit.
2. Look for a match in the user's local alias file `$HOME/lib/names`. If a line is found beginning with the word *name*, print the rest of the line on standard output and exit.
3. Look for a match in the system-wide alias files, which are listed in the master path file `/etc/mail/namefiles`. If a line is found beginning with the word *name*, print the rest of the line on standard output and exit.

If an alias file is a directory name *dir*, then search the file *dir*/*name*. By default, the file `/etc/mail/namefiles` lists the directory `/etc/mail/lists` and the file `/etc/mail/names`.

4. Otherwise print *name* and exit.

The alias files may contain comments (lines beginning with `#`) and information lines of the form:

name list-of-addresses

Tokens on these lines are separated by white-space. Lines may be continued by placing a backslash (\) at the end of the line.

If the `-s` option is not specified and more than one name is being translated, each line of output will be prefixed with the name being translated.

The `-v` option causes debugging information to be written to standard output.

FILES

`$HOME/lib/names`	private aliases
`/etc/mail/namefiles`	list of files to search
`/etc/mail/names`	standard file to search

SEE ALSO

`uucp(1)`. `mail(1)`,
`smtp(1M)`, `smtpd(1M)`, `smtpqer(1M)`, `smtpsched(1M)`, `tosmtp(1M)`.

NAME

`mailx` – interactive message processing system

SYNOPSIS

`mailx` [*options*] [*name* ...]

DESCRIPTION

The command `mailx` provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, `mailx` provides commands to facilitate saving, deleting, and responding to messages. When sending mail, `mailx` allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of `mailx` work only if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the `mailbox` for that user. When `mailx` is called to read messages, the `mailbox` is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the `mbox` and is normally located in the user's `HOME` directory [see `MBOX` (**Environment Variables**) for a description of this file]. Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the `-f` option of the `mailx` command. Messages in the secondary file can then be read or otherwise processed using the same **Commands** as in the primary `mailbox`. This gives rise within these pages to the notion of a current `mailbox`.

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, `mailx` attempts to read messages from the `mailbox`. Command-line options are:

`-d`	Turn on debugging output.
`-e`	Test for presence of mail. `mailx` prints nothing and exits with a successful return code if there is mail to read.
`-f` [*filename*]	Read messages from *filename* instead of `mailbox`. If no *filename* is specified, the `mbox` is used.
`-F`	Record the message in a file named after the first recipient. Overrides the `record` variable, if set (see **Environment Variables**).
`-h` *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. This option and its argument is passed to the delivery program.
`-H`	Print header summary only.
`-i`	Ignore interrupts. See also `ignore` (**Environment Variables**).
`-I`	Include the newsgroup and article-id header lines when printing mail messages. This option requires the `-f` option to be specified.
`-n`	Do not initialize from the system default *mailx.rc* file.
`-N`	Do not print initial header summary.
`-r` *address*	Use *address* as the return address when invoking the delivery program. All tilde commands are disabled. This option and

its argument is passed to the delivery program.

`-s` *subject*	Set the Subject header field to *subject*.
`-T` *file*	Message-id and article-id header lines are recorded in *file* after the message is read. This option will also set the `-I` option.
`-u` *user*	Read *user*'s `mailbox`. This is only effective if *user*'s `mailbox` is not read protected.
`-U`	Convert `uucp` style addresses to internet standards. Overrides the `conv` environment variable.
`-V`	Print the `mailx` version number and exit.
`-~`	Permits tilde escape commands no matter where the input is coming from, including non-tty input.

When reading mail, `mailx` is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating `mailx` can accept regular commands (see **Commands** below). When sending mail, `mailx` is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A subject longer than 1024 characters causes `mailx` to print the message *mail: ERROR signal 10*; the mail will not be delivered.) As the message is typed, `mailx` reads the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See **Tilde Escapes** for a summary of these commands.

At any time, the behavior of `mailx` is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the `set` and `unset` commands. See **Environment Variables** below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to be undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as `lp`(1) for recording outgoing mail on paper. Alias groups are set by the alias command (see **Commands** below) and are lists of recipients of any type.

Regular commands are of the form

[*command*] [*msglist*] [*arguments*]

If no command is specified in *command mode*, print is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

`n`	Message number `n`.
`.`	The current message.
`^`	The first undeleted message.

`$` The last message.

`*` All messages.

`n-m` An inclusive range of message numbers.

`user` All messages from `user`.

`/string`
All messages with `string` in the subject line (case ignored).

`:`*c* All messages of type *c*, where *c* is one of:

`d`	deleted messages
`n`	new messages
`o`	old messages
`r`	read messages
`u`	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions [see `sh`(1)]. Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, `mailx` tries to execute commands from the optional system-wide file (`/etc/mail/mailx.rc`) to initialize certain parameters, then from a private start-up file (`$HOME/.mailrc`) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: `!`, `Copy`, `edit`, `followup`, `Followup`, `hold`, `mail`, `preserve`, `reply`, `Reply`, `shell`, and `visual`. An error in the start-up file causes the remaining lines in the file to be ignored. The `.mailrc` file is optional, and must be constructed locally.

International Features

`mailx` enables the retrieval, editing and processing of mail messages, including characters from supplementary code sets. Supplementary code set characters can be used in the *subject* line.

Commands

The following is a complete list of `mailx` commands:

`!`*shell-command*
Escape to the shell. See `SHELL` (**Environment Variables**).

# *comment*
Null command (comment). This may be useful in `.mailrc` files.

`=` Print the current message number.

`?` Prints a summary of commands.

alias *alias name ...*
group *alias name ...*
Declare an alias for the given names. The names are substituted when *alias* is

used as a recipient. Useful in the `.mailrc` file.

`alternates` *name ...*
: Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, `alternates` prints the current list of alternate names. See also `allnet` (**Environment Variables**).

`cd` [*directory*]
`chdir` [*directory*]
: Change directory. If *directory* is not specified, `$HOME` is used.

`copy` [*filename*]
`copy` [*msglist*] *filename*
: Copy messages to the file without marking the messages as saved. Otherwise equivalent to the `save` command.

`Copy` [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the `Save` command.

`delete` [*msglist*]
: Delete messages from the `mailbox`. If `autoprint` is set, the next message after the last one deleted is printed (see **Environment Variables**).

`discard` [*header-field ...*]
`ignore` [*header-field ...*]
: Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are `status` and `cc`. The fields are included when the message is saved. The `Print` and `Type` commands override this command. If no header is specified, the current list of header fields being ignored will be printed. See also the `undiscard` and `unignore` commands.

`dp` [*msglist*]
`dt` [*msglist*]
: Delete the specified messages from the `mailbox` and print the next message after the last one deleted. Roughly equivalent to a `delete` command followed by a `print` command.

`echo` *string ...*
: Echo the given strings [like `echo(1)`].

`edit` [*msglist*]
: Edit the given messages. The messages are placed in a temporary file and the `EDITOR` variable is used to get the name of the editor (see **Environment Variables**). Default editor is `ed(1)`.

`exit`
`xit`
: Exit from `mailx`, without changing the `mailbox`. No messages are saved in the `mbox` (see also `quit`).

`file` [*filename*]
`folder` [*filename*]
: Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

`%`	the current `mailbox`.
`%`*user*	the `mailbox` for *user*.
`#`	the previous file.
`&`	the current `mbox`.

Default file is the current `mailbox`.

`folders`
: Print the names of the files in the directory set by the `folder` variable (see **Environment Variables**).

`followup` [*message*]
: Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the `record` variable, if set. See also the `Followup`, `Save`, and `Copy` commands and `outfolder` (**Environment Variables**).

`Followup` [*msglist*]
: Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the `followup`, `Save`, and `Copy` commands and `outfolder` (**Environment Variables**).

`from` [*msglist*]
: Prints the header summary for the specified messages.

`group` *alias name ...*
`alias` *alias name ...*
: Declare an alias for the given names. The names are substituted when *alias* is used as a recipient. Useful in the `.mailrc` file.

`headers` [*message*]
: Prints the page of headers which includes the message specified. The `screen` variable sets the number of headers per page (see **Environment Variables**). See also the `z` command.

`help` Prints a summary of commands.

`hold` [*msglist*]
`preserve` [*msglist*]
: Holds the specified messages in the `mailbox`.

`if` *s* | *r*
mail-commands
`else`
mail-commands
`endif` Conditional execution, where *s* executes following *mail-commands*, up to an `else` or `endif`, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the `.mailrc` file.

`ignore` [*header-field ...*]
`discard` [*header-field ...*]
: Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are `status` and `cc`. All fields are included when the message is saved. The `Print` and `Type` commands override this command. If no header is specified, the current list of header fields being ignored will be printed. See also the `undiscard` and

unignore commands.

list Prints all commands available. No explanation is given.

mail *name* ...
Mail a message to the specified users.

Mail *name*
Mail a message to the specified user and record a copy of it in a file named after that user.

mbox [*msglist*]
Arrange for the given messages to end up in the standard mbox save file when mailx terminates normally. See MBOX (**Environment Variables**) for a description of this file. See also the exit and quit commands.

next [*message*]
Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglist*s above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]
Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the cmd variable. If the page variable is set, a form feed character is inserted after each message (see **Environment Variables**).

preserve [*msglist*]
hold [*msglist*]
Preserve the specified messages in the mailbox.

Print [*msglist*]
Type [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

print [*msglist*]
type [*msglist*]
Print the specified messages. If crt is set, the messages longer than the number of lines specified by the crt variable are paged through the command specified by the PAGER variable. The default command is pg(1) (see **Environment Variables**).

quit Exit from mailx, storing messages that were read in mbox and unread messages in the mailbox. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If record is set to a file name, the response is saved at the end of that file (see **Environment Variables**).

reply [*message*]

`respond` [*message*]
: Reply to the specified message, including all other recipients of the message. If `record` is set to a file name, the response is saved at the end of that file (see **Environment Variables**).

`Save` [*msglist*]
: Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the `Copy`, `followup`, and `Followup` commands and `outfolder` (**Environment Variables**).

`save` [*filename*]
`save` [*msglist*] *filename*
: Save the specified messages in the given file. The file is created if it does not exist. The file defaults to `mbox`. The message is deleted from the `mailbox` when `mailx` terminates unless `keepsave` is set (see also **Environment Variables** and the `exit` and `quit` commands).

`set`
`set` *name*
`set` *name=string*
`set` *name=number*
: Define a variable called *name*. The variable may be given a null, string, or numeric value. `Set` by itself prints all defined variables and their values. See **Environment Variables** for detailed descriptions of the `mailx` variables.

`shell` Invoke an interactive shell [see also `SHELL` (**Environment Variables**)].

`size` [*msglist*]
: Print the size in characters of the specified messages.

`source` *filename*
: Read commands from the given file and return to command mode.

`top` [*msglist*]
: Print the top few lines of the specified messages. If the `toplines` variable is set, it is taken as the number of lines to print (see **Environment Variables**). The default is 5.

`touch` [*msglist*]
: Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it is placed in the `mbox`, or the file specified in the `MBOX` environment variable, upon normal termination. See `exit` and `quit`.

`Type` [*msglist*]
`Print` [*msglist*]
: Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the `ignore` command.

`type` [*msglist*]
`print` [*msglist*]
: Print the specified messages. If `crt` is set, the messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` variable. The default command is `pg`(1) (see **Environment Variables**).

`undelete` [*msglist*]
: Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If `autoprint` is set, the last message of those

restored is printed (see **Environment Variables**).

`undiscard` *header-field ...*
`unignore` *header-field ...*
Remove the specified header fields from the list being ignored.

`unset` *name ...*
Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

`version`
Prints the current version.

visual [*msglist*]
Edit the given messages with a screen editor. The messages are placed in a temporary file and the `VISUAL` variable is used to get the name of the editor (see **Environment Variables**).

write [*msglist*] *filename*
Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the `save` command.

xit
`exit` Exit from `mailx`, without changing the `mailbox`. No messages are saved in the `mbox` (see also `quit`).

`z[+ | -]`
Scroll the header display forward or backward one screen–full. The number of headers displayed is set by the `screen` variable (see **Environment Variables**).

Tilde Escapes

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (˜). See `escape` (**Environment Variables**) for changing this special character.

`~!` *shell-command*
Escape to the shell.

`~.` Simulate end of file (terminate message input).

`~:` *mail-command*
`~_` *mail-command*
Perform the command-level request. Valid only when sending a message while reading mail.

`~?` Print a summary of tilde escapes.

`~A` Insert the autograph string `Sign` into the message (see **Environment Variables**).

`~a` Insert the autograph string `sign` into the message (see **Environment Variables**).

`~b` *names ...*
Add the *names* to the blind carbon copy (Bcc) list.

`~c` *names ...*
Add the *names* to the carbon copy (Cc) list.

`~d` Read in the `dead.letter` file. See `DEAD` (**Environment Variables**) for a description of this file.

`~e`
: Invoke the editor on the partial message. See also `EDITOR` (**Environment Variables**).

`~f` [*msglist*]
: Forward the specified messages. The messages are inserted into the message without alteration.

`~h`
: Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

`~i` *string*
: Insert the value of the named variable into the text of the message. For example, `~A` is equivalent to `~iSign`. Environment variables set and exported in the shell are also accessible by ~i.

`~m` [*msglist*]
: Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

`~p`
: Print the message being entered.

`~q`
: Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in `dead.letter`. See `DEAD` (**Environment Variables**) for a description of this file.

`~r` *filename*
`~<` *filename*
`~<` !*shell-command*
: Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

`~s` *string* ...
: Set the subject line to *string*.

`~t` *names* ...
: Add the given *names* to the To list.

`~v`
: Invoke a preferred screen editor on the partial message. See also `VISUAL` (**Environment Variables**).

`~w` *filename*
: Write the message into the given file, without the header.

`~x`
: Exit as with `~q` except the message is not saved in `dead.letter`.

`~|` *shell-command*
: Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

Environment Variables

The following are environment variables taken from the execution environment and are not alterable within `mailx`.

`HOME=`*directory*
: The user's base of operations.

`MAILRC=`*filename*
: The name of the start-up file. Default is `$HOME/.mailrc`.

The following variables are internal `mailx` variables. They may be imported from the execution environment or set via the `set` command at any time. The `unset` command may be used to erase variables.

`allnet`
: All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is `noallnet`. See also the `alternates` command and the `metoo` variable.

`append`
: Upon termination, append messages to the end of the `mbox` file instead of prepending them. Default is `noappend`.

`askcc` Prompt for the Cc list after the Subject is entered. Default is `noaskcc`.

`askbcc`
: Prompt for the Bcc list after the Subject is entered. Default is `noaskbcc`.

`asksub`
: Prompt for subject if it is not specified on the command line with the `-s` option. Enabled by default.

`autoprint`
: Enable automatic printing of messages after `delete` and `undelete` commands. Default is `noautoprint`.

`bang` Enable the special-casing of exclamation points (!) in shell escape command lines as in `vi`(1). Default is `nobang`.

`cmd=`*shell-command*
: Set the default command for the `pipe` command. No default value.

`conv=`*conversion*
: Convert uucp addresses to the specified address style. The only valid conversion now is `internet`, which uses domain-style addressing. Conversion is disabled by default. See also the `-U` command-line option.

`crt=`*number*
: Pipe messages having more than *number* lines through the command specified by the value of the `PAGER` variable [`pg`(1) by default]. Disabled by default.

`DEAD=`*filename*
: The name of the file in which to save partial letters in case of untimely interrupt. Default is `$HOME/dead.letter`.

`debug` Enable verbose diagnostics for debugging. Messages are not delivered. Default is `nodebug`.

`dot` Take a period on a line by itself during input from a terminal as end-of-file. Default is `nodot`.

`EDITOR=`*shell-command*
: The command to run when the `edit` or `~e` command is used. Default is `ed`(1).

`escape=`*c*
: Substitute *c* for the ˜ escape character. Takes effect with next message sent.

`folder=`*directory*
: The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), `$HOME` is prepended to it. In order to use the plus (+) construct on a `mailx` command line, `folder` must be an exported `sh` environment variable. There is no default for the `folder` variable. See also `outfolder` below.

`header`
: Enable printing of the header summary when entering `mailx`. Enabled by default.

`hold` Preserve all messages that are read in the `mailbox` instead of putting them in the standard `mbox` save file. Default is `nohold`.

`ignore`
: Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is `noignore`.

`ignoreeof`
: Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the `˜.` command. Default is `noignoreeof`. See also `dot` above.

`keep` When the `mailbox` is empty, truncate it to zero length instead of removing it. Disabled by default.

`keepsave`
: Keep messages that have been saved in other files in the `mailbox` instead of deleting them. Default is `nokeepsave`.

`MBOX=`*filename*
: The name of the file to save messages which have been read. The `xit` command overrides this function, as does saving the message explicitly in another file. Default is `$HOME/mbox`.

`metoo` If your login appears as a recipient, do not delete it from the list. Default is `nometoo`.

`LISTER=`*shell-command*
: The command (and options) to use when listing the contents of the `folder` directory. The default is `ls`(1).

`onehop`
: When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

`outfolder`
: Causes the files used to record outgoing messages to be located in the directory specified by the `folder` variable unless the path name is absolute. Default is `nooutfolder`. See `folder` above and the `Save`, `Copy`, `fol-`

lowup, and `Followup` commands.

`page` Used with the `pipe` command to insert a form feed after each message sent through the pipe. Default is `nopage`.

`PAGER=`*shell-command*
: The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is `pg`(1).

`prompt=`*string*
: Set the *command mode* prompt to *string*. Default is "`? `".

`quiet` Refrain from printing the opening message and version when entering `mailx`. Default is `noquiet`.

`record=`*filename*
: Record all outgoing mail in *filename*. Disabled by default. See also `outfolder` above. If you have the `record` and `outfolder` variables set but the `folder` variable not set, messages are saved in *+filename* instead of *filename*.

`save` Enable saving of messages in `dead.letter` on interrupt or delivery error. See `DEAD` for a description of this file. Enabled by default.

`screen=`*number*
: Sets the number of lines in a screen–full of headers for the `headers` command. It must be a positive number.

`sendmail=`*shell-command*
: Alternate command for delivering messages. Default is `/usr/bin/rmail`.

`sendwait`
: Wait for background mailer to finish before returning. Default is `nosendwait`.

`SHELL=`*shell-command*
: The name of a preferred command interpreter. Default is `sh`(1).

`showto`
: When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

`sign=`*string*
: The variable inserted into the text of a message when the `~a` (autograph) command is given. No default [see also `~i` (**Tilde Escapes**)].

`Sign=`*string*
: The variable inserted into the text of a message when the `~A` command is given. No default [see also `~i` (**Tilde Escapes**)].

`toplines=`*number*
: The number of lines of header to print with the `top` command. Default is 5.

`VISUAL=`*shell-command*
: The name of a preferred screen editor. Default is `vi`.

FILES

`$HOME/.mailrc`	personal start-up file
`$HOME/mbox`	secondary storage file
`/var/mail/*`	post office directory
`/usr/share/lib/mailx/mailx.help*`	help message files
`/etc/mail/mailx.rc`	optional global start-up file
`/tmp/R[emqsx]*`	temporary files

SEE ALSO

ls(1), mail(1), pg(1), vi(1).

NOTES

The `-h` and `-r` options can be used only if `mailx` is using a delivery program other than `/usr/bin/rmail`.

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be `unset`.

The full internet addressing is not fully supported by `mailx`. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by mail(1) (the standard mail delivery program).

Mailx(1) relies on the message header to be in a specific format. When editing a mail message, changing the contents of the fields of the header is acceptable. However, modifying the format of the message header, such as inserting or deleting lines, or adding spaces can cause unpredictable results.

NAME

makekey − generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper and lowercase letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

Makekey is intended for programs that perform encryption (e.g., ed(1) and crypt(1)). Usually, its input and output will be pipes.

EXAMPLES

```
$ /usr/lib/makekey < inkey > outkey
```

The 10 character input key in file "inkey" is encrypted and placed in file "outkey".

SEE ALSO

crypt(1), ed(1).
passwd(4) in the *Programmer's Reference for the DG/UX System*

NAME

man – locate and print entries from the reference manuals

SYNOPSIS

man [-T*term*] [-dw] [-M*path*] [*class*] *name* ...
man [-M*path*] -k *keyword* ...
man [-M*path*] -f *filename* ...

where:

term	A terminal type (for a list of possible values, type man term); default = $TERM
path	A colon- or space-separated list of directories to search; default = $MANPATH
class	An integer from 0 to 8 indicating the class of entry (see **Man Page Classes** below)
name	The name of the entry in lowercase letters
keyword	A word for which to search, from the **NAME** section of an entry
filename	The base file name of an entry for which to search

DESCRIPTION

Man locates and prints the specified entries of this manual. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) It can display complete entries that you select by *name*, or one-line summaries selected either by *keyword* or by the base name (i.e., without suffix) of an entry *filename*.

With one or more *name* arguments, if you do not specify a class number, the whole manual is searched for the specified entry name, and all occurrences of it are printed.

In many cases, more than one command or system call or routine is listed under a single entry name; for example, the basename and dirname commands are described in the basename(1) manual page. You can access such manual pages by specifying any of the entries on the page.

Options

-T Print the entry as appropriate for terminal type *term*.

-M Change the search path for entries to *path*. *Path* contains a colon- or space-separated list of directories that contain manual page directory subtrees. Each directory in the *path* is assumed to contain subdirectories of the form man*N*, where *N* is a digit.

-w Print on the standard output only the pathnames of the entries in /usr/catman, or to the current directory if -d is also specified.

-d Search the current directory rather than /usr/catman; requires the full filename (e.g., spline.1g, rather than just spline).

-k Print on the standard output all one-line summaries from the whatis database that contain any of the given *keyword*s.

-f Locate entries related to any of the given *filename*s. For each entry in the whatis database of the form *filename*(*class*[*x*]) (where *class* is a number and *x* is a lowercase letter), man prints on the standard output the associated one-line summary.

Environment Variables

MANPATH Specify directories to be searched for manual pages. This variable contains a colon- or space-separated list of directories; the entries should be in these directories or in subdirectories with names of the form man*N*, where *N* is a digit. These directories will be searched first by man for each entry *name* specified. If no matching manual entries are found therein, man will also search the directories /usr/catman/*_man. (Thus by default man searches only /usr/catman/*_man.) The -M option overrides the value of MANPATH.

PAGER	Specify a program into which output is to be piped. The program will be invoked for each entry name matching the specified *name* argument(s). Therefore, quitting the display of an entry may result in displaying the next entry, if one exists. The default is no piping.
TERM	Specify the terminal type for which output is to be adapted [see environ(5)]; the default is lp. The -T option overrides the TERM setting. You should use the -Tlp option when sending the output to a line printer, since TERM is normally set to a value other than lp.

Man Page Classes

The manual pages are divided into the following classes:

(0)	Table of contents and permuted keyword-in-context index
(1)	Commands and application programs
(2)	System calls
(3)	Subroutines and libraries
(4)	File formats
(5)	Miscellaneous features
(6)	Network protocols
(7)	System special files
(8)	System maintenance procedures

The *User's Reference for the DG/UX System*, *Programmer's Reference for the DG/UX System*, and *System Manager's Reference for the DG/UX System* have two class-0 entries in common: contents(0) and index(0). The contents(0) manual page lists all the manual pages alphabetically by class. The index(0) manual page contains a permuted keyword-in-context index for all the DG/UX manual pages.

If your system manager installs the TCP/IP manual pages, contents(0) and index(0) are overwritten with a version that is a superset of the DG/UX version. If your system manager installs the NFS manual pages, contents(0) and index(0) are overwritten with a version that is a superset of the DG/UX and TCP/IP versions.

Adding New Manual Pages

To add manual pages to the system, the superuser can put them in usr/catman, or you can put them in your own directory and add that directory's absolute pathname to the MANPATH variable (see **Environment Variables** above). The manual page should be in lineprinter format.

The manual page may be compressed (via the pack program) or uncompressed. The filename should be of the form *name*.*class*[*x*][.z], where *x* is a lowercase letter and .z is required for compressed entries and is automatically appended by the pack program.

The macros normally used to format manual pages are part of Xroff (available from Image Network, 140 South Whisman Road, Mountain View, California) and GNU troff (available from the Free Software Foundation 675 Mass Ave., Cambridge, MA 02139) and are described in the man(7) manual page, available with Xroff and GNU troff. These macros can be formatted with xroff and groff, which is also part of Xroff and GNU troff. Starting with Release 5.4, the DG/UX System has not included nroff or troff.

EXAMPLES

To display the chmod(1) manual page:

```
man 1 chmod
```

To display the chmod(1) and chmod(2) manual pages:

```
man chmod
```

To lineprint the manual page for basename and dirname:

```
man -Tlp basename | lp
or
man -Tlp dirname | lp
```

FILES

/usr/catman/?_man/man[0-8]/*	Formatted manual entries
/usr/catman/?_man/whatis	Table of contents (whatis) databases

SEE ALSO

apropos(1), whatis(1), ul(1), more(1), term(5), syscon(7).
ditroff(1), xroff(1), man(7) in Xroff.
groff(1), gtroff(1), man(7) in GNU troff.

NOTES

The man command prints manual entries that were formatted by nroff before the system was installed. Entries are specially formatted for processing by ul(1), which is called by the man command. Printing on other printers or typesetting requires the addition of some form of troff and the man page source files (only formatted entries are included for use by the man command).

When piping the output of man through the more(1) command, specify the -f option to the pager for best results. Otherwise the reference manual lines and pager prompts may sometimes be printed in the wrong places. The mispositioning occurs when more(1) becomes confused by non-printing video attribute characters typically displayed to a terminal screen. Note that if you are using the PAGER environment variable, you can place pager options into it in addition to simple command names.

NAME

`merge` – three-way file merge

SYNOPSIS

`merge` [`-L` *label1* [`-L` *label3*]] [`-p`] [`-q`] *file1 file2 file3*

DESCRIPTION

`merge` incorporates all changes that lead from *file2* to *file3* into *file1*. The result goes to standard output if `-p` is present, into *file1* otherwise. `merge` is useful for combining separate changes to an original. Suppose *file2* is the original, and both *file1* and *file3* are modifications of *file2*. Then `merge` combines both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. `merge` outputs a message if overlaps occurred, and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, the user should edit the result and delete one of the alternatives. If the `-L` *label1* and `-L` *label3* options are given, the labels are output in place of the names *file1* and *file3* in overlap reports. Any overlap message is suppressed if the `-q` option is given.

DIAGNOSTICS

Exit status is 0 for no overlaps, 1 for some overlaps, 2 for trouble.

IDENTIFICATION

Author: Walter F. Tichy.
Revision Number: 1.2; Release Date: 1991/10/11.
Copyright © 1982, 1988, 1989 by Walter F. Tichy.
Copyright © 1990 by Paul Eggert.

SEE ALSO

`diff3(1)`, `diff(1)`, `rcsmerge(1)`, `co(1)`.

NAME

`mesg` – permit or deny messages

SYNOPSIS

`mesg [ n ] [ y ]`

DESCRIPTION

`mesg n`	Prevents other users from sending you messages with `write`(1). (It takes away non-user write permission on the user's terminal).
`mesg y`	Resets the permission so that you can receive messages.
`mesg`	Reports the current state without changing it.

EXAMPLES

```
$ mesg
is y

$ mesg n
$ mesg
is n
```

In the above example, the user has turned off the ability to receive messages.

FILES

`/dev/tty*`

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

SEE ALSO

`write`(1).

NAME

mkdir – make a directory

SYNOPSIS

mkdir [-c] [-p] [-m*mode*] *dirname* ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by umask(1)). Standard entries (. for the directory itself, and .. for its parent) are made automatically.

Mkdir requires write permission in the parent directory.

With the -p option, mkdir also creates any non-existing parent directories of the new directory. The -m option lets you specify the new directories' mode. For values of mode, see chmod(1).

The -c option specifies that the directory created is a control point directory. When -c and -p are both given, any non-existing parent directories are created as regular directories with the same owner, group, and mode as the control point directory. You set and display a control point directory's allocation limits with the cpd(1) command. You can create control point directories only on local filesystems; therefore, diskless workstations cannot create control point directories.

EXAMPLES

```
$ pwd
/usr/user1
$ mkdir c_programs
$ cd c_programs
$ pwd
/usr/user1/c_programs
$
```

The example above creates a directory called c_programs in the user1 directory.

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero. An error code is stored in errno.

SEE ALSO

chmod(1), cpd(1), sh(1), rm(1), umask(1).

NAME

mkmsgs – create message files for use by gettxt

SYNOPSIS

mkmsgs [-o] [-i *locale*] *inputstrings msgfile*

DESCRIPTION

The mkmsgs utility is part of the AT&T-style message facility. It is used to create a file of text strings that can be accessed using the text retrieval tools (see gettxt(1), srchtxt(1), exstr(1), and gettxt(3C)). It will take as input a file of text strings for a particular geographic locale (see setlocale(3C)) and create a file of text strings in a format that can be retrieved by both gettxt(1) and gettxt(3C). By using the -i option, you can install the created file under the /usr/lib/locale/*locale*/LC_MESSAGES directory (*locale* corresponds to the language in which the text strings are written).

inputstrings	the name of the file that contains the original text strings.
msgfile	the name of the output file where mkmsgs writes the strings in a format that is readable by gettxt(1) and gettxt(3C). The name of *msgfile* can be up to 14 characters in length, but may not contain either \0 (null) or the ASCII code for / (slash) or : (colon).
-i *locale*	install *msgfile* in the /usr/lib/locale/*locale*/LC_MESSAGES directory. Only someone who is super-user or a member of group bin can create or overwrite files in this directory. Directories under /usr/lib/locale will be created if they don't exist.
-o	overwrite *msgfile*, if it exists.

The input file contains a set of text strings for the particular geographic locale. Text strings are separated by a new-line character. Nongraphic characters must be represented as alphabetic escape sequences. Messages are transformed and copied sequentially from *inputstrings* to *msgfile*. To generate an empty message in *msgfile*, leave an empty line at the correct place in *inputstrings*.

Strings can be changed simply by editing the file *inputstrings*. New strings must be added only at the end of the file; then a new *msgfile* file must be created and installed in the correct place. If this procedure is not followed, the retrieval function will retrieve the wrong string and software compatibility will be broken.

EXAMPLES

The following example shows an input message source file C.str:

```
File %s:\t cannot be opened\n
%s: Bad directory\n
        .
        .
        .
write error\n
        .
        .
```

The following command uses the input strings from C.str to create text strings in the appropriate format in the file UX in the current directory:

```
mkmsgs C.str UX
```

The following command uses the input strings from FR.str to create text strings in the appropriate format in the file UX in the directory

/usr/lib/locale/french/LC_MESSAGES/UX.

```
        mkmsgs -i french FR.str UX
```

These text strings would be accessed if you had set the environment variable LC_MESSAGES=french and then invoked one of the text retrieval tools listed at the beginning of the **DESCRIPTION** section.

FILES

/usr/lib/locale/*locale*/LC_MESSAGES/* Message files created by mkmsgs(1M)

SEE ALSO

exstr(1), gettxt(1), srchtxt(1), gettxt(3C), setlocale(3C).
gencat(1), catgets(1), catopen(3C), catgets(3C) — X/Open-style message facilities.

NAME

`more`, `page` – display file one screenful at a time

SYNOPSIS

`more` [`-cdflpsu`] [*-n*] [*+linenumber*] [*+/pattern*] [*name* ...]

`page` [`more` *options*]

DESCRIPTION

`More` is a filter that allows you to examine a long text one CRT-screenful at a time. It pauses after each screenful, printing `--More--` at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user types a space, another screenful is displayed. Other possibilities are listed below in the *Commands* section.

Options

The command line options are:

-n An integer which is the size (in lines) of the window which `more` will use instead of the default.

`-c` `More` will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while `more` is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

`-d` `More` will prompt the user with the message "Press space to continue, ′q′ to quit." at the end of each screenful, and will respond to subsequent illegal user input by printing "Press ′h′ for instructions." instead of ringing the bell. This is useful if `more` is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

`-f` This causes `more` to count logical lines, rather than screen lines. That is, long lines are not folded. This option is recommended for viewing `ul`(1) output, since it may contain escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus `more` may think that lines are longer than they actually are, and fold lines erroneously.

`-l` Do not treat ^L (form feed) specially. If this option is not given, `more` will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

`-p` Clear the screen before each screenful is printed (but only if a full screenful is being printed), and print $k - 1$ rather than $k - 2$ lines in each screenful, where k is the number of lines the terminal can display.

`-s` Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing formatted text, this option maximizes the useful information present on the screen.

`-u` Normally, `more` will handle underlining in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, `more` will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The `-u` option suppresses this processing.

+linenumber
  Start up at *linenumber*.

+/pattern
  Start up two lines before the line containing the regular expression *pattern*.

If the program is invoked as `page`, then it behaves the same as `more` with the `-p` option.

`More` accesses the `terminfo`(4) database to determine terminal characteristics such as the screen size, and to determine the default window and scroll sizes. On a terminal with no fixed number of rows, such as a hardcopy printer, the screen size defaults to 24 lines. The default window size is normally two less than the screen size. However, if the `-p` option is specified or the program is invoked as `page`, the window size defaults to one less than the length of the screen. The scroll size defaults to half the window size.

`More` looks in the environment variable `MORE` to preset any flags desired. For example, if you prefer to view files using the `-c` mode of operation, the `csh` command `setenv MORE -c` or the `sh` command sequence `MORE='-c' ; export MORE` would cause all invocations of `more` to use this mode. Normally, the user will place the command sequence which sets up the `MORE` environment variable in the `.login` or `.profile` file (see `profile(4)`).

If `more` is reading from a file, rather than a pipe, then a percentage is displayed along with the `--More--` prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Commands

Other sequences that may be typed when `more` pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1 unless otherwise noted):

i space Display *i* more lines, or another screenful if no argument is given.

i d Display *i* more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*. Otherwise, the default is to scroll 11 lines.

i ^D (control-D)
  Same as `d`.

i z Same as typing a space except that *i*, if present, becomes the new window size.

i s Skip *i* lines and print a screenful of lines.

i f Skip *i* screenfuls and print a screenful of lines.

i b Skip back *i* screenfuls and print a screenful of lines. This command works only when the input is a file, not a pipe.

i ^B (control-B)
  Same as `b`.

q or Q Exit from `more`.

= Display the current line number.

v Start up the editor `vi(1)` at the current line. This command works only when the input is a file, not a pipe.

h Help command; give a description of all the `more` commands.

i /expr Search for the *i*-th occurrence of the regular expression *expr*. If the search is successful a screenful is displayed, starting two lines before the place where the expression was found. Otherwise, if the input is a file, the position in the file remains unchanged; if the input is a pipe, `more` terminates. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n Search for the *i*-th occurrence of the last regular expression entered.

´ (single quote)
: Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file. This command works only when the input is a file, not a pipe.

!*command*
: Invoke a shell and execute *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n Skip to the *i*-th next file given in the command line. If *i* doesn't make sense, skip to the last file.

i:p Skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, `more` counts the current file when doing the skipping. If *i* doesn't make sense, `more` skips back to the first file. If `more` is not reading from a file, the terminal bell is rung and nothing else happens.

:f Display the current file name and line number.

:q or :Q
: Exit from `more` (same as `q` or `Q`).

. (dot) Repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may type the line kill character to cancel the numerical argument being formed. In addition, the user may type the erase character to redisplay the `--More--(xx%)` message.

At any time when output is being sent to the terminal, the user can type the quit character (normally control-\) to interrupt the display. `more` will stop sending output, and will display the usual `--More--` prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to `noecho` mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a tty device then `more` acts just like `cat`, except that a header is printed before each file if there is more than one.

EXAMPLES

```
$ more textfile
```

Display the contents of the file "textfile" using the `more` processor. This allows for display of the file "textfile" on a screen by screen basis.

```
$ man more | more -f
```

Prints the manual page for the `more` command and pipes the output to the `more` processor. This allows for display of the `more` documentation on a screen by screen basis. The `-f` option ensures that `more` will display the output of the `man`(1) command correctly, since `man` generates escape sequences.

```
$ ls -l | more
```

Display all information about the files in the current working directory and pipes the output to the `more` processor. This allows for display of the directory listing on a screen by screen basis.

FILES

`/usr/lib/terminfo/?/*`	Terminal data base
`/usr/lib/more.help`	Help file

SEE ALSO

`pg`(1), `cat`(1), `sh`(1), `csh`(1), `ul`(1).
`profile`(4), `environ`(5), `term`(5) in the *Programmer's Reference for the DG/UX System*.

BUGS

Skipping backwards is too slow on large files.

NAME

mt – magnetic tape control

SYNOPSIS

mt [-f *tapename*] *command* [*count*]

DESCRIPTION

mt sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable TAPE is used. If TAPE does not exist, mt prints an error message. Note that *tapename* must refer to a raw (not block) tape device. For tape-positioning operations, the tape device should be a non-rewinding device; otherwise, the tape rewinds at the end of the operation regardless of any position request. For example, specify /dev/rmt/0n instead of /dev/rmt/0.

By default, mt performs the requested operation once; *count* specifies multiple operations.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

eof, weof	Write *count* EOF marks at the current position on the tape.
fsf	Forward space *count* files.
fsr	Forward space *count* records.
bsf	Back space *count* files.
bsr	Back space *count* records.
rewind	Rewind the tape. The *count* argument is ignored with this command.
status	Print status information about the tape unit. The *count* argument is ignored with this command.

The exit status values are:

0	The operation succeeded.
1	mt does not recognize *command*, or cannot open the specified tape drive.
2	The operation failed.

FILES

/dev/rmt/*	raw magnetic tape interface

SEE ALSO

dd(1), ioctl(2), environ(5).

NAME

mv — move files

SYNOPSIS

mv [-f] [-i] *file1* [*file2* ...] *target*

DESCRIPTION

The mv command moves *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, mv 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 moved to that directory.

If mv 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 mv occurs, if permissible; otherwise, the command exits. When the parent directory of *filen* is writable and has the sticky bit set, one or more of the following conditions must be true:

the user must own the file
the user must own the directory
the file must be writable by the user
the user must be a privileged user

The following options are recognized:

-i mv will prompt for confirmation whenever the move would overwrite an existing *target*. A y answer means that the move should proceed. Any other answer prevents mv from overwriting the *target*.

-f mv will move the file(s) without prompting even if it is writing over an existing *target*. This option overrides the -i option. Note that this is the default if the standard input is not a terminal.

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.

NOTES

If *filen* and *target* are on different file systems, mv copies the file and deletes the original; any links to other files are lost.

A -- permits the user to mark explicitly the end of any command line options, allowing mv to recognize filename arguments that begin with a -. As an aid to BSD migration, mv will accept - as a synonym for --. This migration aid may disappear in a future release. If a -- and a - both appear on the same command line, the second will be interpreted as a filename.

SEE ALSO

chmod(1), cp(1), cpio(1), ln(1), rm(1).

NAME

`nawk`, `awk` – pattern scanning and processing language

SYNOPSIS

`nawk` [`-F` *re*] [`-v` *var=value*] [*'prog'*] [*file...*]

`nawk` [`-F` *re*] [`-v` *var=value*] [`-f` *progfile*] [*file...*]

DESCRIPTION

`nawk` scans each input *file* for lines that match any of a set of patterns specified in *prog*. The *prog* string must be enclosed in single quotes (') to protect it from the shell. For each pattern in *prog* there may be an associated action performed when a line of a *file* matches the pattern. The set of pattern-action statements may appear literally as *prog* or in a file specified with the `-f` *progfile* option. Input files are read in order; if there are no files, the standard input is read. The file name `-` means the standard input.

Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern. Any *file* of the form `var=`*value* is treated as an assignment, not a filename, and is executed at the time it would have been opened if it were a filename. The option `-v` followed by `var=`*value* is an assignment to be done before *prog* is executed; any number of `-v` options may be present.

An input line is normally made up of fields separated by white space. (This default can be changed by using the `FS` built-in variable or the `-F` *re* option.) The fields are denoted `$1, $2, ...; $0` refers to the entire line.

A pattern-action statement has the form:

> *pattern* `{` *action* `}`

Either pattern or action may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line. Pattern-action statements are separated by newlines or semicolons.

Patterns are arbitrary Boolean combinations (`!`, ||, `&&`, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

> *expression relop expression*
> *expression matchop regular_expression*
> *expression* `in` *array-name*
> `(`*expression*`,`*expression*`, ...` `)` `in` *array-name*

where a *relop* is any of the six relational operators in C, and a *matchop* is either `~` (contains) or `!~` (does not contain). An *expression* is an arithmetic expression, a relational expression, the special expression

> *var* `in` *array*

or a Boolean combination of these.

Regular expressions are as in `egrep`(1). In patterns they must be surrounded by slashes. 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 pattern.

The special patterns `BEGIN` and `END` may be used to capture control before the first input line has been read and after the last input line has been read respectively. These keywords do not combine with any other patterns.

A regular expression may be used to separate fields by using the `-F` *re* option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

`ARGC`	command line argument count
`ARGV`	command line argument array
`ENVIRON`	array of environment variables; subscripts are names
`FILENAME`	name of the current input file
`FNR`	ordinal number of the current record in the current file
`FS`	input field separator regular expression (default blank and tab)
`NF`	number of fields in the current record
`NR`	ordinal number of the current record
`OFMT`	output format for numbers (default `%.6g`)
`OFS`	output field separator (default blank)
`ORS`	output record separator (default new-line)
`RS`	input record separator (default new-line)
`SUBSEP`	separates multiple subscripts (default is 034)

An action is a sequence of statements. A statement may be one of the following:

```
if ( expression ) statement [ else statement ]
while ( expression ) statement
do statement while ( expression )
for ( expression ; expression ; expression ) statement
for ( var in array ) statement
delete array[subscript] #delete an array element
break
continue
{ [ statement ] ... }
expression      # commonly variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next            # skip remaining patterns on this input line
exit [expr]     # skip the rest of the input; exit status is expr
return [expr]
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators `+`, `-`, `*`, `/`, `%`, `^` and concatenation (indicated by a blank). The operators `++ -- += -= *= /= %= ^= > >= < <= == != ?:` are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as `[i,j,k]` are permitted; the constituents are concatenated, separated by the value of `SUBSEP`. String constants are quoted (" "), with the usual C excapes recognized within.

The `print` statement prints its arguments on the standard output, or on a file if >*expression* is present, or on a pipe if | *cmd* is present. The arguments are separated by the current output field separator and terminated by the output record separator. The `printf` statement formats its expression list according to the format [see *printf*(3S) in the *Programmer's Reference for the DG/UX System (Volume 2)*]. The built-in function `close`(*expr*) closes the file or pipe *expr*.

The mathematical functions: `atan2`, `cos`, `exp`, `log`, `sin`, `sqrt`, are built-in.

Other built-in functions include:

`gsub`(*for*, *repl*, *in*)
: behaves like `sub` (see below), except that it replaces successive occurrences of the regular expression (like the `ed` global substitute command).

`index(`*s*`,` *t*`)` returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.

`int` truncates to an integer value.

`length(`*s*`)` returns the length of its argument taken as a string, or of the whole line if there is no argument.

`match(`*s*`,` *re*`)` returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. `RSTART` is set to the starting position (which is the same as the returned value), and `RLENGTH` is set to the length of the matched string.

`rand` random number on (0, 1).

`split(`*s*`,` *a*`,` *fs*`)`
: splits the string *s* into array elements *a*[*1*], *a*[2], *a*[*n*], and returns *n*. The separation is done with the regular expression *fs* or with the field separator `FS` if *fs* is not given.

`srand` sets the seed for `rand`

`sprintf(`*fmt*`,` *expr*`,` *expr*`, ... )`
: formats the expressions according to the `printf`(3S) format given by *fmt* and returns the resulting string.

`sub`(*for*, *repl*, *in*)
: substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, `nawk` substitutes in the current record (`$0`).

`substr`(*s*, *m*, *n*)
: returns the *n*-character substring of *s* that begins at position *m*.

The input/output built-in functions are:

`close(`*filename*`)`
: closes the file or pipe named *filename*.

cmd | `getline`
: pipes the output of *cmd* into `getline`; each successive call to *getline* returns the next line of output from *cmd*.

`getline`	sets `$0` to the next input record from the current input file.
`getline <`*file*	sets `$0` to the next record from *file*.
`getline` *x*	sets variable *x* instead.
`getline` *x* `<`*file*	sets *x* from the next record of *file*.
`system(`*cmd*`)`	executes *cmd* and returns its exit status.

All forms of `getline` return 1 for successful input, 0 for end of file, and −1 for an error.

`nawk` also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as

```
function name(args,...) { stmts }
```

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The `return` statement may be used to return a value.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average:

```
      { s += $1 }
END   { print "sum is", s, " average is", s/NR }
```

Print 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 }
```

Simulate `echo`(1):

```
BEGIN {
      for (i = 1; i < ARGC; i++)
            printf "%s", ARGV[i]
      printf "\n"
      exit
      }
```

Print a file, filling in page numbers starting at 5:

```
/Page/{ $2 = n++; }
      { print }
```

Assuming this program is in a file named `prog`, the following command line prints the file `input` numbering its pages starting at 5: `nawk -f prog n=5 input`.

SEE ALSO

`egrep(1)`, `grep(1)`, `oawk(1)`, `sed(1)`, `lex(1)`, `printf(3S)`.
The `awk` chapter in the *Using the DG/UX Software Development Tools*.
A. V. Aho, B. W. Kerninghan, P. J. Weinberger, *The AWK Programming Language* Addison-Wesley, 1988.

NOTES

`nawk` is a new version of `awk` that provides capabilities unavailable in previous versions. This version will eventually become the default version of `awk`.

BUGS

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

`newform` – change the format of a text file

SYNOPSIS

`newform` [`-s`] [`-i`*tabspec*] [`-o`*tabspec*] [`-b`*n*] [`-e`*n*] [`-p`*n*] [`-a`*n*] [`-f`] [`-c`*char*] [`-l`*n*] [*files*]

DESCRIPTION

`Newform` reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for `-s`, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "`-e15 -l60`" will yield results different from "`-l60 -e15`." Options are applied to all *files* on the command line.

`-i`*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in `tabs`(1). In addition, *tabspec* may be `--`, in which `newform` assumes that the tab specification is to be found in the first line read from the standard input (see `fspec`(4)). If no *tabspec* is given, *tabspec* defaults to `-8`. A *tabspec* of `-0` expects no tabs; if any are found, they are treated as `-1`.

`-o`*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for `-i`*tabspec*. If no *tabspec* is given, *tabspec* defaults to `-8`. A *tabspec* of `-0` means that no spaces will be converted to tabs on output.

`-l`*n* Set the effective line length to *n* characters. If *n* is not entered, `-l` defaults to 72. The default line length without the `-l` option is 80 characters. Note that tabs and backspaces are considered to be one character (use `-i` to expand tabs to spaces).

`-b`*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see `-l`*n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when `-b` with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -l1 -b7 file-name
```

The `-l1` must be used to set the effective line length shorter than any existing line in the file so that the `-b` option is activated.

`-e`*n* Same as `-b`*n* except that characters are truncated from the end of the line.

`-c`*char* Change the prefix/append character to *char*. Default character for *char* is a space.

`-p`*n* Prefix *n* characters (see `-c`*k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

`-a`*n* Same as `-p`*n* except characters are appended to the end of a line.

`-f` Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* `-o` option. If no `-o` option is specified, the line which is printed will contain the default specification of `-8`.

`-s` Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a `*` and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform -s -i -l -a -e filename
```

International Features

`newform` can perform processing of characters from supplementary code sets as well as ASCII characters.

Options:

`-b`*n*	*n* must be in columns, not the number of characters.
`-e`*n*	*n* must be in columns, not the number of characters.
`-p`*n*	*n* must be in columns, not the number of characters.
`-a`*n*	*n* must be in columns, not the number of characters.
`-l`*n*	*n* must be in columns, not the number of characters.

`-c`*char*
The prefix/append character *char* must be one single-byte character.

EXAMPLES

```
$ newform -l31 -b19 -e1 newform_file
```

This command will read the file "newform_file" and reformat it to the given specifications: set output line length to 31 characters and to achieve this, truncate 19 characters from the front of each line and truncate 1 character from the end of each line, then display the result on the standard output. Other options allow for the expanding of tabs and padding of lines to a new length.

DIAGNOSTICS

All diagnostics are fatal.

`usage: . . .`	`Newform` was called with a bad option.
`not -s format`	There was no tab on one line.
`can't open file`	Self-explanatory.
`internal line too long`	A line exceeds 512 characters after being expanded in the internal work buffer.

`tabspec in error`	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
`tabspec indirection illegal`	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 – normal execution
1 – for any error

SEE ALSO

`csplit`(1), `tabs`(1).
`fspec`(4) in the *Programmer's Reference for the DG/UX System*.

BUGS

`Newform` normally only keeps track of physical characters; however, for the `-i` and `-o` options, `newform` will keep track of backspaces in order to line up tabs in the appropriate logical columns.

`Newform` will not prompt the user if a *tabspec* is to be read from the standard input (by use of `-i--` or `-o--`).

If the `-f` option is used, and the last `-o` option specified was `-o--`, and was preceded by either a `-o--` or a `-i--`, the tab specification format line will be incorrect.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [*group*]

DESCRIPTION

Newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by newgrp, regardless of whether it terminated successfully or terminated due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking newgrp; however, all unexported variables are either reset to their default value or set to null. System variables (such as PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than $ (default) and has not exported PS1. After an invocation of newgrp, successful or not, the user's PS1 is set to the default prompt string $. Note that the shell command export (see sh(1)) and the C shell command setenv (see csh(1)) export variables so that they retain their assigned value when invoking new shells.

With no arguments, newgrp changes the group identification back to the group specified in the user's password file entry.

If the first argument to newgrp is a –, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in /etc/group as a member of that group.

FILES

/etc/group	System's group file
/etc/passwd	System's password file

SEE ALSO

login(1), sh(1), csh(1).
group(4), passwd(4), environ(5) in the *Programmer's Reference for the DG/UX System*

BUGS

There is no convenient way to enter a password into /etc/group. Use of group passwords is not encouraged, since they encourage poor security practices. Group passwords may disappear in the future.

NAME

`news` – print news items

SYNOPSIS

`news [ -a ] [ -n ] [ -s ]` [*items*]

DESCRIPTION

`News` keeps the user informed of current events. By convention, these events are described by files in the directory `/usr/news`, to which anyone having access (typically system administrator) to the file can enter news items.

When invoked without arguments, `news` prints the contents of all current files in `/usr/news`, most recent first, with each preceded by an appropriate header. `News` stores the currency time as the modification date of a file named `.news_time` in the user's home directory. The identity of this directory is determined by the environment variable HOME; only files more recent than this currency time are considered current.

The `-a` option causes `news` to print all items, regardless of currency. In this case, the stored time is not changed.

The `-n` option causes `news` to report the names of the current items without printing their contents, and without changing the stored time.

The `-s` option causes `news` to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of `news` in your `.profile` file, or in the system's `/etc/profile`.

All other arguments are assumed to be specific news items that are to be printed.

If you type an *interrupt* during the printing of a news item, printing stops and the next item is started. Another *interrupt* within one second of the first terminates the program.

EXAMPLES

```
news
```

Prints all the news items on the system that you have not already read.

```
news reorg
```

Prints just the news item named `reorg`.

FILES

```
/etc/profile
/usr/news/*
$HOME/.news_time
```

SEE ALSO

`profile`(4), `environ`(5) in the *Programmer's Reference for the DG/UX System.*

NAME

nice – run a command at a higher or lower priority

SYNOPSIS

nice [*-increment*] *command* [*arguments*]

DESCRIPTION

Nice executes *command* with a lower or higher CPU scheduling priority. Scheduling priority numbers are such that the higher the number, the lower the priority. A process at a lower priority will probably execute more slowly than a process with a higer priority. The highest priority level is -20. The lowest is 19.

By default, nice executes *command* at a lower priority (which is your shell's priority plus 10).

If you provide an *increment* argument, the system executes your command at a priority that much higher or lower than your current priority. For example, with an argument of -10, nice will execute your command at your current priority plus 10. With an argument of --10, nice will execute your command at your current priority minus 10. If your shell's normal priority is 0 and you invoke nice with an argument of -10, nice executes your command at a priority of 10.

Only a superuser can raise a process's priority by passing nice a negative number. If a non-superuser invokes nice with a negative number, nice assumes -0 as the argument.

If you attempt to execute a command at a priority greater than 19, nice executes it at 19. If you attempt to execute a command at a priority less than –20, nice executes it at –20. You cannot interrupt a process running at a very high priority (such as –20). To interrupt such a process, use the renice(1) command first to lower the process's priority.

System runtime conditions determine exactly how priority affects execution speed. To change the priority of a process that is already running, use the renice(1) command.

Note that this command is not the same as the built-in version of nice that you get if you use the csh. See csh(1). If you use the csh but prefer to use this version of nice, invoke this command with its full pathname, /usr/bin/nice.

EXAMPLES

```
$ nice -19 who
```

nice will execute the who command at a lower CPU priority (that is, at a priority 19 points lower than your shell's current priority).

```
$ nice --10 who
```

If you are the superuser, nice will execute the who command at a higher (10 points higher) CPU priority. If you are not superuser, nice executes the command at your current priority.

```
$ nice who
```

nice will execute who at a priority that is 10 points lower than your shell's priority.

DIAGNOSTICS

Nice returns the exit status of the subject command.

SEE ALSO

csh(1), nohup(1), renice(1), sh(1), getpriority(2), nice(2), setpriority(2).

NAME

`nl` – line numbering filter

SYNOPSIS

`nl` [`-b`*type*] [`-f`*type*] [`-h`*type*] [`-v`*start#*] [`-i`*incr*] [`-p`] [`-l`*num*] [`-s`*sep*] [`-w`*width*] [`-n`*format*] [`-d`*delim*] [*file*]

DESCRIPTION

`nl` reads lines from the named *file*, or the standard input if no *file* is named, and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

`nl` views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer. For example, `-bt` (the default) numbers non-blank lines in the body section and does not number any lines in the header and footer sections.

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
`\:\:\:`	header
`\:\:`	body
`\:`	footer

Unless optioned otherwise, `nl` assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

`-b`*type* Specifies which logical page body lines are to be numbered. Recognized *type*s and their meanings are:

`a`	number all lines
`t`	number lines with printable text only
`n`	no line numbering
`p`*exp*	number only lines that contain the regular expression specified in *exp* (see `ed`(1))

Default *type* for logical page body is `t` (text lines numbered). All characters from supplementary code sets are considered printable.

`-f`*type* Same as `-b`*type* except for footer. Default *type* for logical page footer is `n` (no lines numbered).

`-h`*type* Same as `-b`*type* except for header. Default *type* for logical page header is `n` (no lines numbered). All characters from supplementary code sets are considered printable.

`-v`*start#* *start#* is the initial value used to number logical page lines. Default *start#* is `1`.

`-i`*incr* *incr* is the increment value used to number logical page lines. Default *incr* is `1`.

`-p`	Do not restart numbering at logical page delimiters.
`-l`*num*	*num* is the number of blank lines to be considered as one. For example, `-l2` results in only the second adjacent blank being numbered (if the appropriate `-ha`, `-ba`, and/or `-fa` option is set). Default *num* is `1`.
`-s`*sep*	*sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab. Characters used for *sep* must be single-byte characters.
`-w`*width*	*width* is the number of characters to be used for the line number. Default *width* is `6`.
`-n`*format*	*format* is the line numbering format. Recognized values are: `ln`, left justified, leading zeroes suppressed; `rn`, right justified, leading zeroes suppressed; `rz`, right justified, leading zeroes kept. Default *format* is `rn` (right justified).
`-d`*delim*	The two delimiter characters specifying the start of a logical page section may be changed from the default characters (`\:`) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the `-d` and the delimiter characters. To enter a backslash, use two backslashes. Characters used for *delim* must be single-byte characters.

International Features

`nl` can process files containing characters from supplementary code sets. Searches are performed on characters, not on bytes.

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will cause the first line of the page body to be numbered `10`, the second line of the page body to be numbered `20`, the third `30`, and so forth. The logical page delimiters are !+.

SEE ALSO

pr(1), ed(1).

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup *command* [*arguments*]

DESCRIPTION

Nohup executes *command* with hangups and quits ignored. If output is not redirected by the user, both standard output and standard error are sent to nohup.out. If nohup.out is not writable in the current directory, output is redirected to $HOME/nohup.out. If standard output is redirected but standard error is not, standard error is automatically redirected to the same file as output.

EXAMPLE

It is frequently desirable to apply nohup to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the nohup applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type sh can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see sh(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
tbl ofile | eqn | nroff > nfile
```

SEE ALSO

chmod(1), csh(1), nice(1), sh(1).
signal(2) in the *Programmer's Reference for the DG/UX System*

NOTES

```
nohup command1; command2
```

nohup applies only to *command1*

```
nohup (command1; command2)
```

is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio -o <list >/dev/rmt/1m&
```

while

```
nohup cpio -o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file errors.

NAME

`notify` – notify user of the arrival of new mail

SYNOPSIS

`notify -y [ -m` *mailfile* `]`
`notify [ -n ]`

DESCRIPTION

When a new mail message arrives, the `mail` command first checks if the recipient's mailbox indicates that the message is to be forwarded elsewhere (to some other recipient or as the input to some command). `notify` is used to set up forwarding on the user's mailbox so that the new message is saved into an alternative mailbox and, if the user is currently logged in, he or she is notified immediately of the arrival of new mail.

Command-line options are:

`-m` *mailfile*	File to save mail messages into while automatic notification is activated. If not specified, it defaults to `$HOME/.mailfile`.
`-n`	Remove mail notification facility
`-y`	Install mail notification facility

If invoked with no arguments, `notify` reports whether automatic mail notification is activated or not.

The notification is done by looking in `/var/adm/utmp` to determine if the recipient is currently logged in, and if so, on which terminal device. Then the terminal device is opened for writing and the user is notified about the new message. The notification will indicate who the message is from. If the message contains a `Subject:` header line it will be included. (For security, all unprintable characters within the header will be converted to an exclamation point.)

If the user is logged in multiple times he or she will get multiple notifications, one per terminal. To disable notifications to a particular login session, the `mesg`(1) command can be used to disable writing to that terminal.

If there are multiple machines connected together via RFS or NFS, `notify` will look up the `/var/adm/utmp` files on the other systems as well. To do this, the file `/etc/mail/notify.sys` will be consulted, which will contain two columns, the first being the name of a system and the second being a path to find the root filesystem for that machine.

If `notify` has troubles delivering the mail to the specified mailfile, `notify` will look up the directory of the mailfile in `/etc/mail/notify.fsys`. If the file's directory is found in the first column of the file, the mail will be forwarded to the system listed in the second column instead of being returned to the sender.

FILES

`/tmp/notif*`	temporary file
`/var/mail/*`	users' standard mailboxes
`/usr/lib/mail/notify2`	program that performs the notification
`/etc/mail/notify.fsys`	list of file systems and home systems
`/etc/mail/notify.sys`	list of machines and paths to their root filesystems
`/var/adm/utmp`	list of users who are logged in

SEE ALSO

`mail`(1), `mesg`(1).
User's Guide.

NOTES

Because `notify` uses the "`Forward to | `*command*" facility of `mail` to implement notifications, `/var/mail/`*username* should not be specified as the place to put newly arrived messages via the `-m` invocation option. The `mail` command uses `/var/mail/`*username* to hold either mail messages, `or` indications of mail forwarding, but not both simultaneously.

If the user is using `layers`(1), the notification will `only` appear in the `login` window.

NAME

oawk – old pattern scanning and processing language

SYNOPSIS

awk [-F*c*] [*prog*] [*parameters*] [*files*]

DESCRIPTION

Although you can still use the oawk utility, it has been superseded by the newer awk utility. See awk(1). Oawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. Each pattern in *prog* can have 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. The -F*c* option specifies *c* as a field separator.

Parameters, in the form x=... y=... etc., may be passed to oawk. The parameters cannot be array elements.

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 the -F*c* option or 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 ; conditional ; 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 enquoted (").

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 (see printf(3S)).

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` (see `grep`(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*

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.

You can use the special patterns `BEGIN` and `END` 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

```
$ oawk "length > 72" infile
```

This form of the command will print only those lines in the file "infile" that are longer than 72 characters.

```
$ oawk '{ s += $3 }
> END  { print "sum is", s, " average is", s/NR }' sales_report
```

This form of the command will add up the third column of the file, "sales_report", and print the sum and average of that sum.

```
$ oawk '{ for (i = NF; i > 0; --i) print $i }' infile
```

This form of the command will print each of the fields in reverse order.

```
$ cat cmd_file
/page/ { $2 = n++; }
       { print }
$ oawk -f cmd_file n=1 report > num_report
```

Using the commands in the file "cmd_file", this form of the command will print file "report", filling in page numbers and outputting file "num_report" which will be the original report with page numbers.

SEE ALSO

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

BUGS

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

od – octal dump

SYNOPSIS

od [-bcDdFfOoSsvXx] [*file*] [[+]*offset*[. | b]]

DESCRIPTION

od displays *file* in one or more formats, as selected by the first argument. If the first argument is missing, -o is default. If no *file* is specified, the standard input is used. For the purposes of this description, "word" refers to a 16-bit unit, independent of the word size of the machine; "long word" refers to a 32-bit unit, and "double long word" refers to a 64-bit unit. The meanings of the format options are:

-b	Interpret bytes in octal.
-c	Interpret bytes as single-byte characters. Certain non-graphic characters appear as C-language escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers (see **EXAMPLES** below).
-D	Interpret long words in unsigned decimal.
-d	Interpret words in unsigned decimal.
-F	Interpret double long words in extended precision.
-f	Interpret long words in floating point.
-O	Interpret long words in unsigned octal.
-o	Interpret words in octal.
-S	Interpret long words in signed decimal.
-s	Interpret words in signed decimal.
-v	Show all data (verbose).
-X	Interpret long words in hex.
-x	Interpret words in hex.

offset specifies an offset from the beginning of *file* where the display will begin. *offset* is normally interpreted as octal bytes. If . is appended, *offset* is interpreted in decimal. If b is appended, *offset* is interpreted in blocks of 512 bytes. If *file* is omitted, *offset* must be preceded by +.

The display continues until an end-of-file is reached.

International Features

od can process characters from supplementary code sets.

The value of the *offset* argument must be specified in bytes.

EXAMPLES

echo "hello world" | od -c

```
0000000  h   e   l   l   o       w   o   r   l   d  \n
0000014
```

This example displays the string "hello world" in ASCII format.

```
$ cat a
test file
$ od -b a
0000000 164 145 163 164 040 146 151 154 145 012
```

```
0000012
```

This example displays the file a on the standard output in byte octal format.

```
$ od -cb a
0000000   t   e   s   t       f   i   l   e  \n
        164 145 163 164 040 146 151 154 145 012
0000012
```

This example displays the file a in ASCII and octal formats.

```
$ od -x a
0000000 7465 7374 2066 696c 650a
0000012
```

This example displays the file a in hexadecimal format.

SEE ALSO

strings(1).

NAME

on – execute a command on a remote system, but with the local environment

SYNOPSIS

on [-i] [-d] [-n] *host command* [*argument*] ...

DESCRIPTION

The on program is used to execute commands on another system, in an environment similar to that invoking the program. All environment variables are passed, and the current working directory is preserved. Starting with dg/ux pass 5.4.1, the umask is passed, and then set on the remote system if it is also dg/ux pass 5.4.1 or higher. To preserve the working directory, the working file system must be either already mounted on the host or be exported to it. Relative path names will only work if they are within the current file system; absolute path names may cause problems.

The standard input is connected to the standard input of the remote command, and the standard output and the standard error from the remote command are sent to the corresponding files for the on command.

The on command uses the simple trusted host authentication that rlogin and remsh use. For details, see hosts.equiv(4)

OPTIONS

-i Interactive mode. Use remote echoing and special character processing. This option is needed for programs that expect to be talking to a terminal. All terminal modes and window size changes are propagated. If you create a symbolic link to on with the name of a host (e.g., ln -s /usr/bin/on yoursys), typing the name of the host is the same as using on with the -i option and the name of the host.

-d Debug mode. Print out some messages as work is being done.

-n No Input. This option causes the remote program to get EOF when it reads from the standard input, instead of passing the standard input from the standard input of the on program. For example, -n is necessary when running commands in the background with job control.

DIAGNOSTICS

unknown host	Host name not found.
cannot connect to server	Host down or not running the server.
can't find	Problem finding the working directory.
can't locate mount point	Problem finding current file system.

Other error messages may be passed back from the server.

SEE ALSO

remsh(1C), rexd(1M), exports(4), hosts.equiv(4).

BUGS

The DG/UX window system can get confused by some environment variables, particularly LINES and COLUMNS.

NAME

`pack`, `pcat`, `unpack` – compress and expand files

SYNOPSIS

`pack [ – ] [ -f ]` *name* ...

`pcat` *name* ...

`unpack` *name* ...

DESCRIPTION

`pack` attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*`.z` with the same access modes, access and modified dates, and owner as those of *name*. The `-f` option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If `pack` is successful, *name* will be removed. Packed files can be restored to their original form using `unpack` or `pcat`.

`pack` uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each `.z` file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

`pack` returns a value that is the number of files that it failed to compress.

No packing will occur if:

> the file appears to be already packed;
> the file name has more than 76 characters;
> the file has links;
> the file is a directory;
> the file cannot be opened;
> no disk storage blocks will be saved by packing;
> a file called *name*`.z` already exists;
> the `.z` file cannot be created;
> an I/O error occurred during processing.

The last segment of the file name must contain no more than 76 characters to allow space for the appended `.z` extension. Directories cannot be compressed.

`pcat` does for packed files what `cat(1)` does for ordinary files, except that `pcat` cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named `name.z` use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say `nnn`, of a packed file named `name.z` (without destroying `name.z`) use the command:

```
pcat name >nnn
```

`pcat` returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the `.z`) has more than 76 characters;
the file cannot be opened;
the file does not appear to be the output of `pack`.

`unpack` expands files created by `pack`. For each file *name* specified in the command, a search is made for a file called *name*`.z` (or just *name*, if *name* ends in `.z`). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the `.z` suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

`Unpack` returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in `pcat`, as well as for the following:

a file with the "unpacked" name already exists;
if the unpacked file cannot be created.

SEE ALSO

`cat(1)`, `compress(1)`.

NAME

`passwd` – change login password

SYNOPSIS

`passwd` [`-f` *filename*] [*uid*]

DESCRIPTION

This command changes (or installs) a password associated with the *uid* (your own by default).

`Passwd` prompts for the old password and then for the new one. You must supply both, and the new password must be typed twice to forestall mistakes.

The new password is checked to ensure that it meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies do not match, then the cycle of prompting for the new password is repeated (at most) two more times.

Passwords must meet the following requirements:

1) Each password must have at least six characters. Only the first eight characters are significant.

2) Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, 'alphabetic' means upper and lowercase letters.

3) Each password must differ from the login name and from any reverse or circular shift of that name. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

4) New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

Anyone whose effective *uid* is zero is called a superuser; see `id`(1) and `su`(1). Superusers may change any password, so `passwd` does not prompt superusers for the old password. Superusers do not have to comply with password construction requirements. A superuser can create a null password by entering a carriage return in response to the prompt for a new password.

Options

`-f` Treat *file* as the password file. By default, `/etc/passwd` is used.

FILES

`/etc/passwd`

SEE ALSO

`login`(1), `yppasswd`(1) `crypt`(3C), `passwd`(4),
Robert Morris and Ken Thompson, *UNIX Password Security*

NOTES

`Passwd` will not change your password if it is stored by the Network Information Service (NIS). Refer to `yppasswd`(1) for more information.

NAME

`paste` – merge lines

SYNOPSIS

`paste` *file1 file2 ...*
`paste -d`*list file1 file2 ...*
`paste -s [-d`*list*`]` *file1 file2 ...*

where:

file1	The first input file
file2	The second input file
list	One or more characters

DESCRIPTION

In the first two forms, `paste` concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of `cat`(1), which concatenates vertically, i.e., one file after the other. In its last form, `paste` replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the tab character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a filename.

Options

`-d` Replace the default line concatenation character with the characters specified in *list*. Without this option, the new-line characters of each but the last file (or last line in case of the `-s` option) are replaced by a tab character. The list is used circularly; i.e., when exhausted, it is reused. In parallel merging (i.e., no `-s` option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: `\n` (new-line), `\t` (tab), `\\` (backslash), and `\0` (empty string, not a null character). You may need to enquote characters if they have special meaning to the shell (e.g., to get one backslash, use `-d "\\\\"`).

`-s` Merge subsequent lines rather than one from each input file. The last character of the file is forced to be a new-line. Use tab for concatenation, unless a *list* is specified with `-d` option.

`-` May be used in place of any filename, to read a line from the standard input (there is no prompting).

International Features

`Paste` can process characters from supplementary code sets as well as ASCII characters.

With the `-d` option, characters from supplementary code sets can be specified for *list*.

EXAMPLES

`ls | paste -d" " -` List directory in one column.

`ls | paste - - - -` List directory in four columns.

`paste -s -d"\t\n" file`
Combine pairs of lines into lines.

DIAGNOSTICS

`line too long`	Output lines are restricted to 511 characters.
`too many files`	Except for the `-s` option, no more than 12 input files may be specified.

SEE ALSO

`cut`(1), `grep`(1), `pr`(1).

NAME

pg – display file forward or backward one screenful at a time

SYNOPSIS

pg [*-number*] [-p *string*] [-cefns] [+ *linenumber*] [+*/pattern/*] [*files* ...]

DESCRIPTION

The pg command is a filter which allows the examination of *files* one screenful at a time on a CRT terminal. (The file name – and/or NULL arguments indicate that pg should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, pg scans the terminfo(4) data base for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal type dumb is assumed.

The command line options are:

–number
: An integer specifying the size (in lines) of the window that pg is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

–p *string*
: Causes pg to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

–c
: Home the cursor and clear the screen before displaying each page. This option is ignored if clear_screen is not defined for your terminal type in the terminfo(4) data base.

–e
: Causes pg *not* to pause at the end of each file.

–f
: Normally, pg splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The –f option inhibits pg from splitting lines.

–n
: Normally, commands must be terminated by a newline character. This option causes an automatic end of command as soon as a command letter is entered.

–s
: Causes pg to print all messages and prompts in standout mode (usually reverse video).

+linenumber
: Start up at *linenumber*.

+/pattern/
: Start up at the first line containing the regular expression *pattern*.

The responses that may be typed when pg pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed.

This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided; it is shown in parentheses below. Control characters are indicated with a caret, e.g., `^c` for Control-C.

The perusal commands and their defaults are as follows:

`(+1)<newline>` or `<space>`
: This causes one page to be displayed. The address is specified in pages.

`(+1) l`
: With a relative address this causes `pg` to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

`(+1) d` or `^D`
: Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

`.` or `^L`
: Typing a single period causes the current page of text to be redisplayed.

`$`
: Displays the last windowful in the file. Use with caution when the input is a pipe because `pg` will wait until the process writing to the pipe is completely finished before displaying anything.

The following commands are available for searching for text patterns in the text. The regular expressions described in `ed`(1) are available. They must always be terminated by a newline, even if the `-n` option is specified.

[*i*]/*pattern*/
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

[*i*]^*pattern*^
[*i*]?*pattern*?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the `?`.

After searching, `pg` will normally display a screen of text with the found line at the top of the screen. This can be modified by appending `m` or `b` to the search command to position the found line in the middle or at the bottom of the window from now on. The suffix `t` can be used to restore the original situation.

The user of `pg` can modify the environment of perusal with the following commands:

[*i*]n
: Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, and its default value is 1.

[*i*]p
: Begin perusing the *i*th previous file in the command line. The prompt reads `Next File`. The *i* is an unsigned number, and its default is 1.

[*i*]w
: Display another window of text. If *i* is present, set the window size to *i*. (The actual size will be *i*−1).

`s` *filename*
: Save the input in the named file. Only the current file being perused is saved.

The white space between the `s` and *filename* is optional. This command must always be terminated by a newline, even if the `-n` option is specified.

`h` Provide help by displaying an abbreviated summary of available commands.

`q` or `Q` Quit `pg`.

`!`*command*
Command is passed to the shell, whose name is taken from the `SHELL` environment variable. If this is not available, the default shell `/bin/sh` is used. This command must always be terminated by a newline, even if the `-n` option is specified.

At any time when output is being sent to the terminal, the user can type the quit key (normally control-\) or the interrupt (break) key to interrupt the display. This causes `pg` to stop sending output and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a tty device, then `pg` acts just like `cat`(1), except that a header is printed before each file if there is more than one.

International Features

`pg` can process files containing characters from supplementary code sets. Searches are performed on characters, not on individual bytes.

`pg` lines longer than lines on the screen at characters.

The prompt *string* with option `-p` can include characters from supplementary code sets.

EXAMPLE

A sample usage of `pg` in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, `pg` responds to interrupt and quit characters by terminating execution. Between prompts, however, these signals interrupt `pg`'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of `more`(1) will find that the z and f commands are available, and that the terminating `/`, `^`, or `?` character may be omitted from the searching commands.

FILES

`/usr/lib/terminfo/?/*`	Terminal information data base
`/tmp/pg*`	Temporary file when input is from a pipe

SEE ALSO

`crypt`(1), `ed`(1), `more`(1), `cat`(1).
`terminfo`(4) in the *Programmer's Reference for the DG/UX System*

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using `pg` as a filter with another command that changes the terminal I/O options (e.g., `crypt`(1)), terminal settings may not be restored correctly.

NAME

`pkginfo` – display software package information

SYNOPSIS

`pkginfo` [`-q`|x|`l`] [`-p`|`i`] [`-a` *arch*] [`-v` *version*] [`-c` *category1*`,`[*category2*[`, ...`]]] [*pkginst*[`,`*pkginst*[`, ...`]]]

`pkginfo` [`-d` *device* [`-q`|x|l] [`-a` *arch*] [-v *version*] [`-c` *category1*`,`[*category2*[`, ...`]]] [*pkginst*[`,`*pkginst*[`, ...`]]]

DESCRIPTION

`pkginfo` displays information about software packages which are installed on the system (with the first synopsis) or which reside on a particular device or directory (with the second synopsis). Only the package name and abbreviation for pre-SVR4 packages will be included in the display.

The options for this command are:

`-q`	Does not list any information, but can be used from a program to check (*i.e.*, query) whether or not a package has been installed.
`-x`	Designates an extracted listing of package information. It contains the package abbreviation, package name, package architecture (if available) and package version (if available).
`-l`	Designates long format, which includes all available information about the designated package(s).
`-p`	Designates that information should be presented only for partially installed packages.
`-i`	Designates that information should be presented only for fully installed packages.
`-a`	Specifies the architecture of the package as *arch*.
`-v`	Specifies the version of the package as *version*. "All compatible versions" can be requested by preceding the version name with a tilde (˜). Multiple white space is replaced with a single space during version comparison.
`-c`	Selects packages to be display based on the category *category*. (Categories are defined in the category field of the `pkginfo` file.) If more than one category is supplied, the package must only match one of the list of categories. The match is not case specific.
pkginst	Designates a package by its instance. An instance can be the package abbreviation or a specific instance (for example, `inst.1` or `inst.beta`). All instances of package can be requested by `inst.*`.
`-d`	Defines a device, *device*, on which the software resides. *device* can be a directory pathname or the identifiers for tape, floppy disk, removable disk, *etc.* The special token "spool" may be used to indicate the default installation spool directory.

SEE ALSO

`pkgadd`(1M), `pkgask`(1M), `pkgchk`(1M), `pkgrm`(1M), `pkgtrans`(1).

NOTES

Without options, `pkginfo` lists the primary category, package instance, and name of all completely installed and partially installed packages. One line per package selected is produced.

The `-p` and `-i` options are meaningless if used in conjunction with the `-d` option.

The options `-q`, `-x`, and `-l` are mutually exclusive.

`pkginfo` cannot tell if a pre-SVR4 package is only partially installed. It is assumed that all pre-SVR4 packages are fully installed.

NAME

pkgmk – produce an installable package

SYNOPSIS

pkgmk [-o] [-d *device*] [-r *rootpath*] [-b *basdir*] [-l *limit*] [-a *arch*] [-v *version*] [-p *pstamp*] [-f *prototype*] [*variable=value* . . .] [*pkginst*]

DESCRIPTION

pkgmk produces an installable package to be used as input to the pkgadd command. The package contents will be in directory structure format.

The command uses the package prototype file as input and creates a pkgmap file. The contents for each entry in the prototype file is copied to the appropriate output location. Information concerning the contents (checksum, file size, modification date) is computed and stored in the pkgmap file, along with attribute information specified in the prototype file.

-o	Overwrites the same instance, package instance will be overwritten if it already exists.
-d·	Creates the package on *device*. *device* can be a directory pathname or the identifiers for a floppy disk or removable disk (for example, /dev/diskette). The default device is the installation spool directory.
-r	Ignores destination paths in the prototype file. Instead, uses the indicated *rootpath* with the source pathname appended to locate objects on the source machine.
-b	Prepends the indicated *basedir* to locate relocatable objects on the source machine.
-l	Specifies the maximum size in 512 byte blocks of the output device as *limit*. By default, if the output file is a directory or a mountable device, pkgmk will employ the df command to dynamically calculate the amount of available space on the output device. Useful in conjunction with pkgtrans to create package with datastream format.
-a	Overrides the architecture information provided in the pkginfo file with *arch*.
-v	Overrides version information provided in the pkginfo file with *version*.
-p	Overrides the production stamp definition in the pkginfo file with *pstamp*.
-f	Uses the file prototype as input to the command. The default prototype filename is [Pp]rototype.
variable=value	Places the indicated variable in the packaging environment. [See prototype(4) for definitions of packaging variables.]
pkginst	Specifies the package by its instance. An instance can be the package abbreviation or a specific instance (for example, inst.1).

SEE ALSO

pkgparam(1), pkgproto(1), pkgtrans(1).

NOTES

Architecture information is provided on the command line with the -a option or in the prototype file. If no architecture information is supplied at all, the output of `uname -m` will be used.

Version information is provided on the command line with the `-v` option or in the `prototype` file. If no version information is supplied, a default based on the current date will be provided.

Command line definitions for both architecture and version override the `prototype` definitions.

NAME

pkgparam – displays package parameter values

SYNOPSIS

pkgparam [-v][-d *device*] *pkginst* [*param*[...]]
pkgparam -f *file* [-v] [*param*[...]]

DESCRIPTION

pkgparam displays the value associated with the parameter or parameters requested on the command line. The values are located in either the pkginfo file for *pkginst* or from the specific file named with the -f option.

One parameter value is shown per line. Only the value of a parameter is given unless the -v option is used. With this option, the output of the command is in this format:

parameter1='value1'
parameter2='value2'
parameter3='value3'

If no parameters are specified on the command line, values for all parameters associated with the package are shown.

Options and arguments for this command are:

-v	Specifies verbose mode. Displays name of parameter and its value.
-d	Specifies the *device* on which a *pkginst* is stored. It can be a directory pathname or the identifiers for tape, floppy disk or removable disk (for example, /var/tmp, /dev/diskette, and /dev/dsk/c1d0s0). The default device is the installation spool directory. If no instance name is given, parameter information for all packages residing in *device* is shown.
-f	Requests that the command read *file* for parameter values.
pkginst	Defines a specific package instance for which parameter values should be displayed. The format *pkginst*.* can be used to indicate all instances of a package.
param	Defines a specific parameter whose value should be displayed.

DIAGNOSTICS

If parameter information is not available for the indicated package, the command exits with a non-zero status.

SEE ALSO

pkgmk(1), pkgproto(1), pgktrans(1), pkgparam(3X).

NOTES

The -f synopsis allows you to specify the file from which parameter values should be extracted. This file should be in the same format as a pkginfo file. As an example, such a file might be created during package development and used while testing software during this stage.

NAME

pkgproto – generate a prototype file

SYNOPSIS

pkgproto [-i] [-c *class*] [*path1*[=*path2*] . . .]

DESCRIPTION

pkgproto scans the indicated paths and generates a prototype file that may be used as input to the pkgmk command.

-i	Ignores symbolic links and records the paths as ftype=f (a file) versus ftype=s(symbolic link)
-c	Maps the class of all paths to *class*.
path1	Pathname where objects are located.
path2	Pathname which should be substituted on output for *path1*.

If no paths are specified on the command line, standard input is assumed to be a list of paths. If the pathname listed on the command line is a directory, the contents of the directory is searched. However, if input is read from stdin, a directory specified as a pathname will not be searched.

EXAMPLE

The following two examples show uses of pkgproto and a parial listing of the output produced.

Example 1:

```
$ pkgproto /usr/bin=bin /usr/usr/bin=usrbin /etc=etc
f none bin/sed=/bin/sed 0775 bin bin
f none bin/sh=/bin/sh 0755 bin daemon
f none bin/sort=/bin/sort 0755 bin bin
f none usrbin/sdb=/usr/bin/sdb 0775 bin bin
f none usrbin/shl=/usr/bin/shl 4755 bin bin
d none etc/master.d 0755 root daemon
f none etc/master.d/kernel=/etc/master.d/kernel 0644 root daemon
f none etc/rc=/etc/rc 0744 root daemon
```

Example 2:

```
$ find / -type d -print | pkgproto
d none / 755 root root
d none /usr/bin 755 bin bin
d none /usr 755 root root
d none /usr/bin 775 bin bin
d none /etc 755 root root
d none /tmp 777 root root
```

SEE ALSO

pkgmk(1), pkgparam(1), pkgtrans(1).

NOTES

By default, pkgproto creates symbolic link entries for any symbolic link encountered (ftype=s). When you use the -i option, pkgproto creates a file entry for symbolic links (ftype=f). The prototype file would have to be edited to assign such file types as "v" (volatile), "e" (editable), or "x" (exclusive directory). pkgproto detects linked files. If multiple files are linked together, the first path encountered is considered the source of the link.

NAME

pkgtrans – translate package format

SYNOPSIS

pkgtrans [-ions] *device1 device2* [*pkginst1* [*pkginst2* [. . .]]]

DESCRIPTION

pkgtrans translates an installable package from one format to another. It translates:

a file system format to a datastream

a datastream to a file system format

a file system format to another file system format

The options and arguments for this command are:

-i	Copies only the pkginfo and pkgmap files.
-o	Overwrites the same instance on the destination device, package instance will be overwritten if it already exists.
-n	Creates a new instance if any instance of this package already exists.
-s	Indicates that the package should be written to *device2* as a datastream rather than as a file system. The default behavior is to write a file system format on devices that support both formats.
device1	Indicates the source device. The package or packages on this device will be translated and placed on *device2*.
device2	Indicates the destination device. Translated packages will be placed on this device.
pkginst	Specifies which package instance or instances on *device1* should be translated. The token all may be used to indicate all packages. *pkginst*.* can be used to indicate all instances of a package. If no packages are defined, a prompt shows all packages on the device and asks which to translate.

EXAMPLE

The following example translates all packages on the floppy drive /dev/diskette and places the translations on /tmp.

```
pkgtrans /dev/diskette /tmp all
```

The next example translates packages pkg1 and pkg2 on /tmp and places their translations (i.e., a datastream) on the 9track1 output device.

```
pkgtrans /tmp 9track1 pkg1 pkg2
```

The next example translates pkg1 and pkg2 on tmp and places them on the diskette in a datastream format.

```
pkgtrans -s /tmp /dev/diskette pkg1 pkg2
```

SEE ALSO

installf(1M), pkgadd(1M), pkgask(1M), pkginfo(1), pkgmk(1), pkgparam(1), pkgproto(1), pkgrm(1M), removef(1M).

NOTES

Device specifications can be either the special node name (/dev/diskette) or the device alias (diskette1). The device spool indicates the default spool directory. Source and destination devices may not be the same.

By default, `pkgtrans` will not transfer any instance of a package if any instance of that package already exists on the destination device. Use of the `-n` option will create a new instance if an instance of this package already exists. Use of the `-o` option will overwrite the same instance if it already exists. Neither of these options are useful if the destination device is a datastream.

NAME

`postdaisy` – PostScript translator for Diablo 630 files

SYNOPSIS

`/usr/lib/lp/postscript/postdaisy` [*options*] [*files*]

DESCRIPTION

The `postdaisy` filter translates Diablo 630 daisy-wheel *files* 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.
`-f` *name*	Print *files* using font *name*. Any PostScript font can be used, although the best results will be obtained only with constant-width fonts. The default font is Courier.
`-h` *num*	Set the initial horizontal motion index to *num*. Determines the character advance and the default point size, unless the `-s` option is used. The default is 12.
`-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 pages whose 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.
`-r` *num*	Selects carriage return and line feed behavior. If *num* is 1, a line feed generates a carriage return. If *num* is 2, a carriage return generates a line feed. Setting *num* to 3 enables both modes.
`-s` *num*	Use point size *num* instead of the default value set by the initial horizontal motion index.
`-v` *num*	Set the initial vertical motion index to *num*. The default is 8.
`-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 0.25 inches.
`-y` *num*	Translate the origin *num* inches along the positive y axis. Positive *num* moves text up the page. The default offset is –0.25 inches.

DIAGNOSTICS

An exit status of 0 is returned if *files* were successfully processed.

FILES

```
/usr/lib/postscript/postdaisy.ps
/usr/lib/postscript/forms.ps
/usr/lib/postscript/ps.requests
```

SEE ALSO

`download`(1), `dpost`(1), `postdmd`(1), `postio`(1), `postmd`(1), `postprint`(1), `postreverse`(1), `posttek`(1).

NAME

postdmd – PostScript translator for DMD bitmap files

SYNOPSIS

/usr/lib/lp/postscript/postdmd [*options*] [*files*]

DESCRIPTION

postdmd translates DMD bitmap *files,* as produced by *dmdps,* or *files* written in the Ninth Edition bitfile(9.5) format 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:

-b *num*	Pack the bitmap in the output file using *num* byte patterns. A value of 0 turns off all packing of the output file. By default, *num* is 6.
-c *num*	Print *num* copies of each page. By default only one copy is printed.
-f	Flip the sense of the bits in *files* before printing the bitmaps.
-m *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin, which by default is located at the center 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 pages whose 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.
-x *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up 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 everything up the page. The default offset is 0.

Only one bitmap is printed on each logical page, and each of the input *files* must contain complete descriptions of at least one bitmap. Decreasing the pattern size using the -b option may help throughput on printers with fast processors (such as PS-810s), while increasing the pattern size will often be the right move on older models (such as PS-800s).

DIAGNOSTICS

An exit status of 0 is returned if *files* were successfully processed.

FILES

/usr/lib/postscript/postdmd.ps
/usr/lib/postscript/forms.ps
/usr/lib/postscript/ps.requests

SEE ALSO

download(1), dpost(1), postdaisy(1), postio(1), postmd(1), postprint(1), postreverse(1), posttek(1).

NAME

`postio` – serial interface for PostScript printers

SYNOPSIS

`/usr/lib/lp/postscript/postio -l` *line* [*options*] [*files*]

DESCRIPTION

`postio` sends *files* to the PostScript printer attached to *line*. If no *files* are specified the standard input is sent. The first group of *options* should be sufficient for most applications:

`-b` *speed*	Transmit data over *line* at baud rate *speed*. Recognized baud rates are 1200, 2400, 4800, 9600, and 19200. The default *speed* is 9600 baud.
`-l` *line*	Connect to the printer attached to *line*. In most cases there is no default and `postio` must be able to read and write *line*. If the *line* doesn't begin with a `/` it may be treated as a Datakit destination.
`-q`	Prevents status queries while *files* are being sent to the printer. When status queries are disabled a dummy message is appended to the log file before each block is transmitted.
`-B` *num*	Set the internal buffer size for reading and writing *files* to *num* bytes. By default *num* is 2048 bytes.
`-D`	Enable debug mode. Guarantees that everything read on *line* will be added to the log file (standard error by default).
`-L` *file*	Data received on *line* gets put in *file*. The default log *file* is standard error. Printer or status messages that don't show a change in state are not normally written to *file* but can be forced out using the `-D` option.
`-P` *string*	Send *string* to the printer before any of the input files. The default *string* is simple PostScript code that disables timeouts.
`-R` *num*	Run *postio* as a single process if *num* is 1 or as separate read and write processes if *num* is 2. By default `postio` runs as a single process.

The next two *options* are provided for users who expect to run `postio` on their own. Neither is suitable for use in spooler interface programs:

`-i`	Run the program in interactive mode. Any *files* are sent first and followed by the standard input. Forces separate read and write processes and overrides many other options. To exit interactive mode use your interrupt or quit character. To get a friendly interactive connection with the printer type `executive` on a line by itself.
`-t`	Data received on *line* and not recognized as printer or status information is written to the standard output. Forces separate read and write processes. Convenient if you have a PostScript program that will be returning useful data to the host.

The last option is not generally recommended and should only be used if all else fails to provide a reliable connection:

`-S`	Slow the transmission of data to the printer. Severely limits throughput, runs as a single process, disables the `-q` option, limits the internal buffer size to 1024 bytes, can use an excessive amount of CPU time, and does nothing in interactive mode.

The best performance will usually be obtained by using a large internal buffer (the `-B` option) and by running the program as separate read and write processes (the `-R 2` option). Inability to fork the additional process causes `postio` to continue as a single read/write process. When one process is used, only data sent to the printer is flow controlled.

The *options* are not all mutually exclusive. The `-i` option always wins, selecting its own settings for whatever is needed to run interactive mode, independent of anything else found on the command line. Interactive mode runs as separate read and write processes and few of the other *options* accomplish anything in the presence of the `-i` option. The `-t` option needs a reliable two way connection to the printer and therefore tries to force separate read and write processes. The `-S` option relies on the status query mechanism, so `-q` is disabled and the program runs as a single process.

In most cases `postio` starts by making a connection to *line* and then attempts to force the printer into the IDLE state by sending an appropriate sequence of `^T` (status query), `^C` (interrupt), and `^D` (end of job) characters. When the printer goes IDLE, *files* are transmitted along with an occasional `^T` (unless the `-q` option was used). After all the *files* are sent the program waits until it's reasonably sure the job is complete. Printer generated error messages received at any time except while establishing the initial connection (or when running interactive mode) cause `postio` to exit with a non-zero status. In addition to being added to the log file, printer error messages are also echoed to standard error.

EXAMPLES

Run as a single process at 9600 baud and send *file1* and *file2* to the printer attached to `/dev/tty01`:

```
postio -l /dev/tty01 file1 file2
```

Same as above except two processes are used, the internal buffer is set to 4096 bytes, and data returned by the printer gets put in file *log*:

```
postio -R2 -B4096 -l/dev/tty01 -Llog file1 file2
```

Establish an interactive connection with the printer at Datakit destination *my/printer*:

```
postio -i -l my/printer
```

Send file program to the printer connected to `/dev/tty22`, recover any data in file results, and put log messages in file *log*:

```
postio -t -l /dev/tty22 -L log program >results
```

NOTES

The input *files* are handled as a single PostScript job. Sending several different jobs, each with their own internal end of job mark (`^D`) is not guaranteed to work properly. `postio` may quit before all the jobs have completed and could be restarted before the last one finishes.

All the capabilities described above may not be available on every machine or even across the different versions of the UNIX system that are currently supported by the program. For example, the code needed to connect to a Datakit destination may work only on System V and may require that the DKHOST software package be available at compile time.

There may be no default *line*, so using the `-l` option is strongly recommended. If omitted, `postio` may attempt to connect to the printer using the standard output. If Datakit is involved, the `-b` option may be ineffective and attempts by `postio` to impose flow control over data in both directions may not work. The `-q` option can help if the printer is connected to RADIAN. The `-S` option is not generally recommended and should be used only if all other attempts to establish a reliable connection fail.

DIAGNOSTICS

An exit status of 0 is returned if the files ran successfully. System errors (such as an inability to open the line) set the low order bit in the exit status, while PostScript errors set bit 1. An exit status of 2 usually means the printer detected a PostScript error in the input *files*.

SEE ALSO

`download`(1), `dpost`(1), `postdaisy`(1), `postdmd`(1), `postmd`(1), `postprint`(1), `postreverse`(1), `posttek`(1).

NAME

`postmd` – matrix display program for PostScript printers

SYNOPSIS

`/usr/lib/lp/postscript/postmd` [*options*] [*files*]

DESCRIPTION

The `postmd` filter reads a series of floating point numbers from *files,* translates them into a PostScript gray scale image, and writes the results on the standard output. In a typical application the numbers might be the elements of a large matrix, written in row major order, while the printed image could help locate patterns in the matrix. If no *files* are specified, or if – is one of the input *files,* the standard input is read. The following *options* are understood:

`-b` *num*	Pack the bitmap in the output file using *num* byte patterns. A value of 0 turns off all packing of the output file. By default, *num* is 6.
`-c` *num*	Print *num* copies of each page. By default, only one copy is printed.
`-d` *dimen*	Sets the default matrix dimensions for all input *files* to *dimen*. The *dimen* string can be given as rows or rows`x`columns. If *columns* is omitted it will be set to rows. By default, `postmd` assumes each matrix is square and sets the number of rows and columns to the square root of the number of elements in each input file.
`-g` *list*	*List* is a comma or space separated string of integers, each lying between 0 and 255 inclusive, that assigns PostScript gray scales to the regions of the real line selected by the `-i` option. 255 corresponds to white, and 0, to black. The `postmd` filter assigns a default gray scale that omits white (that is, 255) and gets darker as the regions move from left to right along the real line.
`-i` *list*	*List* is a comma, space or slash(/) separated string of *N* floating point numbers that partition the real line into 2*N*+1 regions. The *list* must be given in increasing numerical order. The partitions are used to map floating point numbers read from the input *files* into gray scale integers that are either assigned automatically by `postmd` or arbitrarily selected using the `-g` option. The default interval *list* is `-1,0,1`, which partions the real line into seven regions.
`-m` *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin which, by default, is located at the center 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 pages whose 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` *window*	*Window* is a comma or space separated list of four positive integers that select the upper left and lower right corners of a submatrix from each of the input *files*. Row and column indices start at 1 in the upper left corner and the numbers in the input *files* are assumed to be written in row major order. By default, the entire matrix is displayed.

`-x` *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up 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 everything up the page. The default offset is 0.

Only one matrix is displayed on each logical page, and each of the input *files* must contain complete descriptions of exactly one matrix. Matrix elements are floating point numbers arranged in row major order in each input file. White space, including newlines, is not used to determine matrix dimensions. By default, `postmd` assumes each matrix is square and sets the number of rows and columns to the square root of the number of elements in the input file. Supplying default dimensions on the command line with the `-d` option overrides this default behavior, and in that case the dimensions apply to all input *files*.

An optional header can be supplied with each input file and is used to set the matrix dimensions, the partition of the real line, the gray scale map, and a window into the matrix. The header consists of keyword/value pairs, each on a separate line. It begins on the first line of each input file and ends with the first unrecognized string, which should be the first matrix element. Values set in the header take precedence, but apply only to the current input file. Recognized header keywords are `dimension`, `interval`, `grayscale`, and `window`. The syntax of the value string that follows each keyword parallels what's accepted by the `-d`, `-i`, `-g`, and `-w` options.

EXAMPLES

For example, suppose file initially contains the 1000 numbers in a 20x50 matrix. Then you can produce exactly the same output by completing three steps. First, issue the following command line:

```
postmd -d20x50 -i"-100 100" -g0,128,254,128,0 file
```

Second, prepend the following header to *file*:

```
dimension 20x50
    interval   -100.0 .100e+3
    grayscale 0 128 254 128 0
```

Third, issue the following command line:

```
postmd file
```

The interval list partitions the real line into five regions and the gray scale list maps numbers less than −100 or greater than 100 into 0 (that is, black), numbers equal to −100 or 100 into 128 (that is, 50 percent black), and numbers between −100 and 100 into 254 (that is, almost white).

NOTES

The largest matrix that can be adequately displayed is a function of the interval and gray scale lists, the printer resolution, and the paper size. A 600x600 matrix is an optimistic upper bound for a two element interval list (that is, five regions) using 8.5x11 inch paper on a 300 dpi printer.

Using white (that is, 255) in a gray scale list is not recommended and won't show up in the legend and bar graph that `postmd` displays below each image.

DIAGNOSTICS

An exit status of 0 is returned if *files* were successfully processed.

FILES

/usr/lib/postscript/postmd.ps
/usr/lib/postscript/forms.ps
/usr/lib/postscript/ps.requests

SEE ALSO

dpost(1), postdaisy(1), postdmd(1), postio(1), postprint(1), posreverse(1), posttek(1).

NAME

postplot – PostScript translator for plot(4) graphics files

SYNOPSIS

/usr/lib/lp/postscript/postplot [*options*] [*files*]

DESCRIPTION

The postplot filter translates plot(4) graphics *files* 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.
-f *name*	Print text using font *name*. Any PostScript font can be used, although the best results will be obtained only with constant width fonts. The default font is Courier.
-m *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin which, by default, is located at the center 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 pages whose 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 landscape.
-w *num*	Set the line width used for graphics to *num* points, where a point is approximately 1/72 of an inch. By default, *num* is set to 0 points, which forces lines to be one pixel wide.
-x *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up the page. Positive *num* moves everything right. The default offset is 0.0 inches.
-y *num*	Translate the origin *num* inches along the positive y axis. Positive *num* moves everything up the page. The default offset is 0.0.

DIAGNOSTICS

An exit status of 0 is returned if *files* were successfully processed.

NOTES

The default line width is too small for write-white print engines, such as the one used by the PS-2400.

FILES

/usr/lib/postscript/postplot.ps
/usr/lib/postscript/forms.ps
/usr/lib/postscript/ps.requests

SEE ALSO

download(1), dpost(1), postdaisy(1), postdmd(1), postio(1), postmd(1), postprint(1), postreverse(1), plot(4).

NAME

`postprint` – translate text files into PostScript

SYNOPSIS

`/usr/lib/lp/postscript/postprint` [`-c` *int*] [`-f` *name*] [`-l` *int*] [`-m` *num*] [`-n` *int*] [`-o` *list*] [`-p` *mode*] [`-r` *int*] [`-s` *num*] [`-t` *int*] [`-x` *num*] [`-y` *num*] [`-C` *file*] [`-L` *file*] [`-P` *string*] [`-R` *action*] [`-F` *dir*] [`-T` *dir*] [`-S` *file*] [*file*] ...

where:

int	An integral number.
num	A real (floating-point) number.
name	A font name.
list	A list of page numbers.
mode	`p` (portrait) or `l` (landscape).
string	An arbitrary PostScript string of text.
action	A special PostScript request name.
dir	A font or device directory.
file	A PostScript input file.

DESCRIPTION

The `postprint` filter translates each text *file* into PostScript and writes the results on the standard output. If no *file* is specified, or if – is given as one of the *file* arguments, the standard input is read.

Options are:

`-c` *int*	Print *int* copies of each page. By default, only one copy is printed.
`-f` *name*	Print *files* using font *name*. Any PostScript font can be used, although the best results will be obtained only with constant width fonts. The default font is Courier.
`-l` *int*	Set the length of a page to *int* lines. The default length is 66. Setting *int* to 0 is allowed, and will cause *postprint* to estimate a value based on the point size being used.
`-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` *int*	Print *int* logical pages on each piece of paper. By default, only one page is printed on each piece of paper.
`-o` *list*	Print those pages given in the comma-separated *list* of page numbers. The *list* may contain both single numbers and ranges in the form *N1–N2*. A missing *N1* means the lowest numbered page; a missing *N2* means the highest.
`-p` *mode*	Print the files in either `portrait` or `landscape` *mode*. Only the first character of *mode* is significant. The default *mode* is `portrait`.
`-r` *int*	Select carriage return behavior. Carriage returns are ignored if *int* is 0, cause a return to column 1 if *int* is 1, and generate a newline if *int* is 2. The default behavior is a return (`-r1`).
`-s` *num*	Print *files* using point size *num*. When printing in landscape mode *num* is scaled by a factor that depends on the imaging area of the device. The default size for portrait mode is 10.
`-t` *int*	Assume tabs are set every *int* columns, starting with the first column. By default, tabs are set every 8 columns.

`-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 0.25 inches.
`-y` *num*	Translate the origin *num* inches along the positive Y axis. Positive *num* moves text up the page. The default offset is −0.25 inches.
`-C` *file*	Copy *file* to the output file. *file* follows the prologue but precedes any job initialization commands. *file* becomes part of the job's global environment and must contain legitimate PostScript commands.
`-L` *file*	Use *file* as the PostScript prologue. The default prologue is `/usr/lib/lp/postscript/postprint.ps`.
`-P` *string*	Add *string* to the output file. *string* follows the prologue but precedes any job initialization commands. *string* becomes part of the job's global environment and must be legitimate PostScript.
`-R` *action*	Request a special *action* (i.e., `manualfeed` or `ledgertray`) on a per page or global basis. The *action* string has the following format: *request*[:*page*[:*file*]] If *page* is omitted or given as 0 the request applies to all pages. If *file* is omitted the request lookup is done in `/usr/lib/lp/postscript/ps.requests`. The collection of recognized requests can be modified or extended by defining a private requests *file*, using the same format as defined in the default file. Multiple occurrences of the **−R** option are supported.

A new logical page is started after 66 lines (or the number of lines specified by the `-l` option) have been printed on the current page. A new logical page is also started whenever an ASCII form feed character is read. Unprintable characters are ignored, and lines that are too long are silently truncated by the printer.

International Features

`postprint` can print documents of various languages. This feature uses PostScript font definition files of the form:

fontdir/devicedir/fontfile

Each of *fontdir*, *devicedir*, and *fontfile* can be set using the options described below.

Options are:

`-F` *fontdir*
: Find the code set definition under font directory *fontdir*. The default font directory is `/usr/lib/font`.

`-T` *devicedir*
: Find the code set definition in device directory `dev`*devicedir*. The default device directory is `devpost`.

`-S` *fontfile*
: Find the code set definition in file *fontfile*, which contains a font definition that implements the code set desired.

The following code set *fontfile*s are provided:

ascii	US English	(default)
ps88591	ISO 8859-1	(Latin 1)

ps88592	ISO 8859-2	(Latin 2)
ps88597	ISO 8859-7	(Latin/Greek)

EXAMPLES

To print `file1` and `file2` in landscape mode, issue the following command:

```
postprint -pland file1 file2
```

To print three logical pages on each physical page in portrait mode:

`postprint -n3` *file*

To print a document using the ISO 8859-1 Western European coded character set:

`postprint -Sps88591` *file*

DIAGNOSTICS

An exit status of 0 is returned if each *file* was successfully processed.

FILES

`/usr/lib/lp/postscript/postprint.ps`	PostScript prologue
`/usr/lib/lp/postscript/forms.ps`	Logical page functions
`/usr/lib/lp/postscript/ps.requests`	Request definitions

SEE ALSO

`download`(1), `dpost`(1), `postdaisy`(1), `postdmd`(1), `postio`(1), `postmd`(1), `postreverse`(1), `posttek`(1).

NOTES

Underlining, boldface, and other forms of overstriking may not work if the carriage return behavior is changed from its default.

The `-S` option is a Data General extension.

NAME

postreverse – reverse the page order in a PostScript file

SYNOPSIS

`/usr/lib/lp/postscript/postreverse` [*options*] [*file*]

DESCRIPTION

The `postreverse` filter reverses the page order in files that conform to Adobe's Version 1.0 or Version 2.0 file structuring conventions, and writes the results on the standard output. Only one input *file* is allowed and if no *file* is specified, the standard input is read. The following *options* are understood:

`-o` *list*	Select pages whose 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.
`-r`	Don't reverse the pages in *file*.

The `postreverse` filter can handle a limited class of files that violate page independence, provided all global definitions are bracketed by `%%BeginGlobal` and `%%EndGlobal` comments. In addition, files that mark the end of each page with `%%EndPage: label ordinal` comments will also reverse properly, provided the prologue and trailer sections can be located. If `postreverse` fails to find an `%%EndProlog` or `%%EndSetup` comment, the entire *file* is copied, unmodified, to the standard output.

Because global definitions are extracted from individual pages and put in the prologue, the output file can be minimally conforming, even if the input *file* wasn't.

EXAMPLES

To select pages 1 to 100 from *file* and reverse the pages:

```
postreverse -o1-100 file
```

To print four logical pages on each physical page and reverse all the pages:

```
postprint -n4 file | postreverse
```

To produce a minimally conforming file from output generated by `dpost` without reversing the pages:

```
dpost file | postreverse -r
```

DIAGNOSTICS

An exit status of 0 is returned if *file* was successfully processed.

NOTES

No attempt has been made to deal with redefinitions of global variables or procedures. If standard input is used, the input *file* will be read three times before being reversed.

SEE ALSO

`download`(1), `dpost`(1), `postdaisy`(1), `postdmd`(1), `postio`(1), `postmd`(1), `postprint`(1), `posttek`(1).

NAME

`posttek` – PostScript translator for tektronix 4014 files

SYNOPSIS

`/usr/lib/lp/postscript/posttek` [*options*] [*files*]

DESCRIPTION

The `posttek` filter translates tektronix 4014 graphics *files* 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.
`-f` *name*	Print text using font *name*. Any PostScript font can be used, although the best results will be obtained only with constant width fonts. The default font is Courier.
`-m` *num*	Magnify each logical page by the factor *num*. Pages are scaled uniformly about the origin which, by default, is located at the center 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 pages whose 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 landscape.
`-w` *num*	Set the line width used for graphics to *num* points, where a point is approximately 1/72 of an inch. By default, *num* is set to 0 points, which forces lines to be one pixel wide.
`-x` *num*	Translate the origin *num* inches along the positive x axis. The default coordinate system has the origin fixed at the center of the page, with positive x to the right and positive y up the page. Positive *num* moves everything right. The default offset is 0.0 inches.
`-y` *num*	Translate the origin *num* inches along the positive y axis. Positive *num* moves everything up the page. The default offset is 0.0.

DIAGNOSTICS

An exit status of 0 is returned if *files* were successfully processed.

NOTES

The default line width is too small for write-white print engines, such as the one used by the PS-2400.

FILES

```
/usr/lib/postscript/posttek.ps
/usr/lib/postscript/forms.ps
/usr/lib/postscript/ps.requests
```

SEE ALSO

`download(1)`, `dpost(1)`, `postdaisy(1)`, `postdmd(1)`, `postio(1)`, `postmd(1)`, `postprint(1)`, `postreverse(1)`.

NAME

pr – print files

SYNOPSIS

pr [[-*column*] [-w*width*] [-a]] [-e*ck*] [-i*ck*] [-drtfp] [+*page*] [-n*ck*] [-o*offset*] [-l*length*] [-s*separator*] [-h *header*] [*file* ...]

pr [[-m] [-w*width*]] [-e*ck*] [-i*ck*] [-drtfp] [+*page*] [-n*ck*] [-o*offset*] [-l*length*] [-s*separator*] [-h *header*] *file1 file2* ...

DESCRIPTION

Pr is used to format and print the contents of a file. If *file* is -, or if no files are specified, pr assumes standard input. pr prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with -h), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the -s option is used, lines are not truncated and columns are separated by the *separator* character.

Either -*column* or -m should be used to produce multi-column output. -a should only be used with -*column* and not -m.

Command line options are

+*page* Begin printing with page numbered *page* (default is 1).

-*column*
: Print *column* columns of output (default is 1). Output appears as if -e and -i are turned on for multi-column output. May not use with -m.

-a Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.

-m Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with -*column*.

-d Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.

-e*ck* Expand input tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

-i*ck* In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

-n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of -m output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

-w*width*
Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (-*column* and -m). There is no line limit for single column output.

-o*offset*
Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

-l*length*
Set the length of a page to *length* lines (default is 66). When the value of *length* is 1 to 10, -t appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When -l*length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text.

-l0 prints the file as if it were a single page, with one header and trailer but no other blank lines added.

-h *header*
Use *header* as the text line of the header to be printed instead of the file name. -h is ignored when -t is specified or -l*length* is specified and the value of *length* is 10 or less. (-h is the only pr option requiring space between the option and argument.)

-p Pause before beginning each page if the output is directed to a terminal (pr will ring the bell at the terminal and wait for a carriage return).

-f Use single form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

-r Print no diagnostic reports on files that will not open.

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of -t overrides the -h option.

-s*separator*
Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless -w is specified.

International Features

pr can process characters from supplementary code sets in addition to ASCII characters.

Options:

-e*ck* The tab character *c* must be a single byte character. *k* is the tab position specified in columns, not in characters.

`-i`*ck*	The tab character *c* must be a single byte character. *k* is the tab position specified in columns, not in characters.
`-n`*ck*	The character *c* to be appended to the line number must be a single byte character.
`-w`*width*	*width* is the width of a line in columns, not in characters.
`-h`*header*	characters from supplementary code sets can be used in the page header, *header*.
`-s`*separator*	The column separator, *separator*, must be a single byte character.

EXAMPLES

```
pr listing
```

Prints the file named "listing" on the standard output. Output is paginated, and each page has a heading consisting of the time and date.

```
pr -2 -h users personnel
```

Prints the contents of the file "personnel" in two columns on standard output. Each page has a header consisting of the time and date, the word "users," and the page number.

```
pr -t -5 listing | pr -t -e > lists.column
```

Formats the file "listing" in five columns without including a header and trailer. The second `pr` command expands tabs that are in the file to eight character columns.

FILES

`/dev/tty*`	To delay messages enabling them to print at the bottom of files rather than interspersed throughout printed output.

SEE ALSO

`cat`(1), `pg`(1).

NAME

printenv – print out the environment

SYNOPSIS

printenv [*name*]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

Examples of the environment variable names are:

HOME	pathname of user's home directory
PATH	search path for binary programs
TERM	type of terminal used

If a *name* is specified and it is not defined in the environment, printenv returns exit status 1, else it returns status 0.

EXAMPLES

```
$ printenv
EXINIT=set number showmode redraw
HOME=/udd/sdd08/intern
LOGNAME=intern
MAIL=/usr/mail/intern
PATH=:/udd/sdd08/intern/util:/bin:/usr/bin:/etc:/usr/local
TERM=d216-dg
$
```

This command prints out the values in the invoking process's environment.

SEE ALSO

csh(1), sh(1), environ(5).

NAME

`printf` – print formatted output

SYNOPSIS

`printf` *format* [*arg*...]

DESCRIPTION

The `printf` command converts, formats, and prints its *arg*s under control of the *format*. It fully supports conversion specifications for strings (`%s` descriptor); however, the results are undefined for the other conversion specifications supported by `printf`(3S).

format	a character string that contains three types of objects: 1) plain characters, which are simply copied to the output stream; 2) conversion specifications, each of which results in fetching zero or more *arg*s; and 3) C-language escape sequences, which are translated into the corresponding characters.
arg	string(s) to be printed under the control of *format*. The results are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character `%`. After the `%`, the following appear in sequence:

An optional field, consisting of a decimal digit string followed by a `$`, specifying the next *arg* to be converted. If this field is not provided, the *arg* following the last *arg* converted is used.

An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it is padded on the left (or right, if the left-adjustment flag '–' has been given) to the field width. The padding is with blanks unless the field width digit string starts with a zero, in which case the padding is with zeros.

An optional *precision* that gives the maximum number of characters to be printed from a string in `%s` conversion. The precision takes the form of a period (`.`) followed by a decimal digit string; a null digit string is treated as zero (nothing is printed). Padding specified by the precision overrides the padding specified by the field width. That is, if *precision* is specified, its value is used to control the number of characters printed.

A field width or precision or both may be indicated by an asterisk (*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen, so the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted. A negative field width argument is taken as a `-' (left-adjustment) flag followed by a positive field width. If the precision argument is negative, it is changed to zero (nothing is printed). In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

The conversion characters and their meanings are:

`%s`	The *arg* is taken to be a string and characters from the string are printed until a null character (\0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A null value for *arg* yields undefined results.

`%%` Print a `%`; no argument is converted.

EXAMPLES

The command

```
printf '%s %s %s\n' Good Morning World
```

results in the output:

```
Good Morning World
```

The following command produces the same output.

```
printf '%2$s %s %1$s\n' World Good Morning
```

Here is an example that prints the first 6 characters of `$PATH` left-adjusted in a 10-character field:

```
printf 'First 6 chars of %s are %-10.6s.\n' $PATH $PATH
```

If `$PATH` has the value `/usr/bin:/usr/local/bin`, then the above command would print the following output:

```
First 6 chars of /usr/bin:/usr/local/bin are /usr/b    .
```

SEE ALSO

awk(1), banner(1), echo(1), printf(3S).

NAME

ps – report process status

SYNOPSIS

`ps` [*options*]

DESCRIPTION

`Ps` prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information displayed is controlled by the selection of *options*.

Options using lists as arguments can specify them in two forms: a list of identifiers separated by commas, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

`-e`	Print information about all processes.
`-d`	Print information about all processes, except process group leaders.
`-a`	Print information about all processes, except process group leaders and processes not associated with a terminal.
`-f`	Generate a *full* listing. (See below for meaning of columns in a full listing).
`-j`	Print session ID and process group ID for each process listed.
`-l`	Generate a *long* listing (see below).
`-c`	Print scheduling class and priority for each process listed. If this option is specified, the processor utilization and nice value columns are not listed. It is intended that in a future release of DG/UX this option will be removed, and that the output format produced by using this option will become the default format.
`-t` *termlist*	List data only about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified as: the device's filename (e.g., `tty04`) or if the device's filename starts with `tty`, just the digit identifier (e.g., `04`).
`-p` *proclist*	List data only about processes whose process ID numbers are given in *proclist*.
`-u` *uidlist*	List data only about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID is printed unless the `-f` option is used, in which case the login name will be printed.
`-g` *grplist*	List data only about processes whose process group leaders are given in *grplist*.
`-s` *sidlist*	List data only about processes whose session leaders are given in *sidlist*.

The column headings and the meaning of the columns in a `ps` listing are given below; the letters `f` and `l` indicate the option (*full* or *long*) that causes the corresponding heading to appear; `all` means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

`F`	`(l)`	Flags (octal and additive) associated with the process: 1 Process is being traced; 2 Process is bound to a virtual processor; 4 Process is not bound to a virtual processor;

`S`	`(l)`	The state of the process:
		– Non-existent;
		S Sleeping;
		W Waiting;
		R Running;
		I Intermediate;
		Z Terminated;
		T Stopped;

`UID`	`(f,l)`	The user ID number of the process owner; the login name is printed under the `-f` option.
`PID`	`(all)`	The process ID of the process; you can kill a process if you know this datum.
`PPID`	`(f,l)`	The process ID of the parent process.
`PGID`	`(all)`	The group ID of the process. Only printed when `-j` option is used.
`SID`	`(all)`	The session ID of the process. Only printed when `-j` option is used.
`C`	`(f,l)`	Processor utilization, represented by an integer from 0 to 7. This number reflects a process' relative interactivity. A process with 7 is highly interactive. A process with 0 is not considered interactive, but uses mostly CPU resources. You cannot control this value; it is produced dynamically by the kernel. Not printed when `-c` option is used.
`CLS`	`(f,l)`	Scheduling class. Only printed when `-c` option is used.
`PRI`	`(l)`	The priority of the process; higher numbers mean lower priority.
`NI`	`(l)`	Nice value; used in priority computation. Not printed when `-c` option is used.
`ADDR`		The memory address of the process.
`SZ`	`(l)`	The size in pages of the resident memory image of the process including shared and unshared segments.
`WCHAN`	`(l)`	The event for which the process is waiting or sleeping; if blank, the process is running.
`STIME`	`(f)`	Starting time of the process, in hours:minutes:seconds.
`TTY`	`(all)`	The controlling terminal for the process.
`TIME`	`(all)`	The cumulative execution time for the process, in minutes:seconds.
`CMD`	`(all)`	The command name; the full command name and its arguments are printed under the `-f` option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked *defunct*.

Under the `-f` option, `ps` tries to determine the command name and arguments given when the process was created by examining the process stack. Failing this, the command name, as it would appear without the `-f` option, is printed in square brackets.

EXAMPLES

```
$ ps
```

Prints information about active processes associated with the current terminal. Lists the process ID, the tty number of the controlling terminal.

```
$ ps -u xyz,abc
```

Prints information about active processes belonging to the login names "xyz" and "abc". Lists the process ID, tty number, time the process took to execute, and the command.

```
$ ps -ef
```

Prints information on all active processes. Lists the user ID, process ID, process ID of the parent process, scheduling information, process starting time, tty number, execution time for the process, and the command and its options.

FILES

/etc/passwd	Supplies UID information
/etc/ps_data	Internal data structure
/dev	Searched to find terminal (tty) names

SEE ALSO

acctcom(1), kill(1), nice(1), who(1).

BUGS

Things can change while ps is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes is irrelevant.

NOTES

DG/UX ps extracts information from the currently running kernel. The -c, -n, and -s options are not implemented.

NAME

pwd – print working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

cd(1), csh(1), sh(1)

NOTES

In csh(1) the command echo $cwd is always faster than pwd. However, it can give a different answer in the rare case that the current directory or a containing directory was moved after the shell descended into it. It can also give a different answer if you have traversed a symbolic link to a directory.

NAME

`rcp` – remote file copy

SYNOPSIS

`rcp` [`-p`] *filename1 filename2*
`rcp` [`-pr`] *filename* ... *directory*

DESCRIPTION

The `rcp` command copies files between machines. Each *filename* or *directory* argument is either a remote file name of the form:

hostname:path

or a local file name (containing no `:` characters, or a `/` before any `:` characters).

If a *filename* is not a full path name, it is interpreted relative to your home directory on *hostname*. A *path* on a remote host may be quoted (using `\`, `"`, or `'`) so that the metacharacters are interpreted remotely.

`rcp` does not prompt for passwords; your current local user name must exist on *hostname* and allow remote command execution by `remsh`(1) or `rsh`(1C).

`rcp` handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form

username@hostname:filename

to use *username* rather than your current local user name as the user name on the remote host. `rcp` also supports Internet domain addressing of the remote host, so that:

username@host.domain:filename

specifies the username to be used, the hostname, and the domain in which that host resides. Filenames that are not full path names will be interpreted relative to the home directory of the user named *username*, on the remote host.

The following options are available:

`-p` Attempt to give each copy the same modification times, access times, and modes as the original file.

`-r` Copy each subtree rooted at *filename*; in this case the destination must be a directory.

EXAMPLES

```
rcp sys8:/udd/test1 test2
```

Copies the remote file `test1` from host sys8 into the file `test2` in your current directory.

```
rcp -r sys8:net net2
```

Copies the contents of the remote directory `net` into the local directory `net2`. The destination argument (`net2`) must either be a directory or not exist. If `net2` does not exist, a directory with that name will be created.

```
rcp wilsonh@sys8:test1 sys9:net/test1
```

Copies `test1`, which is located on the remote machine `sys8`, into the file `test1` on the remote machine `sys9`. The name `wilsonh` represents the user's username on `sys8`.

FILES

`$HOME/.profile`, `$HOME/.rhosts`, `/etc/hosts.equiv`.

DIAGNOSTICS

If you forget to quote metacharacters intended for the remote host you get an incomprehensible error message.

SEE ALSO

`ftp`(1C), `remsh`(1C), `rlogin`(1C), `rshd`(1M), `hosts.equiv`(4).

NOTES

`rcp` is meant to copy between different hosts; using `rcp` to copy a file onto itself, as with:

```
rcp tmp/file myhost:/tmp/file
```

results in a severely corrupted file.

`rcp` does not detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

`rcp` can become confused by output generated by commands in a `$HOME/.profile` on the remote host.

`rcp` requires that the source host have permission to execute commands on the remote host when doing third-party copies.

NAME

`REELexchange` – commands for reading and writing IBM and ANSI tapes

DESCRIPTION

`REELexchange` is a set of commands for reading and writing IBM and ANSI tapes. The standard tape formats are defined in:

MVS/370 Magnetic Tape Labels and File Structure Administration Release Number: 1.1, Order Number:GC26-4064-1

ANSI Magnetic Tape Labels and File Structure for Information Interchange ANSI X3.27-1978

`REELexchange` supports all aspects of the tape formats including multiple file, multiple volume storage and the following record/blocking schemes:

fixed length records

fixed length, blocked records

variable length records

variable length, blocked records

variable length, spanned records

variable length, blocked, spanned records

unformatted

Tape access capabilities include:

reading file(s) from tape(s);

writing file(s) to tape(s);

initial labeling of blank tapes;

record translation;

label manipulation;

This man page is divided into the following sections and subsections:

General Guidelines

This section includes configuration information, definition of terminology, and other general information regarding `REELexchange`.

Configuration Files

The `REELexchange` commands use a number of configuration files for describing system/tape drive characteristics and for specifying default values for certain commands. The default directory used by `REELexchange` to search for configuration files is `/var/reelexchange`. This default can be modified by creating an environment variable called REELXL whose value is the name of the alternate configuration directory. For purposes of illustration, the directory `/var/reelexchange` is used throughout the remainder of this man page since it should be the configuration directory for the vast majority of users. Below is a list of configuration files used by `REELexchange` along with a brief description of each file.

`/var/reelexchange/Adn/*`

There should be one file in this directory for each tape drive on the system. The name of the file should be the name by which you wish to refer to the tape drive. This is the value which will be used with the `-a` option to the `taccess` and `tlabel` commands. Initially, this directory contains one ADN file, `drive1` which contains the following information:

```
S 1 reel
T 1600 SYSV.1600 /dev/rmt/0 /dev/rmt/0n
```

The information on the first line of the file ("S 1 reel") is used internally by the `REELexchange` commands. All of your ADN files should contain this information on the first line as well. The second line is a tape identifier entry (indicated by "T"). This entry identifies tape density in bpi (1600), the tape drive type (SYSV.1600), the device-special file for the tape drive (/dev/rmt/0), and the corresponding rewind device file (/dev/rmt/0n). The ADN file, `drive1` is intended to be used as a prototype for creating your own ADN files. DG/UX supports 800, 1600, 3200 and 6250 bpi tape drives. The corresponding tape drive types for these tape densities are SYSV.800, SYSV.1600, SYSV.3200 and SYSV.6250, respectively.

`/var/reelexchange/default.adn`

This file contains the name of one of the ADN files in the `/var/reelexchange/Adn` directory. The tape drive name specified in this file will be used as the default ADN file by the `taccess` and `tlabel` commands whenever no ADN file is specified via the `-a` option.

`/var/reelexchange/tapecap`

This file contains entries describing each type of tape drive supported by `REELexchange`.

/var/reelexchange/tapecap
/var/reelexchange/tapecap

These files, if present, contain default parameters for IBM and ANSI labels, respectively (see below).

File Labels

IBM and ANSI tape standards include labels for each file on tape. These labels are automatically processed during file reads. They can also be defined for use during file writes. The `tkey` command sets the labels and the `tdisplay` command reports all current settings.

`REELexchange` has built-in defaults for file labels. These defaults can be modified through the creation of files in the `/var/reelexchange` directory. Creating a file named `d_ibm` with keyword/value pairs (as described for the `tkey` command) creates default settings for IBM file labels. A file named `d_ansi` does the same for ANSI file labels.

Common default modifications are for expiration date and system name. Here are example lines from a `d_ibm` file which override those particular defaults:

```
exp=12/31/1991
sys=reel1
```

Volume Serial Numbers

`REELexchange` requires that each tape be identified by its Volume Serial Number (VSN). The VSN is a six-character name encoded on the tape. When a tape session is started, the user must provide the tape's VSN. `REELexchange` then reads the tape to verify that the mounted tape does indeed possess the named VSN.

If a tape's VSN is unknown, the `tlabel` command can determine it. Mount the tape and issue the command:

```
$ tlabel -r
```

Tapesets

The term tapeset is used throughout this man page. A tapeset is one or more individual tapes that are considered one logical tape. That is, tapes can be grouped together and considered to be one continuous tape. The tapes in the tapeset are ordered from one to *n* , where *n* is the number of tapes in the group.

Tapes are also referred to as volumes. Therefore, a multiple volume tapeset is a tapeset consisting of two or more tapes.

When dealing with multiple volume tapesets, order must be maintained for the tape data to be correctly accessed. The `taccess` command, as described later, maintains tape ordering.

Tape Sessions

A tape session consists of mounting a tape, reading and/or writing the tape, and unmounting the tape. REELexchange requires that the taccess command be run when starting a tape session and the trelease command be run when finishing a tape session.

The use of these two commands is described later in this man page. The following subsections describe tape session scenarios where taccess and tread are used with each and every session.

Record and Blocking Schemes

REELexchange supports all record and blocking schemes: fixed and variable length records, fixed block, spanned block, etc. Read operations automatically detect and process the formatting of each file. For writing files, these formats can be defined for each file.

See tkey and tdisplay man pages for more detailed information.

Record Translation

One of the difficulties of reading/writing IBM and ANSI tapes on a UNIX system is translating between IBM/ANSI records and UNIX data. The difficulty arises from the fact that the UNIX operating system does not support records. On UNIX, the notion of a record only has meaning to applications. For example, the program vi considers a record to be all the characters it finds between the ASCII character '\n' (newline); database applications define records to be arbitrary fixed or variable length chunks of data.

When REELexchange is reading or writing a file, it cannot guess the most appropriate way to do record translation. The tkey command provides special keys to control how record translation is done.

The keyword pchar is used to define the ASCII character REELexchange should use for padding and unpadding records. pchar may be assigned any octal constant (3 octal digits) or the value "none" which disables pad character processing. Pad character processing only affects fixed length record formats. REELexchange adds pad characters to records shorter than the current record length when writing tape files and removes trailing pad characters from records while reading tape files.

WARNING: Interaction of pchar and tchar may result in the loss of some characters in a text file. For example, if pchar =' ' (space) and tchar ='\n' (newline), and if a text file contains consecutive spaces before the newline character, then those spaces will be lost after the file is written to a tape and retrieved back to disk later. This is because the consecutive space characters are regarded as padding characters and are stripped off in the unpadding process when retrieving from the tape.

The keyword tchar is used to define the ASCII character REELexchange should use to delimit records. When writing a tape file, twrite uses tchar to determine record boundaries in the input stream. tchar should be specified when creating tapes that use variable length record formats.

The keyword conv controls whether data is converted from/to EBCDIC before it is read/written to tape. `conv=ebcidic` implies that the tape contains EBCDIC data. `conv=none` implies no conversion (tape contains ASCII data).

The `tdisplay` command may be used to display the current setting of the translation keywords. An example of the output generated by `tdisplay -t` is shown below:

```
    Translation processing:

                  Key     Value
                  ---     -----
                  pchar   40 (octal)
                  tchar   12 (octal)
                  conv    none
```

See the subsection entitled "Setting Record Formats" for further information.

Reading Files from a Tapeset

Reading files from tape is accomplished via the `tread` command. Files can be read and placed directly in UNIX files or they can be directed to standard output (stdout). The `tread` command automatically adjusts for the formatting scheme of each file and translates incoming records according to the settings made via the `tkey` command. A look at an example shows how simple it is to read in files from a labeled tape.

Sequential File Reads

EXAMPLE A

In this example, a single IBM labeled, 1600 bpi, 2400 feet long tape with VSN 000001 contains three data files which are to be read into UNIX files. The computer system has a single nine-track taped drive identified to `REELexchange` as the default drive. To begin the session, issue the command:

```
$ taccess -v000001 -d1600 -f2300
```

The `taccess` command identifies the tape to `REELexchange`. It must be the first command issued before any other `REELexchange` commands can be used on the tape.

To read the first file, issue this command:

```
$ tread >file1
```

This command reads the first file on the tape and places it in the UNIX file, file1. The tape is automatically positioned to the beginning of the second file, so to read the second file issue the command:

```
$ tread >file2
```

The third file can then be read with the same command modified to put the data in file3. Terminate the session with the command:

```
$ trelease
```

Random File Reads

The `tposn` command allows files to be read in any order from the tape. It positions the tape to the start point of any file on the tape. In the preceding Example A, the second file on the tape could have been read first by using the command:

```
$ tposn -r -n2
```

Tape files are numbered according to their order on the tape beginning with the first file at number 1. The second file is number 2, etc.

Automatically Reading Every File

EXAMPLE B

In this example, a tape contains an unknown number of files all of which need to be read into UNIX files. Below is a UNIX shell (bourne shell) script which automatically reads in all files and puts them into files numbered from 1 to *n* where *n* is the number of files on the tape. To use the script, edit a UNIX file to contain only this text:

```
FILENO=1
while ( tread > $FILENO )
do
     FILENO=‘expr $FILENO + 1‘
done
rm $FILENO
```

Understanding File Labels

Each tape has file labels which contain descriptive information about each file and its contents. `REELexchange` makes file label information available via the `tdisplay` command. To display the label information of the file at the current tape position, use the command:

```
$ tdisplay -r
```

Here is a sample of a report generated by this command:

```
Read labels

        Key     Value         Field Name
        ---     -----         ----------
VOL:    ...     000000        Volume Serial Number
        own     test_own      Owner Name and Address Code
HDR1:   fid     test_fid      Data Set Identifier
        ...     000000        Data Set Serial Number
        ...     1             Volume Seqence Number
        ...     1             Data Set Sequence Number
        gen     100           Generation Number
        genv    50            Version Number
        cre     1/1/91        Creation Date
        exp     1/1/92        Expiration Date
        acc     1             Data Set Security
        ...     0               Block Count
        sys     test_sys        System Code
HDR2:   fmt     V               Record Format (vbs)
        blen    40              Block Length
        rlen    100             Record Length
        ...     3               Tape Density
        ...     0               Data Set Position
        job     test_job        Job Step Identification
        cc      A               Control Character
        bat     R               Block Attribute
        devid   test_did        Reserved (device id)
        cpid    C               Checkpoint Data Set ID
```

When writing a file to tape, these labels can be configured as needed. See the subsection entitled "Setting File Labels."

Multiple Volume Tapesets

REELexchange handles multi-volume tapesets simply. The -v option of the taccess command takes an ordered list of VSNs (volume serial numbers) which represents the tapes in the order they belong to the tapeset. The first tape in the group should be mounted before executing taccess. As other tapes in the group need to be mounted, REELexchange prompts for the appropriate unmount and mount. The prompts appear as:

```
Please mount 000002 on tape0

Hit CR when complete
```

Creating Labeled Tapes

To label a tape, mount it and issue the tlabel command. For example, to label an IBM tape you might issue the command:

```
$ tlabel -d1600 -lIBM -v000001
```

Where -d1600 indicates 1600 bpi density, -lIBM indicates IBM format, and -v0000001 indicates the volume serial number (VSN) is 000001. See tlabel(1) for more information.

Writing Files to a Tapeset

The command twrite allows a UNIX file to be written on a tape file. twrite writes file labels and then reformats the UNIX file into tape records. Both the labels and records are constructed according to configurable parameters.

As with tread , the twrite command can only be used following an initial taccess command. Moreover, it is important to position the tape correctly before writing a file. The tposn command is used to position the tape before files are written.

EXAMPLE C

Consider the example where a labeled (IBM) but empty tape is the destination for three UNIX files. The first step is to start the tape session by mounting the tape and issuing the command:

```
$ taccess -v000001 -d1600 -f2350
```

The UNIX files are named data1, data2, and data3. To write data1 to tape:

```
$ tposn -w -n1
$ twrite < data1
```

The first command positions the tape for writing at the beginning of the first tape (file number 1). The second command actually performs the file write. To write the second and third files:

```
$ twrite < data2
$ twrite < data3
```

To close the session, use the trelease command.

As mentioned before, current label information and record formats are used to create appropriate tape labels as each file is written. The next two subsections describe what labels and formats are and how to modify them.

Setting File Labels

Each tape file has labels. When a file is written to tape, REELexchange uses the current values of its internal label variables to create the appropriate tape labels. The tkey command is used to set the label variables. Their current values are summarized by the tdisplay command.

IBM tape file labels are reported by tdisplay in this format:

```
Write labels

        Key     Value           Field Name
        ---     -----           ----------
VOL:    ...     000000          Volume Serial Number
        own     test_own        Owner Name and Address Code
HDR1:   fid     test_fid        Data Set Identifier
        ...     000000          Data Set Serial Number
        ...     1               Volume Seqence Number
        ...     1               Data Set Sequence Number
        gen     100             Generation Number
        genv    50              Version Number
        cre     1/1/91          Creation Date
        exp     1/1/92          Expiration Date
        acc     1               Data Set Security
        ...     0               Block Count
        sys     test_sys        System Code
HDR2:   fmt     V               Record Format (vbs)
        blen    40              Block Length
        rlen    100             Record Length
        ...     3               Tape Density
        ...     0               Data Set Position
        job     test_job        Job Step Identification
        cc      A               Control Character
        bat     R               Block Attribute
        devid   test_did        Reserved (device id)
        cpid    C               Checkpoint Data Set ID
```

ANSI tape labels are reported by `tdisplay` in this format:

```
Write labels

          Key      Value            Field Name
          ---      -----            ----------
VOL:      ...      000000           Volume Serial Number
          acc                       Volume accessibility
          own                       Owner Identifier
HDR1:     fid                       File Identifier
          ...      000000           File Set Identifier
          ...      1                File Section Number
          ...      1                File Sequence Number
          gen      0                Generation Number
          genv     0                Version Number
          cre      1/1/1991         Creation Date
          exp      1/1/2001         Expiration Date
          facc                      File accessibility
          ...      0                Block Count
          sys                       System Code
HDR2:     fmt      F                Record Format
          blen     800              Block Length
          rlen     80               Record Length
          usr                       User Data
          ...      0                Buffer-Offset
```

Label fields are modified with the `tkey` command.
`tkey` recognizes the "key" names listed by `tdisplay`. (Note the key value "..." indicates a field maintained by `REELexchange` only--the user cannot set the field's value). For example, to set the Generation Number to 5, use this command:

```
$ tkey gen=5
```

To change both the Generation Number and the Expiration Date:

```
$ tkey gen=5 exp=12/31/1999
```

File label settings can be grouped into a file, one parameter per line and read by `tkey` with the `-f` option:

```
$ tkey -f/usr/data/label_settings
```

Where the file /usr/data/label_settings can contain entries of this type:

```
gen=5
exp=12/31/1999
```

Default File Labels

`REELexchange` uses default values for the file label parameters. Sites may establish different defaults via `d_ibm` (for IBM tapes) and `d_ansi` (for ANSI tapes) files located in the `/var/reelexchange` directory. The built-in default values are:

```
IBM                 ANSI
---                 ----
own=""              own=""
fid=""              acc=""
gen=0               fid=""
genv=0              gen=0
cre=current_date    genv=0
exp=1/1/2001        cre=current_date
acc=0               exp=1/12001
sys=""              facc=''
fmt=F               fmt=F
blen=800            blen=800
rlen=80             rlen=80
job=""              usr1=""
cc=''
bat=""
devid=""
cpid=''
```

Setting Record Formats

`REELexchange` supports all record formats for both IBM and ANSI standards.

The IBM record format may be set one of two ways. The first is by setting the `fmt` and `bat` label fields to legal values describing the desired record format (values are defined by the IBM standard). The second, and preferred way, is by using a special `tkey` keyword `rfmt`. Legal values for `rfmt` and the record formats they imply are:

`u`	- unformatted
`f`	- fixed length records
`fb`	- fixed length blocked records
`v`	- variable length records
`vb`	- variable length blocked records
`vs`	- variable length spanned records
`vbs`	- variable length block spanned records

`rfmt` changes the value of the `fmt` and `bat` keywords to the appropriate values for the given record format. The current value of `rfmt` is displayed by `tdisplay` at the end of the line for the `fmt` key.

The ANSI record format is set using the `tkey` keyword `fmt`. Legal values for `fmt` and the record formats they imply are:

`F`	- fixed length records
`D`	- variable length records
`S`	- spanned records

See the subsection titled "Record Translation" for more information.

Multiple Volume Tapesets

REELexchange can write to multi-volume tapesets. When beginning a session where a tapeset will consume more than one tape volume, list several labeled tapes on the opening taccess command. List AT LEAST as many tapes as will be used during the session. For example:

```
$ taccess -v0000001,0000002,0000003 -d1600 -f2350
```

WARNING: If too few tapes are listed and an attempt is made to write or read past the end of the last tape, REELexchange will abort the access session and the entire session may have to be repeated.

When twrite comes to the end of the first tape, it prompts for the second tape in the list to be mounted. It does the same at the end of the second tape, prompting for the third tape. This process continues until the session is closed with the trelease command, or until it runs out of tapes. Should this happen, twrite signals the condition and fails to write the last file:

```
twrite: Error, nextvol ran off end of volume list
```

The REELexchange Commands

This section gives a brief description of each of the REELexchange commands. For a formal description of command syntax, refer to the man page for the command in question.

taccess

The taccess command must always be used to begin a tape session. Through it, the user identifies the tapeset and its characteristics: density, length, tape format. Also, the user specifies the Volume Serial Number(s) (VSNs) for the tape volume(s). If a multi-volume tapeset is being processed, then the full list of VSNs must be named.

The taccess command also implicitly performs the equivalent of tposn -r -n1 (i.e. It positions the tape to read the first file).

tdisplay

tdisplay reports the current settings of file labels and record translation parameters. The tkey command is used to modify label and translation settings.

The -r option reports the file labels found after a tposn -r or a tread command has been issued.

The -w option reports the file labels which would be written by a subsequent twrite command. When a tposn command is executed, these labels always revert to the default values. The tkey command allows the labels to be modified before writing.

The -d option displays the default file labels used during file writes. They can be changed with the tkey -d command.

Current record translation settings are reported via the -t option:

Translation processing:

Key	**Value**
pchar	40(octal)
tchar	12(octal)
conv	none

tkey

`tkey` modifies file labels and record translation parameters. See the subsections entitled "Record Translation" and "Setting File Labels" for more information.

tlabel

`tlabel` writes the appropriate volume labels on a tape. A tape must be labeled before it can be written or read by `twrite` and `tread`. `tlabel -r` reports the VSN of the labeled tape.

tposn

`tposn` moves the tape to be correctly positioned for either a subsequent read (`tread`) or write (`twrite`). Read and write start at different positions for a file, so the position must be made specifically for the planned operation. With the `-r` option, `tposn` positions for a read. The `-w` option positions for a subsequent write.

`tposn` resets file labels. If positioning for a write, `tposn` resets the current file labels to default values. These default values can be modified using the `tkey` command.

tread

The `tread` command reads a file from tape and sends the output to standard output (stdout). It automatically reads file labels making them available for review via the `tdisplay` command. `tread` automatically detects the tape file format and translates records according to the current translation settings as configured by the `tkey` command.

trelease

`trelease` ends a tape session.

tsniff

`tsniff` produces a table of contents report on the tapeset. It reads each tape in order and produces a report. For example:

```
Tape Contents - vsn: 000001

fseq   fid        fmt   blen   rlen   blocks
----   ---        ---   ----   ----   ------
1                 F     100    100    194
2                 F     1000   100    20
3                 D     200    100    28
4                 S     30     100    209
```

twrite

The `twrite` command writes a file to the tape receiving input from standard input (stdin). It also writes file labels as configured by the `tkey` command. `twrite` automatically translates the UNIX data into tape records according to the current translation settings, also configured by the `tkey` command.

FILES

`/var/reelexchange/default.adn`	File containing name of default tape drive.
`/var/reelexchange/tapecap`	File containing descriptive entries for all tape drives supported by `REELexchange`.
`/var/reelexchange/Adn/*`	Each file in this directory describes a tape drive on the system. The name of one of these files can be specified with the `-a` option of the `taccess` command to identify a particular tape drive to `REELexchange`.
`/var/reelexchange/d_ibm`	IBM label defaults.
`/var/reelexchange/d_ansi`	ANSI label defaults.

DIAGNOSTICS

`Error,adn` *name* does not exist

`Problem:` The named tape drive is not identified under the current configuration.
`Solution:` Check the drive name. If a new drive needs to be configured, reinstall the software.

`Error - ans _ r unknown label type` *name*

`Problem:` The tape uses a label type not understood by REELexchange.
`Solution:` None. REELexchange can only read IBM or ANSI standard tapes.

`Error, blen not in range 1 to 32760`

`Problem:` The block length must be in the stated range.
`Solution:` Reset the block length keyword to a legal value.

`Error: corrupted adn file` *filename*

`Problem:` The configuration files are inconsistent.
`Solution:` Reinstall the REELexchange software.

`Error,fmt invalid`

`Problem:` An illegal format has been specified.
`Solution:` Reselect the format, using one of the legal values described in chapter 3.

`Error, hard error on rewind ioctl`

`Problem:` The tape drive failed while rewinding.
`Solution:` Release the tape and unmount it.

`Error, illegal label size` *number*
 `Problem:` The label record size (*number*) is less than eighty bytes.
 `Solution:` The record size must be eighty bytes or greater.

`Error, i/o error reading label`
 `Problem:` The tape drive failed during a read operation.
 `Solution:` Release the tape and unmount it. Retry the tape session. If the problem persists, tape drive maintenance may be in order.

`Error: No default device name...`
 `Problem:` A default tape drive name was not specified during installation.
 `Solution:` Reinstall the REELexchange software and designate the default tape drive name.

`Error, read error skipping file in t_posn`
 `Problem:` The tape drive failed during a positioning move.
 `Solution:` Release the tape and unmount it. Retry the tape session. If the problem persists, tape drive maintenance may be in order.

`Error, record does not equal block size`
 `Problem:` The block size is not an integral multiple of the record size. This occurs under the fixed record format.
 `Solution:` Redefine the block and/or record size appropriately.

`Error, record exceeds block size` *num1* > *num2*
 `Problem:` The given record size exceeds the given block size for a fixed or non-spanned record format.
 `Solution:` Redefine the block and/or record size appropriately.

`Error,r_defadn could not access default adn file`*name*
 `Problem:` The configuration files are inconsistent.
 `Solution:` Reinstall the REELexchange software.

`Error: tape on` *adn* `has VSN` *VSN1* `instead of` *VSN2*
 `Problem:` The mounted tape has a VSN(VSN1) different from the VSN(VSN2) specified by the earlier taccess command.
 `Solution:` Release the tape and retry the session. Use the tlabel command to check the tape's VSN if necessary.

`Error: Unknown label Identifier:`*name*
 `Problem:` The tape labels are non-standard.
 `Solution:` None. REELexchange supports IBM and ANSI standard tapes.

`Error, unknown record format` *name*
 `Problem:` The specified format is non-standard.
 `Solution:` None. REELexchange supports IBM and ANSI standard tapes.

`Error, unknown segment descriptor`
 `Problem:` The tape format is non-standard.
 `Solution:` None. REELexchange supports IBM and ANSI standard tapes.

`Error, Variable length records require termination processing`
 `Problem:` The file contains variable length records and record translation has not been specified.
 `Solution:` Use the tkey command to enable record translation processing.

`REEL-Permission Denied`
 `Problem:` The REELexchange software is not properly installed.
 `Solution:` Reinstall the REELexchange software.

SEE ALSO

`taccess`(1), `tdisplay`(1), `tkey`(1), `tlabel`(1), `tposn`(1), `tread`(1), `trelease`(1), `tsniff`(1), `twrite`(1).

NAME

remsh, rsh – create a remote shell to execute a command

SYNOPSIS

remsh *host* [-l *username*] [-n] *command*
rsh *host* [-l *username*] [-n] *command*
host [-l *username*] [-n] *command*

DESCRIPTION

Use the remsh command to connect to the specified *host* and execute the specified *command*. The remsh command copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are passed to the remote command; remsh normally terminates when the remote command does. Caution: the runtime environment for the command is not the same as for a login shell so some environment variables such as TZ (timezone) and those specified in your .login or .profile will not be set. For more details about the runtime envionment see the rshd man page.

NOTE: Your system administrator may choose to call this command rsh in addition to remsh.

The remote username used is the same as your local username, unless you specify a different remote name with the -l option. This remote name must be equivalent to the originating account. You will not need to give a password.

You can have a private equivalence list in a file .rhosts in your log-in directory. Each line in this file should contain a *remote-hostname* and a *username* separated by a space, indicating the users (and their respective systems) to whom you want to give access to your account.

The /etc/hosts.equiv file allows users who have accounts on two systems to use Remote Commands between systems without creating a .rhosts file. Each line in the /etc/hosts.equiv file should contain a *hostname*. This file gives users from the systems listed and who have accounts on the local system access to their accounts. These users must have the same username on both systems.

CAUTION: If two different users from foreign systems listed in the /etc/hosts.equiv file have the same user ID, then the two users will have access to each other's accounts.

If you omit *command*, you will be logged in on the remote host using rlogin.

Unquoted shell metacharacters are interpreted on the local machine, whereas quoted metacharacters are interpreted on the remote machine. Thus, the command:

remsh *otherhost* cat *remotefile* >> *localfile*

appends the remote file *remotefile* to the local file *localfile,* whereas:

remsh *otherhost* cat *remotefile* ">>" *otherremotefile*

appends *remotefile* to *otherremotefile*.

Hostnames are specified in the file /etc/hosts. Each host has one standard name (the first name given in the file) and one or more optional nicknames.

remsh can be set up to use a favorite remote system by typing only the name of the host. To set up this feature, create a symbolic link in a directory on your search path, named the desired hostname and directed at `/usr/bin/remsh`. If you are running `csh`(1), you should then run `rehash` to pick up this new link. For example, assume that you have `/usr/writers` as a directory on your path and `poets` is the name of a remote system you want to log in to. You would make the link as follows: `ln -s /usr/bin/remsh /usr/writers/poets`. This would allow you to log into the remote system `poets` by typing `poets` from the shell.

FILES

`/etc/hosts`

SEE ALSO

`rlogin`(1C), `rshd`(1M), `hosts.equiv`(4).
`on`(1C) in ONC/NFS.

BUGS

If you are using `csh`(1) and put an `remsh` in the background without redirecting its input away from the terminal, the command will block even if no reads are posted by the remote command. If you do not want input, redirect the input of `remsh` to `/dev/null` using the `-n` option.

You cannot run an interactive command (such as `vi`(1)) with `remsh`; use `rlogin`(1C).

Stop signals stop the local `remsh` process only.

NAME

`renice` – alter priority of running processes

SYNOPSIS

`/etc/renice` [*priority*] [[`-p`] *pid* ...] [[`-g`] *pgrp* ...] [[`-u`] *user* ...]

`/etc/renice` *priority pid* [*pid*]

where:

priority	An integer in the range −20 to +19. If you supply a number less than −20, `renice` uses −20 (which puts your process at the highest possible priority). If you supply a number greater than 19, `renice` uses 19 (lowest possible priority). In the first syntax given above, you can omit *priority* and have `renice` assume 0 as the default. In the second form of syntax, you must supply a *priority* value.
pid	A process identification number.
pgrp	A process group ID.
user	A user name.

DESCRIPTION

`Renice` resets the scheduling priority of one or more running processes. Users other than superuser can reset the priority of processes they own, but only to a higher number (lower priority). The superuser can raise as well as lower priorities.

Options

`-p`	Processes whose process ids are specified; this is the default.
`-g`	Processes in the specified process group.
`-u`	Processes owned by the specified user.

Useful priorities in DG/UX are:

19	The affected processes will run only when nothing else in the system wants to.
0	This is the "base" scheduling priority.
<0	Lower numbers make processes go faster (the lower limit is −20).

EXAMPLE

```
/etc/renice -10 987 -u daemon root -p 32
```

Resets to −10 the priority of processes with IDs 987 and 32, and all processes owned by users `daemon` and `root`.

```
/etc/renice -p 12488 12489 12490
```

Resets to 0 the priority of processes 12488, 12489, and 12490.

```
/etc/renice 19 12488 12489 12490
```

Resets to 19 the priority of processes 12488, 12489, and 12490.

FILES

`/dgux`	
`/etc/passwd`	To map user names to user ID numbers

SEE ALSO

`nice`(1), `getpriority`(2), `nice`(2), `setpriority`(2).

NOTES

If you make the priority a very low number (such as −20), you cannot interrupt the process. To regain control, reset the priority to a number greater than 0.

Non-superusers cannot increase scheduling priorities (that is, lower the priority numbers) of their own processes, even if they are the ones who originally decreased the priorities.

NAME

reset – reset the teletype bits to a sensible state

SYNOPSIS

reset

DESCRIPTION

Reset sets the terminal to cooked mode, turns off cbreak and raw modes, turns on nl, and restores special characters that are undefined to their default values.

This is most useful after a program dies leaving a terminal in a funny state. You must type <NL>reset<NL> to get this command to work because <CR> often doesn't work. Don't be alarmed if you don't see this command echo to the screen.

SEE ALSO

stty(1)

BUGS

Doesn't set tabs properly; it can't intuit personal choices for interrupt and line kill characters, so it may leave these set to the local system standards.

NAME

rlogin – remote login

SYNOPSIS

rlogin *rhost* [-e*c*] [-l *username*] [-8]

DESCRIPTION

Use the rlogin command to log in to another system over the network. The remote system will prompt you for a login and password, as in login(1C), unless auto-login is set up. If the remote system is running Trusted DG/UX, the user must be specifically authorized for rlogin service.

All echoing takes place on the remote host, so rlogin is transparent. Flow control via ^S and ^Q occurs on the local machine. To have these flow control characters processed on the remote machine, invoke rlogin with the -8 switch. The flushing of input and output on interrupts are handled properly. A line of the form "~." disconnects from the remote host, where "~" is the escape character. A different escape character may be specified by the -e option. Do not type a space between the -e option and the new escape character.

You can use remsh to streamline the process of logging into remote systems. Note that remsh is normally disabled on the Trusted DG/UX System. If the remote system is running Trusted DG/UX, this procedure will not work. Although rlogin is used to log in to the remote system, you will need to type only the hostname of the remote system, omitting rlogin from the command line. To set up this feature, create a symbolic link in a directory on your search path, named the desired hostname, and directed at /usr/bin/remsh. If you are running csh(1), you should then run rehash to pick up this new link. For example, assume that you have /usr/writers as a directory on your path and poets is the name of a remote system you want to log in to. You would make the link as follows: ln -s /usr/bin/remsh /usr/writers/poets. This would allow you to log into the remote system poets by typing poets from the shell.

If you are using csh(1), you can suspend a remote login session and return to the shell by using the escape sequence (~) followed by the suspend command. The suspend command is ^z by default.

Specify a different *username* with the -l option. (There must be a space between the -l and the *username*.) Use this option when your username on the foreign system is different from your username on the current system.

You can enable auto-login by having a private equivalence list in a file .rhosts in your log-in directory. Each line in this file should contain a remote hostname and a *username* separated by a space, indicating the users (and their respective systems) to whom you want to give access to your account.

The /etc/hosts.equiv file allows users with accounts on two systems to use Remote Commands between systems without creating a .rhosts file. Each line in /etc/hosts.equiv should contain a *hostname*. This file gives users from the systems listed who have accounts on the local system access to their accounts. These users must have the same username on both systems. On the Trusted DG/UX System, auto-login is disabled; you must always supply a password.

The rlogin command provides an eight-bit data path to the network if the local system stty settings provide one. Make use of the eight-bit data path with the -8 option.

WARNING: If two different users from foreign systems listed in the /etc/hosts.equiv file have the same username, then the two users will have access to each other's accounts.

The `rlogin` command and `rlogind` server allow for the dynamic exchange of window size information. This is particularly useful in an environment in which you use windowing software such as X windows. Suppose that within a window, you use `rlogin` to log in to a host. If you change that window's dimensions through the mouse, the new dimensions are propagated to the corresponding remote server, `rlogind`. The remote kernel data structures are then changed to reflect these size changes. This information exchange is transparent to a user. For this enhancement to be fully realized, both the local and remote machines must be running the appropriate versions of `rlogin` and `rlogind`.

EXAMPLES

```
$ rlogin syst3 ↵

login: jones ↵
Password:
```

Connects to the remote system `syst3`. The remote system prompts for a username and a password.

```
$ rlogin syst4 –ep ↵

login: smith ↵
Password:
```

Connects to the remote system `syst4`. Changes the escape character to `p`. The remote system prompts for a username and password.

SEE ALSO

`remsh`(1C).

Trusted Facility Manual for the C2 Trusted DG/UX™ System (093-701110)

BUGS

More terminal characteristics should be added.

NAME

rm, rmdir – remove, delete files or directories

SYNOPSIS

`rm [ -f ] [ -i ]` *file* ...

`rm -r [ -f ] [ -i ]` *dirname*

`rmdir [ -p ] [ -S ]` *dirname*

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry is the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

When you try to remove a file that does not have write permission and the input is a terminal, the file's permissions, followed by a question mark, are printed and a line is read from the standard input. It the line that you type in begins with y, the file is deleted; otherwise, the file remains.

If you try to remove a file that is a directory, an error message is printed. You will not get an error message if you use the optional argument -r.

Options are:

-f Turns prompting off for removing files that the user has no write permission on.

-r If the file to be removed is a directory, recursively deletes the entire contents of the specified directory and the directory itself.

-i Interactive mode: rm and, under -r, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

Options are:

-p Try to delete the named directory and all its parent directories, which become empty. Prints a message to standard output indicating whether or not the whole path is removed or part of the path remains.

-s Don't print messages to standard output when -p is in effect.

EXAMPLES

```
$ rm code
```

Removes the file named "code" from the current working directory.

```
$ rm -i *.o
```

You are asked whether each file that ends in ".o" should be removed. If you had a file "jqr.o", the rm command would print

```
jqr.o: ?
```

rm waits for you to respond "y" for yes, or "n" for no.

```
$ rm -rf $HOME/ITEMS
```

All files and directories in the directory "$HOME/ITEMS" are removed and then the directory "ITEMS" itself is removed. If you do not have write permission for a file, the "f" option removes it without notifying you of that fact.

DIAGNOSTICS

Generally self-explanatory. You can't remove the parent directory pointer file (`..`).

SEE ALSO

`file`(1), `ls`(1).
`unlink`(2) in the *Programmer's Reference for the DG/UX System*

NAME

`rpcgen` – an RPC protocol compiler

SYNOPSIS

`rpcgen` *infile*
`rpcgen -c | -h | -l |-m [ -o` *outfile* `] [` *infile* `]`
`rpcgen -s` *transport* `[ -o` *outfile* `] [` *infile* `]`

DESCRIPTION

`rpcgen` is a tool that generates C code to implement an RPC protocol. The input to `rpcgen` is a language similar to C known as RPC Language (Remote Procedure Call Language). Information about the syntax of RPC Language is available in *Managing ONC™/NFS® and Its Facilities on the DG/UX™ System.*

`rpcgen` is normally used as in the first synopsis where it takes an input file and generates four output files. If the *infile* is named `proto.x`, then `rpcgen` will generate a header file in `proto.h`, XDR routines in `proto_xdr.c`, server-side stubs in `proto_svc.c`, and client-side stubs in `proto_clnt.c`.

The other synopses shown above are used when one does not want to generate all the output files, but only a particular one. Their usage is described in the OPTIONS section below.

The C-preprocessor, `cpp(1)`, is run on all input files before they are actually interpreted by `rpcgen`, so all the `cpp` directives are legal within an `rpcgen` input file. For each type of output file, `rpcgen` defines a special `cpp` symbol for use by the `rpcgen` programmer:

RPC_HDR	defined when compiling into header files
RPC_XDR	defined when compiling into XDR routines
RPC_SVC	defined when compiling into server-side stubs
RPC_CLNT	defined when compiling into client-side stubs

In addition, `rpcgen` does a little preprocessing of its own. Any line beginning with '`%`' is passed directly into the output file, uninterpreted by `rpcgen`.

You can customize some of your XDR routines by leaving those data types undefined. For every data type that is undefined, `rpcgen` will assume that there exists a routine with the name `xdr_` prepended to the name of the undefined type.

OPTIONS

`-c` Compile into XDR routines.

`-h` Compile into `C` data-definitions (a header file)

`-l` Compile into client-side stubs.

`-m` Compile into server-side stubs, but do not generate a main routine. This option is useful for doing callback-routines and for people who need to write their own main routine to do initialization.

`-o` *outfile*
Specify the name of the output file. If none is specified, standard output is used (`-c`, `-h`, `-l` and `-s` modes only).

`-s` *transport*
Compile into server-side stubs, using the given transport. The supported transports are `udp` and `tcp`. This option may be invoked more than once so as to compile a server that serves multiple transports.

SEE ALSO

cpp(1)
Managing ONC/NFS and Its Facilities on the DG/UX System.

BUGS

Nesting is not supported. As a work-around, structures can be declared at top-level, and their name used inside other structures in order to achieve the same effect.

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures and types.

NAME

`rup` – show host status of local machines (RPC version)

SYNOPSIS

`rup [ -h ] [ -l ] [ -t ] [` *host...* `]`

DESCRIPTION

`rup` gives a status similar to `ruptime` for remote machines. It broadcasts on the local network, and displays the responses it receives.

Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below.

When *host* arguments are given, rather than broadcasting `rup` will only query the list of specified hosts.

A remote host will only respond if it is running the `rstatd` server, which is normally started up from `inetd`(1M).

OPTIONS

`-h`	Sort the display alphabetically by host name.
`-l`	Sort the display by load average.
`-t`	Sort the display by up time.

SEE ALSO

`ruptime`(1C), `inetd`(1M), `rstatd`(1M).

BUGS

Broadcasting does not work through gateways.

NAME

ruptime – show host status of local machines

SYNOPSIS

`ruptime [ -a ] [ -r ] [ -t | -u | -l ]`

DESCRIPTION

Use the `ruptime`(1C) command to display a status line for each machine that is on the local network and running `rwhod`(1C). These lines are formed from packets broadcast once every three minutes by each host running `rwhod` on the network.

Machines for which no status report has been received for eleven minutes are shown as being down.

Users who are idle an hour or more are not counted unless the `-a` flag is given.

Normally, the listing is sorted alphabetically by hostname. The `-l`, `-r`, `-t`, and `-u` flags specify sorting by load average, reverse sort, uptime, and number of users, respectively.

EXAMPLES

In the following example, the last three columns represent load averages for the intervals 1, 5, and 15 minutes. The load average is the average number of jobs in the run queue. It is a relative indication of how busy the systems are.

```
$ ruptime ↵
sys14    up   10:46,  4 users, load  0.04,  0.03,  0.04
sys16  down    1:14
sys10    up 1+02:11,  1 user,  load  2.40,  2.52,  2.43
$
```

Shows the host status of the machines on the local area network.

FILES

`/var/spool/rwho/whod.*`

SEE ALSO

`rwho`(1C), `rwhod`(1M).

NAME

`rusers` – who's logged in on local machines (RPC version)

SYNOPSIS

`rusers` [`-ahilu`] [*host*...]

DESCRIPTION

The `rusers` command produces output similar to `who`(1), but for remote machines. It broadcasts on the local network, and prints the responses it receives. Normally, the listing is in the order that responses are received, but this order can be changed by specifying one of the options listed below. When *host* arguments are given, rather than broadcasting `rusers` will only query the list of specified hosts.

When the `-l` flag is given, a `rwho`(1C) style listing is used. In addition, if a user has not typed to the system for a minute or more, the idle time is reported.

A remote host will only respond if it is running the `rusersd` daemon, which is normally started up from `inetd`(1M).

OPTIONS

`-a`	Give a report for a machine even if no users are logged on.
`-h`	Sort alphabetically by host name.
`-i`	Sort by idle time.
`-l`	Give a longer listing in the style of `who`(1).
`-u`	Sort by number of users.

FILES

`/etc/servers`

SEE ALSO

`inetd`(1M), `rwho`(1C), `rusersd`(1M), `who`(1).

BUGS

Broadcasting does not work through gateways.

NAME

`rwall` – write to all users over a network

SYNOPSIS

`/usr/etc/rwall` *hostname*...
`/usr/etc/rwall -n` *netgroup*...
`/usr/etc/rwall -h hostname -n netgroup`

DESCRIPTION

`rwall` reads a message from standard input until EOF. It then sends this message, preceded by the line '`Broadcast Message` ...', to all users logged in on the specified host machines. With the `-n` option, it sends to the specified network groups, which are defined in `netgroup`(4).

A machine can only receive such a message if it is running `rwalld`(1M), which is normally started up by the daemon `inetd`(1M).

SEE ALSO

`wall`(1), `inetd`(1M), `rwalld`(1M), `shutdown`(1M), `netgroup`(4).

BUGS

The timeout is fairly short in order to be able to send to a large group of machines (some of which may be down) in a reasonable amount of time. Thus, the message may not get through to a heavily loaded machine.

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [-a]

DESCRIPTION

The rwho command produces output similar to who(1), but for all machines that are on the local network and running rwhod(1M). If no report has been received from a machine for eleven minutes, rwho assumes the machine is down and provides no information on its users.

If users haven't typed to the system for a minute or more, then rwho reports this idle time. However, if users haven't typed to the system for an hour or more, rwho doesn't display their status unless you use the -a flag.

Command line flags other than -a are ignored.

EXAMPLES

```
$ rwho -a ⤶

jones    sys10:tty00   Dec 17 08:07
wilson   sys04:tty03   Dec 17 08:02   2:15
smith    sys08:tty25   Dec 17 07:01
brown    sys02:tty15   Dec 17 08:03    :14
```

Displays users who are logged in on machines that are on the local area network and running rwhod, including those who have not typed to the system in an hour or more.

FILES

/var/spool/rwho/whod.*

SEE ALSO

ruptime(1C), rwhod(1M).

BUGS

The rwho command becomes unwieldy when the number of machines on the local net is large.

NAME

`sact` – print current SCCS file editing activity

SYNOPSIS

`sact` *files*

DESCRIPTION

`Sact` informs the user of any impending deltas to a named SCCS file. This situation occurs when `get`(1) with the `-e` option has been previously executed without a subsequent execution of `delta`(1). If a directory is named on the command line, `sact` treats each file in the directory as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, standard input lines are read as the names of SCCS files to be processed.

The output for each named file consists of five fields separated by spaces.

`Field 1`	Specifies the SID of a current delta in the SCCS file that will be changed to make the new delta.
`Field 2`	Specifies the SID for the new delta to be created.
`Field 3`	Contains the logname of the user who will make the delta (i.e., executed a `get` for editing).
`Field 4`	Contains the date that `get -e` was executed.
`Field 5`	Contains the time that `get -e` was executed.

DIAGNOSTICS

Use `help`(1) for explanations.

SEE ALSO

`delta`(1), `get`(1), `unget`(1).

NAME

`sar` – system activity reporter

SYNOPSIS

`sar` [`-ubdycwaqvmprA`] [`-o` *file*] *t* [*n*]

`sar` [`-ubdycwaqvmprA`] [`-s` *time*] [`-e` *time*] [`-i` *sec*] [`-f` *file*]

DESCRIPTION

The first case of `sar` samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If you specify the `-o` option, `sar` writes complete samples to *file* (in binary format) in addition to displaying them on the screen. The default value for *n* is 1.

In the second instance, with no sampling interval specified, `sar` extracts data from a previously recorded *file*, either the one specified by the `-f` option or, by default, the standard system activity daily data file `/usr/adm/sa/sa`*dd* for the current day *dd*.

The starting and ending times of the report can be bounded via the `-s` and `-e` *time* arguments of the form *hh*[:*mm*[:*ss*]]. The `-i` option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by the following options. Column headings that end in "`/s`" indicate an average rate per second over the interval (see NOTES, below). The default reporting option is `-u` if no others are specified.

`-a` Report use of file access system routines:

`iget/s`	number of inode entry searches per second (local files only);
`namei/s`	number of pathname searches per second;
`dirblk/s`	number of reads per second associated with buffering a portion of a directory file (local files only).

`-b` Report buffer activity:

`bread/s, bwrit/s`	average transfers per second of data between system buffers and disk or other block devices;
`lread/s, lwrit/s`	average accesses per second of system buffers;
`%rcache`	read cache hit ratio, i.e., the fraction of the number of logical reads which were found in the buffer cache (100% minus the ratio of `bread/s` to `lread/s`). This number is skewed due to the read-aheads performed by the operating system, which are counted in the `bread` value;
`%wcache`	write cache hit ratio, i.e., the fraction of the number of logical writes which were found in the buffer cache (100% minus the ratio of `bwrit/s` to `lwrit/s`);
`pread/s, pwrit/s`	average operations per second via raw (physical) device mechanism.

`-c` Report system calls:

`scall/s`	system calls per second of all types;
`sread/s, swrit/s, fork/s, exec/s`	specific system calls per second;
`rchar/s, wchar/s`	characters transferred per second by read and write system calls.

`-d` Report activity for each disk drive. When data is displayed, the device specification (e.g., `sd(insc@E(FFF8A000),0,0)`) represents a physical disk drive.

`%busy`	portion of time device was busy servicing a transfer request;
`avque`	average number of requests outstanding and being serviced during that time (measured only when the disk is busy);
`r+w/s, blks/s`	number of data transfers per second from or to devices, and number of 512-byte blocks transferred per second;
`avwait`	average time in milliseconds that a transfer request waits idly on the queue;
`avserv`	average time in milliseconds for a transfer request to be completed (which for disks includes seek rotational latency and data transfer times).

−m Report message and semaphore activities:

`msg/s, sema/s`	`msgsnd`(2) and `semop`(2) system calls issued per second.

−p Report paging activities:

`vflt/s`	address translation page faults per second (valid page not in memory);
`pflt/s`	page faults per second from protection errors (illegal access to page) or "copy-on-writes";
`pgfil/s`	page faults per second satisfied by reading pages from program files;
`rclm/s`	valid pages reclaimed per second for the free list.

−q Report average queue length while occupied, and percentage of time occupied:

`runq-sz`	number of bound and runnable processes;
`swpq-sz`	number of unbound runnable processes (the smaller this number, the better);
`%runocc, %swpocc`	instantaneous snapshots of whether the queue is occupied or not, 0% indicates not occupied, 100% indicates occupied.

−r Report unused memory pages and disk blocks:

`freemem`	the number of pages available to user processes (see `getpagesize`(2));
`freeswp`	the number of 512-byte disk blocks available in the paging area.

−u Report CPU usage.

`%usr, %sys, %idle`	portion of CPU time running in user mode, running in system mode, and otherwise idle.

−v Report status of text, process, inode and file tables:

`proc-sz, inod-sz, file-sz, lock-sz`	the number of entries used/allocated for the process table, the inode table, the file table, and the shared memory record table, evaluated once at the sampling point. Entries in the inode table, file table, and shared memory record table are allocated dynamically, so the number of entries in use is the same as the number of entries allocated;
`ov`	overflows occurring between sampling points.

−w Report swapping and switching activity:

`swpin/s`	the number of processes bound per second;

`bswin/s`	the number of page faults per second associated with user address space;
`swpot/s`	the number of processes unbound per second;
`bswot/s`	the number of pages that belonged to bound processes reclaimed per second;
`pswch/s`	process switches per second.

`-y` Report TTY device activity:

`rawch/s, canch/s, outch/s`	raw mode input character rate, input character rate processed by canon (see `termio`(7) and `tty`(7)), and output character rate;
`rcvin/s, xmtin/s, mdmin/s`	receive, transmit and modem interrupt rates.

`-A` Report all data. This supersedes all other options and is equivalent to `-udqbwcayvmpr`.

Restart entries, indicated by

```
dgux restarts
```

in the output, mark times when the system was rebooted (or the run level changed) and system activity counters reset to zero.

EXAMPLES

```
$ sar
dgux sys23 4.30 14.5 AViiON    06/08/90

00:00:05    %usr    %sys   %idle
01:00:04       3       2      95
02:00:04       3       2      94
03:00:04       3       2      96
04:00:04       3       2      96
05:00:03       3       2      96
06:00:04       3       2      96
07:00:04       3       2      96

Average        3       2      95
$
```

The system activity reporter, `sar`, will display cumulative activity statistics. In the above example, `sar` reports CPU usage from the standard system activity daily file for the current day.

```
$ sar -c
dgux sys23 4.30 14.5 AViiON    06/08/90

00:00:05 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
01:00:04       4       0       0     0.1     0.0       0       0
02:00:04       4       0       0     0.0     0.0       0       0
03:00:04       3       0       0     0.1     0.0       0       0
04:00:04       3       0       0     0.1     0.0       0       0
05:00:03       3       0       0     0.1     0.0       0       0
06:00:04       3       0       0     0.1     0.0       0       0
07:00:04       3       0       0     0.1     0.0       0       0
```

```
Average          3        0        0     0.1      0.0        0        0
$
```

In this example, `sar` reports system call usage from the standard system activity daily file for the current day.

```
$ sar -q
dgux sys23 4.30 14.5 AViiON    06/08/90

00:00:05 runq-sz %runocc swpq-sz %swpocc
01:00:04     1.0     100     1.0     100
02:00:04     1.0     100     1.0     100
03:00:04     1.0     100     1.0     100
04:00:04     1.0     100     1.0     100
05:00:03     1.0     100     1.0     100
06:00:04     1.0     100     1.0     100
07:00:04     1.0     100     1.0     100

Average      1.0     100     1.0     100
$
```

In this example, `sar` reports average queue length while occupied, and whether the queue is occupied or not at the time of the sample.

```
$ sar -v
dgux sys23 4.30 14.5 AViiON    06/08/90

00:00:05 proc-sz ov inod-sz ov file-sz ov lock-sz
01:00:04  69/256  0 240/240  0 102/102  0   0/  0
02:00:04  69/256  0 240/240  0 102/102  0   0/  0
03:00:04  69/256  0 242/242  0 102/102  0   0/  0
04:00:04  69/256  0 240/240  0 102/102  0   0/  0
05:00:03  69/256  0 240/240  0 102/102  0   0/  0
06:00:04  69/256  0 240/240  0 102/102  0   0/  0
07:00:04  69/256  0 240/240  0 102/102  0   0/  0
$
```

In this example, `sar` reports on the process, inode, file, and shared memory lock tables: the size of each table and overflows occurring between sampling points.

To watch CPU activity evolve for 10 minutes, simultaneously saving data to a file named `temp`:

```
    sar -o temp 60 10
```

To review disk activity recorded in file `temp`:

```
    sar -d -f temp
```

FILES

/usr/adm/sa/sa*dd*	daily data file, where *dd* are digits representing the day of the month.
/usr/lib/sa/sadc	data collection program.

NOTES

A sampling interval of less than 5 seconds is discouraged, for then the activity of `sar` itself may affect the sample.

Using `sar` with no sampling interval, causing it to read from a named file or the default daily file, presumes that something has been done to collect data in that file. Otherwise, only the restart entries will be displayed.

By reporting rates per second, `sar` smooths bursts of extreme activity and inactivity. For example, if a burst of 20 characters of output occurs within a one-second sample on an otherwise idle machine, `sar` will report an output character rate of 20 characters per second. If that same activity occurred within a ten-second sample, `sar` would report a rate of 2 characters per second.

BUGS

If more than one reporting option is specified, the headers are printed all together and the output may be difficult to read.

`sar` cannot be used to report from data files collected on systems that are not running the DG/UX System. The versions of `sar` and `sadc` released prior to DG/UX 4.30 are incompatible with later releases.

SEE ALSO

`sar`(1M) in the *System Manager's Reference for the DG/UX System*.

NAME

`script` – make typescript of a terminal session

SYNOPSIS

`script` [`-a`] [*filename*]

DESCRIPTION

`script` makes a typescript of everything printed on your terminal. The typescript is written to *filename*, or appended to *filename* if the `-a` option is given. If no file name is given, the typescript is saved in the file `typescript`.

The script ends when the forked shell exits or when Ctrl-D is typed.

SEE ALSO

`csh`(1), `ksh`(1), `sh`(1).

NOTES

`script` places *everything* that appears on the screen in the log file, including prompts.

NAME

sdiff – side-by-side difference program

SYNOPSIS

`sdiff` [*options* ...] *file1 file2*

DESCRIPTION

`Sdiff` uses the output of `diff`(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a `<` in the gutter if the line exists only in *file1*; a `>` in the gutter if the line exists only in *file2*; and a `|` for lines that are different.

For example:

```
x     |     y
a           a
b     <
c     <
d           d
      >     c
```

Options are:

`-w` *n*	Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
`-l`	Print only the left side of any lines that are identical.
`-s`	Do not print identical lines.
`-o` *output*	Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by `diff`(1), are printed; a set of differences share a common gutter character. After printing each set of differences, `sdiff` prompts the user with a `%` and waits for one of the following user-typed commands:

`l`	Append the left column to the output file
`r`	Append the right column to the output file
`s`	Turn on silent mode; do not print identical lines
`v`	Turn off silent mode
`e l`	Call the editor with the left column
`e r`	Call the editor with the right column
`e b`	Call the editor with the concatenation of left and right
`e`	Call the editor with a zero length file
`q`	Exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

International Features

`sdiff` can process files containing characters from supplementary code sets.

Option:

-w*n* The width specified by *n* is in columns, not characters. Multi-column characters which across the right margin of a line are displayed as ASCII spaces.

SEE ALSO

diff(1), ed(1).

NAME

sed – stream editor

SYNOPSIS

sed [-n] [-e *script*] [-f *sfile*] [*files*]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The -f option causes the script to be taken from file *sfile*; these options accumulate. If there is just one -e option and no -f options, the flag -e may be omitted. The -n option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[*address* [, *address*]]*function* [*arguments*]

In normal operation, sed cyclically copies a line of input into a *pattern space* (unless there is something left after a D command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under -n) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a $ that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of ed(1) modified thus:

In a context address, the construction *?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address xabc\xdefx, the second x stands for itself, so that the regular expression is abcxdef.

The escape sequence \n matches a new-line *embedded* in the pattern space.

A period . matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

You can apply editing commands only to non-selected pattern spaces with the negation function ! (below).

The following list of functions gives the maximum number of permissible addresses for each function in parentheses:

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) `a\`
text Append. Place *text* on the output before reading the next input line.

(2) `b` *label* Branch to the `:` command bearing *label*. If *label* is empty, branch to the end of the script.

(2) `c\`
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) `d` Delete the pattern space. Start the next cycle.

(2) `D` Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) `g` Replace the contents of the pattern space by the contents of the hold space.

(2) `G` Append the contents of the hold space to the pattern space.

(2) `h` Replace the contents of the hold space by the contents of the pattern space.

(2) `H` Append the contents of the pattern space to the hold space.

(1) `i\`
text Insert. Place *text* on the standard output.

(2) `l` List the pattern space on the standard output in an unambiguous form. Non-printing characters are displayed in octal notation, and long lines are folded.

(2) `n` Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) `N` Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) `p` Print. Copy the pattern space to the standard output.

(2) `P` Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) `q` Quit. Branch to the end of the script. Do not start a new cycle.

(1) `r` *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) `s`*/regular expression/replacement/flags*
Substitute the *replacement* string for instances of *regular expression* in the pattern space. Any character may be used instead of `/`. For a fuller description see `ed`(1). *Flags* is zero or more of:

- n — n = 1 to 512. Substitute for just the *n*th occurrence (on the line) of the *regular expression.*
- `g` — Global. Substitute for all nonoverlapping instances of the *regular expression*, not just the first one.
- `p` — Print the pattern space if a replacement was made.
- `w` *wfile* — Write. Append the pattern space to *wfile* if a replacement was made.

(2) `t` *label* Test. Branch to the `:` command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution

of a `t`. If *label* is empty, branch to the end of the script.

(2)`w` *wfile* Write. Append the pattern space to *wfile*.

(2)`x` Exchange the contents of the pattern and hold spaces.

(2)`y`*/string1/string2/*
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)`!` *function*
Don't. Apply the *function* (or group, if *function* is `{`) only to lines *not* selected by the address(es).

(0) `:` *label* Takes no action; it bears a *label* for `b` and `t` commands to branch to.

(1)`=` Place the current line number on the standard output as a line.

(2)`{` Execute the following commands through a matching `}` only when the pattern space is selected.

(0) An empty command is ignored.

(0)`#` If a `#` appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the `#` is an n, then the default output will be suppressed. The rest of the line after `#n` is also ignored. A script file must contain at least one non-comment line.

International Features

`sed` can process characters from supplementary code sets as well as ASCII characters.

Searches and pattern matching with regular expressions are performed on characters, not bytes.

Comments in *script* files can contain characters from supplementary code sets.

EXAMPLES

To change .H 2 at the beginning of a line to .H2 and insert a new line containing .PA after the .H2 line:

```
sed -e '/^\.H 2/N;s/^\.H 2\(.*\)\(\n\)/.H2\1\2.PA\2/' ch1.mm
```

To split before .PS each line that starts with .TC:

```
sed '/^\.TC/H;s/ \.PS.*//p;/TC/p;/TC/x;s/..* \.PS/.PS/' infile
```

SEE ALSO

awk(1), ed(1), grep(1).

NAME

`sh`, `jsh`, `rsh`, `restsh` – shell, the command programming language

SYNOPSIS

```
sh [ -acefhiknrstuvx ] [ args ]
jsh [ -acefhiknprstuvx ] [ args ]
rsh [ -acefhiknrstuvx ] [ args ]
restsh [ -acefhiknrstuvx ] [ args ]
```

DESCRIPTION

`Sh` is a command programming language that executes commands read from a terminal or a file. A file of commands must have read and execute permissions set in order for you to run it–see also `umask` under "Special Commands," below. The command `jsh` is an interface to the shell which provides all of the functionality of `sh` and enables Job Control (see "Job Control," below). `rsh` and `restsh` are restricted versions of the standard command interpreter `sh`; they 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. `sh` also provides `editread`, an optional interface used for editing command lines entered from the shell. It also provides a history facility that saves previously typed commands (see *Using the DG/UX System* for more information).

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters `*`, `@`, `#`, `?`, `-`, `$`, and `!`.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see `exec`(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see `signal`(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by `|` (or, for historical compatibility, by `^`). The standard output of each command but the last is connected by a `pipe`(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by `;`, `&`, `&&`, or `||`, and optionally terminated by `;` or `&`. Of these four symbols, `;` and `&` have equal precedence, which is lower than that of `&&` and `||`. The symbols `&&` and `||` also have equal precedence. A semicolon (`;`) causes sequential execution of the preceding pipeline; an ampersand (`&`) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol `&&` (`||`) executes the *list* following it only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

`for` *name* [`in` *word* ...] `do` *list* `done`

Each time a *for* command is executed, *name* is set to the next *word* taken from the `in` *word* list. If `in` *word* ... is omitted, the `for` command executes the `do` *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

`case` *word* `in` [*pattern* [|*pattern*] ... *list* `;;`] ... `esac`
: A `case` command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for filename generation (see "Filename Generation") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

`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, the `if` command returns a zero exit status.

`while` *list* `do` *list* `done`
: A `while` command repeatedly executes the `while` *list* and, if the exit status of the last command in the list is zero, executes the `do` *list*; otherwise the loop terminates. If no commands in the `do` *list* are executed, `while` returns a zero exit status; use `until` in place of `while` to negate the loop termination test.

`(`*list*`)`
: Execute *list* in a sub-shell.

`{`*list*`;}`
: *list* is simply executed.

name `()` `{`*list*`;}`
: Define a function referenced by *name*. The body of the function is the *list* of commands between `{` and `}`. Execution of functions is described below (see "Execution").

The following words are recognized only when they are the first word of a command and when they are not quoted:

```
if  then  else  elif  fi  case  esac  for  while  until  do
done  {  }
```

Comments

`#` before a word causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (`` ` ` ``) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution

The character `$` introduces substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is positional. Positional parameters may be assigned values by `set`. Keyword parameters (also known as variables) may be assigned values by writing:

name=value [*name=value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

`${`*parameter*`}`
: The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is `*` or `@`, all the positional parameters, starting with `$1`, are substituted (separated by spaces). Parameter `$0` is set from argument zero when the shell is invoked.

`${`*parameter*`:-`*word*`}`
: If *parameter* is set and is non-null, substitute its value; otherwise, substitute *word*.

`${`*parameter*`:=`*word*`}`
: If *parameter* is not set or is null, set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

`${`*parameter*`:?`*word*`}`
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

`${`*parameter*`:+`*word*`}`
: If *parameter* is set and is non-null, substitute *word*; otherwise, substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, `pwd` is executed only if `d` is not set or is null:

```
    echo ${d:-`pwd`}
```

If the colon (`:`) is omitted from the above expressions, the shell only checks whether *parameter* is set.

The shell sets these parameters automatically:

#	The number of positional parameters in decimal.
–	Flags supplied to the shell on invocation or by the `set` command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

`HOME` The default argument (home directory) for the `cd` command.

`PATH` The search path for commands (see "Execution," below). The user may not change `PATH` if executing under `rsh`.

`CDPATH` The search path for the `cd` command.

`MAIL` If this parameter is set to the name of a mail file *and* the `MAILPATH` parameter is not set, the shell informs the user of the arrival of mail in the specified file.

`MAILCHECK`
: This parameter specifies how often (in seconds) the shell will check for mail in the files specified by the `MAILPATH` or `MAIL` parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

`MAILPATH`	A colon (`:`) separated list of filenames. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each filename can be followed by `%` and a message that will be printed when the modification time changes. The default message is `you have mail`.
`PS1`	Primary prompt string, by default `$`.
`PS2`	Secondary prompt string, by default `>`.
`IFS`	Internal field separators, normally `space`, `tab`, and `new-line`.
`SHACCT`	If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as `acctcom`(1) and `acctcms`(1M) can be used to analyze the data collected.
`SHELL`	When the shell is invoked, it scans the environment (see "Environment," below) for this name. If it is found and there is an `r` in the filename part of its value, the shell becomes a restricted shell.

The shell gives default values to `PATH`, `PS1`, `PS2`, `MAILCHECK` and `IFS`. `HOME` and `MAIL` are set by `login`(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in `IFS`) and split into distinct arguments where such characters are found. Explicit null arguments (`""` or `''`) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Filename Generation

Following substitution, each command *word* is scanned for the characters `*`, `?`, and `[`. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted filenames that match the pattern. If no filename matches the pattern, the word is left unchanged. The character `.` at the start of a filename or immediately following a `/`, as well as the character `/` itself, must be matched explicitly.

`*`	Matches any string, including the null string.
`?`	Matches any single character.
`[ . . . ]`	Matches any one of the enclosed characters. A pair of characters separated by `-` matches any character lexically between the pair, inclusive. If the first character following the opening `[` is a `!`, any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and terminate a word unless quoted:

```
; & ( ) | ^ < > new-line space tab
```

You can make a character stand for itself by preceding it with a `\`. This is called *quoting*. The pair `\new-line` is ignored. All characters enclosed between a pair of single quote marks (`''`), except a single quote, are quoted. Inside double quote marks (`""`), parameter and command substitution occurs and `\` quotes the characters `\`, `` ` ``, `"`, and `$`. `"$*"` is equivalent to `"$1 $2 ..."`, whereas `"$@"` is equivalent to `"$1" "$2" . . . .`

Prompting

When used interactively, the shell prompts with the value of `PS1` before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of `PS2`) is issued.

Input/Output

Before a command is executed, you can redirect its input and output using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command*. They are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

<*word*	Use file *word* as standard input (file descriptor 0).
>*word*	Use file *word* as standard output (file descriptor 1). If the file does not exist, it is created; otherwise, it is truncated to zero length.
>>*word*	Use file *word* as standard output. If the file exists, output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
«[-]*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) `\new-line` is ignored, and \ must be used to quote the characters \, $, `, and the first character of *word*. If - is appended to «, all leading tabs are stripped from *word* and from the document.
<&*digit*	Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using >&*digit*.
<&-	The standard input is closed. Similarly for the standard output using >&-.

Note that when the shell creates a file, the base mode is 666, rather than 777. The mode is then filtered through the current `umask`. See `umask` under "Special Commands," below, and `umask`(1).

If any of the above is preceded by a digit, the file descriptor associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections from left to right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file *xxx*. It then associates file descriptor 2 with *xxx*. If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by &, the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see `environ`(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the `export` command is used to bind the shell's parameter to the environment (see also `set -a`). A parameter may be removed from the environment with the `unset` command.

The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by `unset`, plus any modifications or additions, all of which must be noted in `export` commands.

You can augment the environment for any *simple-command* by prefixing it with one or more assignments to parameters. Thus:

```
TERM=605x cmd
```

and

```
(export TERM; TERM=605x; cmd )
```

are equivalent (as far as the execution of *cmd* is concerned).

If the `-k` flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints `a=b c` and `c`:

```
echo a=b c
set -k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by `&`; otherwise, signals have the values inherited by the shell from its parent, with the exception of signal 11. See also the `trap` command below.

Execution

Each time a command is executed, the above substitutions are made. If the command name matches one of the *special commands* listed below, it is executed in the shell process. If the command name does not match a *special command* but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters `$1, $2, ....` are set to the arguments of the function. If the command name matches neither a *special command* nor the name of a defined function, a new process is created and the system tries to execute the command using `exec`(2).

The shell parameter `PATH` defines the search path for the directory containing the command. Alternative directory names are separated by a colon (`:`). The default path is `:/bin:/usr/bin` (specifying the current directory, `/bin`, and `/usr/bin`, in that order). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list.

If the command name contains a `/`, the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an executable program (binary) file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a subshell.

The shell remembers a command's location in the search path (to avoid having to search through your path again should you invoke the command again later). If the command was found in a relative directory, however, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the `PATH` variable is changed or the `hash -r` command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by `PATH` finds the directory containing *file*. You must have read and execute permission for the *file*.

`break` [*n*]
Exit from the enclosing `for` or `while` loop, if any. If *n* is specified, break *n* levels.

`continue` [*n*]
Resume the next iteration of the enclosing `for` or `while` loop. If *n* is specified, resume at the *n*th enclosing loop.

`cd` [*arg*]
Change the current directory to *arg*. The shell parameter `HOME` is the default *arg*. The shell parameter `CDPATH` defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (`:`). The default path is *null* (specifying the current directory). Note that the current directory is specified by a null pathname, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a `/`, the search path is not used; otherwise, each directory in the path is searched for *arg*. The `cd` command may not be executed by `rsh`.

`echo` [*arg* ...]
Echo arguments. See `echo`(1) for usage and description.

`eval` [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

`exec` [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

`exit` [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed. An end-of-file will also cause the shell to exit.

`export` [*name* ...]
The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

`hash` [`-r`] [*name* ...]
: This option is available in DG/UX only. The shell finds and remembers the location in the search path of each command specified by *name*. The `-r` option makes the shell forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. Some situations require that the stored location of a command be recalculated. These commands are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* is incremented when the recalculation is done.

`newgrp` [*arg* ...]
: Equivalent to `exec newgrp` *arg* .... See `newgrp(1)` .

`pwd` Print the current working directory. This is a `sh` built-in command that is not the same as the `/bin/pwd` command discussed in the `pwd(1)` manual page. The `/bin/pwd` program exists for the `csh`—it does not have a built-in `pwd` command.

`read` [*name* ...]
: One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

`readonly` [*name* ...]
: The given *name*s are marked *readonly* and the values of these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

`return` [*n*]
: Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

`set` [`--aefhkntuvx` [*arg* ...]]

`-a`	Mark variables that are modified or created for export.
`-e`	Exit immediately if a command exits with a non-zero exit status.
`-f`	Disable filename generation.
`-h`	Find and remember function commands as functions are defined (function commands are normally located when the function is executed).
`-k`	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
`-n`	Read commands but do not execute them.
`-t`	Exit after reading and executing one command.
`-u`	Treat unset variables as an error when substituting.
`-v`	Print shell input lines as they are read.
`-x`	Print commands and their arguments as they are executed.
`--`	Do not change any of the flags; useful in setting `$1` to `-`.

Using + rather than - turns these flags off. These flags can also be used upon invocation of the shell. The current setting of flags may be found in

`$-`. The remaining arguments are positional parameters and are assigned, in order, to `$1`, `$2`, ... If no arguments are given, the values of all names are printed.

`shift` [*n*]
: The positional parameters from `$n+1` ... are renamed `$1` .... If *n* is not given, it is assumed to be 1.

`test`
: Evaluate conditional expressions. See `test`(1) for usage and description.

`times`
: Print the accumulated user and system times for processes run from the shell.

`trap` [*arg*] [*n*] ...
: Read and execute the command *arg* when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) or signal 18 (termination of child process) produces an error. If *arg* is absent, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The `trap` command with no arguments prints a list of commands associated with each signal number.

`type` [*name* ...]
: For each *name*, indicate how it would be interpreted if used as a command name.

`ulimit` [`-`[`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.

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)

`umask` [*nnn*]
The user file-creation mask is set to *nnn* (see `umask`(2)). If *nnn* is omitted, the current value of the mask is printed. Note that the shell and any programs running under the shell, like `ed`(1), create files with a maximum permission of 666, even if you set the mask to 000. The mask value is subtracted from 777 to arrive at the final mode, however. A mask of 012 yields a mode of 665, for example. You must use `chmod` to add the execution permission. This is especially important if you are creating a shell program, since it must have read and execute permissions in order to run.

`wait` [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through `exec`(2) and the first character of argument zero is `-`, commands are initially read from `/etc/profile` and from `$HOME/.profile`, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as `/bin/sh`. The flags below are interpreted by the shell on invocation only. Unless the `-c` or `-s` flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

`-c` *string*	Commands are read from *string*.
`-s`	If no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.
`-i`	If the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that `kill 0` does not kill an interactive shell) and INTERRUPT is caught and ignored (so that `wait` is interruptible). In all cases, QUIT is ignored by the shell.
`-r`	The shell is a restricted shell.

The remaining flags and arguments are described under the `set` command above.

Job Control (jsh)

When the shell is invoked as `jsh`, Job Control is enabled in addition to all of the functionality described previously for `sh`. Typically Job Control is enabled for the interactive shell only. Non-interactive shells typically do not benefit from the added functionality of Job Control.

With Job Control enabled every command or pipeline the user enters at the terminal is called a *job*. All jobs exist in one of the following states: foreground, background or stopped. These terms are defined as follows: 1) a job in the foreground has read and write access to the controlling terminal; 2) a job in the background is denied read access and has conditional write access to the controlling terminal [see `stty`(1)]; 3) a stopped job is a job that has been placed in a suspended state, usually as a result of a `SIGTSTP` signal [see `signal`(5)].

Every job that the shell starts is assigned a positive integer, called a *job number* which is tracked by the shell and will be used as an identifier to indicate a specific job. Additionally the shell keeps track of the *current* and *previous* jobs. The *current job* is the most recent job to be started or restarted. The *previous job* is the first non-current job.

The acceptable syntax for a Job Identifier is of the form:

%*jobid*

where, *jobid* may be specified in any of the following formats:

`%` or `+`	for the current job
–	for the previous job
?*string*	specify the job for which the command line uniquely contains *string*.
n	for job number *n*, where *n* is a job number
pref	where *pref* is a unique prefix of the command name (for example, if the command `ls -l foo` were running in the background, it could be referred to as `%ls`); *pref* cannot contain blanks unless it is quoted.

When Job Control is enabled, the following commands are added to the user's environment to manipulate jobs:

`bg` [%*jobid* . . .]
: Resumes the execution of a stopped job in the background. If %*jobid* is omitted the current job is assumed.

`fg` [%*jobid* . . .]
: Resumes the execution of a stopped job in the foreground, also moves an executing background job into the foreground. If %*jobid* is omitted the current job is assumed.

`jobs` [`-p`|`-l`] [%*jobid* ...]

`jobs -x` *command* [*arguments*]
: Reports all jobs that are stopped or executing in the background. If %*jobid* is omitted, all jobs that are stopped or running in the background will be reported. The following options will modify/enhance the output of `jobs`:

`-l`	Report the process group ID and working directory of the jobs.
`-p`	Report only the process group ID of the jobs.
`-x`	Replace any *jobid* found in *command* or *arguments* with the corresponding process group ID, and then execute *command* passing it *arguments*.

`kill` [`-signal`] %*jobid*
: Builtin version of `kill` to provide the functionality of the `kill` command for processes identified with a *jobid*.

`stop` %*jobid* . . .
: Stops the execution of a background job(s).

`suspend`
: Stops the execution of the current shell (but not if it is the login shell).

`wait` [%*jobid* . . .]
: `wait` builtin accepts a job identifier. If %*jobid* is omitted `wait` behaves as described above under `Special Commands`.

Rsh Only

Rsh sets up login names and execution environments that are more controlled than those of the standard shell. rsh is identical to sh, except that the following are disallowed:

> changing directory (see cd(1))
> setting the value of $PATH and $SHELL
> specifying command names containing /
> redirecting output (> and >>)

The restrictions above are enforced after .profile is interpreted.

When a command to be executed is a shell procedure, rsh invokes sh to execute it. Thus, you can give procedures to the end-user shell 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 to give the writer of the .profile complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., /usr/rbin) that can be safely invoked by rsh. Some systems also provide a restricted editor red.

EXIT CODES

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively, execution of the shell file is abandoned except under special conditions:

> The "echo" and "pwd" built-in commands have counterparts in the /bin directory. If you use /bin/echo or /bin/pwd in a shell script rather than the built-in echo or bin, execution of the script will continue after an error.

Otherwise, the shell returns the exit status of the last command executed (see also the exit command above).

jsh Only

If the shell is invoked as jsh and an attempt is made to exit the shell while there are stopped jobs, the shell issues one warning:

```
There are stopped jobs.
```

This is the only message. If another exit attempt is made, and there are still stopped jobs they will be sent a SIGHUP signal from the kernel and the shell is exited.

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

acctcom(1), cd(1), echo(1), env(1), login(1), newgrp(1), test(1), umask(1).
acctcms(1M) in the *System Manager's Reference for the DG/UX System*
dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), profile(4), environ(5) in the *Programmer's Reference for the DG/UX System*

See *Using the DG/UX System* for complete information on using the Bourne shell.

NOTES

If a command is executed and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell continues to execute (with `exec`) the original command. Use the `hash` command to correct this situation.

If you move the current directory or one above it, `pwd` may not give the correct response. Use the `cd` command with a full pathname to correct this situation.

If /usr/ucb (a link to /usr/bin) is found on the path before /usr/bin, the behavior of the built in commands echo and test change to act like a BSD system.

NAME

`shl` – shell layer manager

SYNOPSIS

`shl`

DESCRIPTION

`Shl` lets a user interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* can receive input from the keyboard. Other layers trying to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To block the output of a layer when it is not current, set the `stty` option `loblk` within the layer.

The `stty` character `swtch` (set to ^Z if NUL) switches control to `shl` from a layer. `Shl` has its own prompt, >>>, to help distinguish it from a layer.

A *layer* is a shell that has been bound to a virtual tty device (`/dev/ttyp??`). The virtual device can be manipulated like a real tty device using `stty`(1) and `ioctl`(2). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* `(1)` through `(7)` cannot be used when creating a layer. They are used by `shl` when no name is supplied. They may be abbreviated to just the digit.

Commands

You can issue the following commands from the `shl` prompt level. Any unique prefix is accepted.

`create` [*name*]
: Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is a digit (1,2...7) bound to the layer. The shell prompt variable `PS1` is set to the name of the layer followed by a space. A maximum of seven layers can be created.

`block` *name* [*name* ...]
: For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option `loblk` within the layer.

`delete` *name* [*name* ...]
: For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see `signal`(2)).

`help` (or `?`)
: Print the syntax of the `shl` commands.

`layers` [`-l`] [*name* ...]
: For each *name*, list the layer name and its process group. The `-l` option produces a `ps`(1)-like listing. If no arguments are given, information is presented for all existing layers.

`resume` [*name*]
: Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

`toggle`
: Resume the layer that was current before the last current layer.

`unblock` *name* [*name* ...]
: For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option `loblk` within the layer.

`quit` Exit `shl`. All layers are sent the SIGHUP signal.

name Make the layer referenced by *name* the current layer.

Using shl with the C Shell

If you are using the `csh(1)`, which by default uses Ctrl-Z as the suspend character, and you want to reset the switch character for `shl`, you can create an alias such as the following:

```
alias shl 'stty old; stty swtch \^g; stty line 1; /usr/bin/shl'
```

International Features

shells managed with `shl` provide the same functionality with characters from supplementary code sets as the current shell.

Layer names must be specified using ASCII characters only.

FILES

`/dev/ttyp??`	Virtual tty devices
`$SHELL`	Variable containing pathname of the shell to use (default is `/bin/sh`).

SEE ALSO

`sh(1)`, `stty(1)`.
`ioctl(2)`, `signal(2)` in the *Programmer's Reference for the DG/UX System*
`pty(7)` in the *System Manager's Reference for the DG/UX System*

NOTES

If you are using `shl` from an xterm window (TERM=xterm), then `shl` will not be able to pass window size information down to each layer. This may cause problems when using utilities such as `vi(1)` and `more(1)`. To resolve this, execute the `resize(1)` command in each layer.

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep *time*

DESCRIPTION

Sleep suspends execution for *time* seconds. You can use it to execute a command after a certain amount of time, as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

EXAMPLES

```
$ sleep 60; date
```

Waits about 60 seconds, then executes the date(1) command. Using this format of the command, your current shell waits.

```
$ ( sleep 60; date )&
```

Waits about 60 seconds, then executes the date(1) command. This format of the command starts a background shell and allows your current shell to continue. For most applications of sleep this is probably prefered.

```
$ while true; do ps -a; sleep 10; done &
```

Executes the ps command approximately every 10 seconds. Prints information about all processes associated with the terminal except for process group leaders.

SEE ALSO

alarm(2), sleep(3C) in the *Programmer's Reference for the DG/UX System*

NAME

`sort` – sort and/or merge files

SYNOPSIS

`sort` [`-cmu`] [`-o`*output*] [`-y`*kmem*] [`-z`*recsz*] [`-dfiMnr`] [`-bt`*x*] [+*pos1* [-*pos2*]] [*files*]

DESCRIPTION

`Sort` sorts lines of all the named files and writes the result on the standard output. The standard input is read if – is used as a file name or if no input files are named.

The default sort order is by ASCII code, wherein all capital letters come before lowercase letters. To get more conventional alphabetical sorting, use the `-f` or `-d` option. Numerical and by-month sorts are also possible; see the options below.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key: the entire input line.

The following options alter the default behavior:

`-c` Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

`-m` Merge only, the input files are already sorted.

`-u` Unique: suppress all but one in each set of lines having equal keys.

`-o`*output*
: The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between `-o` and *output*.

`-y`*kmem*
: The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, `sort` begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, `sort` will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, `-y0` is guaranteed to start with minimum memory. By convention, `-y` (with no argument) starts with maximum memory.

`-z`*recsz*
: The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the `-c` or `-m` options, a popular system default size will be used. Lines longer than the buffer size will terminate `sort` abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

`-M` Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that JAN < FEB < ... < DEC. Invalid fields compare low to JAN. The `-M` option implies the `-b` option (see above).

`-d` Dictionary order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

`-f` Fold lower case letters into upper case.

`-i` Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

`-n` An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The `-n` option implies the `-b` option (see below). Note that the `-b` option is effective only when restricted sort key specifications are in effect.

`-r` Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 −pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *−pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. You can alter the treatment of field separators using these options:

`-t`*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

`-b` Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the `-b` option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the `b` flag may be attached independently to each *+pos1* or *−pos2* argument (see below).

Pos1 and *pos2* each have the form *m*`.`*n* optionally followed by one or more of the flags `bdfinr`. A starting position specified by *+m*`.`*n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing `.`*n* means `.`0, indicating the first character of the *m*+1st field. If the `b` flag is in effect, *n* is counted from the first non-blank in the *m*+1st field; *+m*`.0b` refers to the first non-blank character in the *m*+1st field.

A last position specified by *−m*`.`*n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing `.`*n* means `.`0, indicating the last character of the *m*th field. If the `b` flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *−m*`.`1b refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

International Features

`sort` can process characters from multibyte code sets. Specifying the `-o` option allows output of characters from multibyte code sets.

Options:

`-d` No comparison is performed for multibyte characters.

`-f`	Only applies to single byte characters.
`-i`	All multibyte characters are also ignored.
+pos1	*n* in *pos1*, *pos2* format `m.n` is the position in the column, not character, after the last character of the *m* field.
-pos2	*n* in *pos1*, *pos2* format `m.n` is the position in the column, not character, after the last character of the *m* field.
`-t`*x*	A character from a multibyte code set can be specified in *x* as the field separator.

Characters from multibyte code sets are collated in code order.

For locales other than C, which do contain multibyte code sets, sort will collate using the collation rules of the given locale. In this case, sort uses the `strcoll`(3C) function to compare entire lines and individual sort keys.

Most non-C locales do not include space characters or punctuation characters in their collation tables, so these characters will be ignored in the `strcoll`(3C) comparison. This results in sort behaving as though the -d (dictionary order) and -b (ignore blanks) options are specified.

If you are working in a non-C locale and would like traditional sort behavior, you can set the environment variable LC_COLLATE to C.

EXAMPLES

Sort the contents of `infile` with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of `infile1` and `infile2`, placing the output in `outfile` and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of `infile1` and `infile2` using the first non-blank character of the second field as the sort key:

```
sort -r +1.0b -1.1b infile1 infile2
```

Print the password file (`passwd`(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort -t: +2n -3 /etc/passwd
```

Print the lines of the already sorted file `infile`, suppressing all but the first occurrence of lines having the same third field (the options `-um` with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

Sort, the contents of `infile1` in Spanish order.

```
LANG=es_ES; export LANG
sort infile
```

FILES

`/usr/tmp/stm???`

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the `-c` option. When the last line of an input file is missing a new-line character, `sort` appends one, prints a warning message, and continues.

`Sort` does not guarantee preservation of relative line ordering on equal keys.

SEE ALSO

`comm`(1), `join`(1), `uniq`(1).

NAME

`spell`, `hashmake`, `spellin`, `hashcheck` – find spelling errors

SYNOPSIS

`spell [ -v ] [ -b ] [ -x ] [ -l ] [ -i ] [` *+local_file* `] [` *files* `]`

`/usr/lib/spell/hashmake`

`/usr/lib/spell/spellin` *n*

`/usr/lib/spell/hashcheck` *spelling_list*

DESCRIPTION

`Spell` collects words from the named *files* and looks them up in a spelling list. Words not in the list or words not derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

`Spell` ignores most `troff`(1), `tbl`(1), and `eqn`(1) constructions.

Under the `-v` option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the `-b` option, British spelling is checked. Besides preferring `centre`, `colour`, `programme`, `speciality`, `travelled`, etc., this option insists upon `-ise` in words like `standardise`.

Under the `-x` option, every plausible stem is printed with = for each word.

By default, `spell` (like `deroff`(1)) follows chains of included files (`.so` and `.nx` `troff`(1) requests), *unless* the names of such included files begin with `/usr/lib`. Under the `-l` option, `spell` will follow the chains of *all* included files. Under the `-i` option, `spell` will ignore all chains of included files.

Under the *+local_file* option, words found in *local_file* are removed from `spell`'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to `spell`'s own spelling list) for each job.

The spelling list is based on many sources. Although it is more haphazard than an ordinary dictionary, `spell` is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Copies of all output from `spell` are accumulated in the default history file. You can specify an alternate history file by setting the history file name argument. For the name of the history file argument and the default setting (see "FILES").

Three routines help maintain and check `spell`'s hash lists:

`hashmake`	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
`spellin` *n*	Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error.
`hashcheck`	Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

EXAMPLES

```
$ cat spellcheck
This is a sample file taht can be used to test teh spell comand.
Obviously there are some speling errers.
$ spell spellcheck > spellout1
$ cat spellout1
comand
errers
speling
taht
teh
$
```

This example shows the contents of a file with some spelling errors. The `spell` command is used to check the file for errors, and the output is sent to a file called `spellout1`. As you can see, all of the misspelled words are in this file.

```
$ spell -v spellcheck >spellout2
$ cat spellout2
comand
errers
speling
taht
teh
+ly   Obviously
+d    used
$
```

This example uses the `-v` option. With this option, the misspelled words are listed as well as all of the words not literally in the spelling list. The plausible derivations of these words are shown.

FILES

`H_SPELL=/var/adm/spellhist` History file.

SEE ALSO

`sed`(1), `sort`(1), `tee`(1).
`deroff`(1), `eqn`(1), `tbl`(1), `troff`(1) in the Documenter's Toolkit.
Using the Documenter's Tool Kit on the DG/UX System

BUGS

The spelling list's coverage is uneven. New installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via `spellin`.

NAME

`spline` – interpolate smooth curve

SYNOPSIS

`spline` [`-a` [*spc*]] [`-k` *const*] [`-n` *interval*] [`-p`] [`-x` [*lower* [*upper*]]]

where:

spc	Horizontal spacing; default = 1
const	A numeric constant
interval	Number of intervals between lower and upper limits
lower	Lower limit of the abscissa; default = 0
upper	Upper limit of the abscissa

DESCRIPTION

`Spline` takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted.

Options are:

`-a` Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1.0 if next argument is not a number.

`-k` The constant k used in the boundary value computation

$$y_0'' = ky_1'',\ y_n'' = ky_{n-1}''$$

is set by the next argument. By default $k = 0$.

`-n` Space output points so that approximately n intervals occur between the lower and upper x limits. If n is less then 1, or n is not integral, the default value is used. (Default $n = 100$)

`-p` Make output periodic, i.e. match derivatives at ends. First and last input values should normally agree.

`-x` The arguments, if present, represent the lower and upper x limits, respectively. Normally these limits are calculated from the data. If both are given, the lower limit must be less than the upper limit or the option is ignored. Automatic generation of the abscissas start at the lower limit.

DIAGNOSTICS

When data is not strictly monotone in x, `spline` reproduces the input without interpolating extra points. In the absence of the `-a` option, if the last abscissa supplied does not have a corresponding ordinate, it is ignored.

BUGS

A limit of 1000 input points is enforced silently.

NAME

split – split a file into pieces

SYNOPSIS

split [*–n*] [*file* [*name*]]

DESCRIPTION

`Split` reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with `aa` appended; the name of the last is `zz` (a maximum of 676 files). *name* cannot be longer than 252 characters. If no output name is given, `x` is the default.

If no input file is given, or if – is given instead, then the standard input file is used.

EXAMPLES

```
$ split -4 longfile longfile
```

The `split` command takes the file `longfile` and splits it into files 4 lines long. The new files will be named `longfileaa`, `longfileab`, and so on to `longfilezz`, if needed. If no output name were specified, the names would default to `xaa`, `xab`, and so on.

SEE ALSO

`bfs`(1), `csplit`(1).

NAME

`srchtxt` – display contents of, or search for a text string in, message data bases

SYNOPSIS

`srchtxt` [`-s`] [`-l` *locale*] [`-m` *msgfile*`,` ...] [*text*]

DESCRIPTION

`srchtxt` can search the characters from supplementary code sets of text strings in the message handling facility for a text string.

FILES

`/usr/lib/locale/`*locale*`/LC_MESSAGES/*`	files created by `mkmsgs`.

SEE ALSO

`exstr`(1), `mkmsgs`(1), `gettxt`(3C).

NAME

starter – information for beginning users

SYNOPSIS

[help] starter

DESCRIPTION

The DG/UX system Help Facility command `starter` provides three categories of information about the DG/UX system to assist new users.

The three categories are:

- Commands a new user should learn first.
- Documents that are important for beginners.
- Local environment information.

The user may choose one of the above categories by entering its corresponding letter (given in the menu), or the user may quit and exit to the shell by typing `q`. When a category is chosen, the user will receive one or more pages of information pertaining to it.

From any screen in the Help Facility, a user may execute a command via the shell (`sh`(1)) by typing a `!` and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), set the shell environment variable `SCROLL` to `no`. Do this in the Bourne shell, `sh`(1), by adding the following line to your `.profile` file (see `profile`(4)):

```
SCROLL=no; export SCROLL
```

In the `csh`(1), add the following line to your `.login` file:

```
setenv SCROLL no
```

If you later decide that you prefer scrolling, set `SCROLL` to `yes`.

Information on each of the other Help Facility commands (`locate`, `usage`, `glossary`, and `help`) is located on their respective manual pages.

SEE ALSO

`glossary`(1), `help`(1), `locate`(1), `csh`(1), `sh`(1), `usage`(1).
`term`(5) in the *Programmer's Reference for the DG/UX System*.

NOTES

If the shell variable `TERM` [see `sh`(1)] is not set in the user's `.profile` file, `TERM` defaults to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to `term`(5).

NAME

`strchg`, `strconf` – change or query stream configuration

SYNOPSIS

`strchg -h` *module*[,*module* ...]

`strchg -p [ -a | -u` *module* `]`

`strchg -f` *file*

`strconf [ -t | -m` *module* `]`

where:

module The name of a STREAMS module

DESCRIPTION

These commands alter or query the configuration of the stream associated with the user's standard input. The `strchg` command pushes modules on and/or pops modules off the stream. Only the super-user or owner of a STREAMS device may alter the configuration of that stream.

The `strconf` command queries the configuration of the stream. Invoked without any arguments, `strconf` prints a list of all the modules in the stream as well as the topmost driver. The list is printed with one name per line where the first name printed is the topmost module on the stream (if one exists) and the last item printed is the name of the driver.

Options for strchg

`-h` Push modules onto a stream. With this option `strchg` takes as arguments the names of one or more pushable streams modules. These modules are pushed in order; that is, the leftmost *module* in the command line is pushed first.

`-p` Pop modules off the stream. With no other options, `strchg` pops the topmost module from the stream. The `-a` and `-u` options modify this behavior.

`-f` *file* Modify the set of modules on the stream. The *file* argument specifies a text file that contains a list of modules representing the desired configuration. Each module name must appear on a separate line where the first name represents the topmost module and the last name represents the module that should be closest to the driver. The `strchg` command will determine the current configuration of the stream and pop and push the necessary modules in order to end up with the desired configuration.

`-a` With the `-p` option, pop all the modules above the topmost driver.

`-u` *module*
With the `-p` option, pop all modules above but not including *module* off the stream.

The `-h`, `-f` and `-p` options are mutually exclusive. The `-a` and `-u` options are also mutually exclusive.

Options for strconf

`-t` Print only the topmost module (if one exists).

`-m` Determine if the named *module* is present on a stream. If it is, `strconf` prints the message `yes` and returns zero. If not, `strconf` prints the message `no` and returns a non-zero value.

The `-t` and `-m` options are mutually exclusive.

EXAMPLES

```
strchg -h ldterm
```

Push the module `ldterm` on the stream associated with the user's standard input.

```
strchg -p < /dev/tty24
```

Pop the topmost module from the stream associated with `/dev/tty24`. The user must be the owner of this device or the super-user.

```
strchg -f fileconf
```

If the file `fileconf` contains the following:

```
compat
ldterm
ptem
```

then this example will configure the user's standard input stream so that the module `ptem` is pushed over the driver, followed by `ldterm`, with `compat` closest to the stream head.

```
strconf
```

List the modules and topmost driver on the stream. For a stream that has only the module `ldterm` pushed above the `syac` driver, this example would produce the following output:

```
ldterm
syac
```

```
strconf -m ldterm
```

Ask if `ldterm` is on the stream. If so, this example produces the following output while returning an exit status of 0:

```
yes
```

DIAGNOSTICS

`strchg` returns zero on success. It prints an error message and returns non-zero status for various error conditions, including usage error, bad module name, too many modules to push, failure of an `ioctl`(2) system call on the stream, or failure to open the *file* argument of the `-f` option.

`strconf` returns zero on success (for the `-m` or `-t` option, "success" means the named or topmost module is present). It returns a non-zero status if invoked with the `-m` or `-t` option and the module is not present. It prints an error message and returns non-zero status for various error conditions, including usage error or failure of an `ioctl`(2) system call on the stream.

SEE ALSO

`autopush`(1M), `streamio`(7)

NOTES

If the user is neither the owner of the stream nor the super-user, the `strchg` command will fail. If the user does not have read permissions on the stream and is not the super-user, the `strconf` command will fail.

If modules are pushed in the wrong order, one could end up with a stream that does not function as expected. For TTY's, if the line discipline module (`ldterm`) is not pushed in the correct place, one could have a terminal that does not respond to any commands.

NAME

`strings` – find the printable strings in an object or other binary file

SYNOPSIS

`strings` [`-o`] [*-number*] *file* ...

DESCRIPTION

`Strings` looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. If the `-o` option is given, then each string is preceded by its offset in the file. If the *-number* option is given, the number is used as the minimum string length rather than 4.

`Strings` is useful for identifying random object files and many other things.

EXAMPLES

```
$ cat file.c ↵

main()
{
printf("hello world\n");
}
```

The `cat` command lists the contents of `file.c`.

```
$ strings file.o ↵

.text
 .data
@.tdesc
hello world
.file
file.c
.text
.data
.tdesc
@LC0
_main
@Ltb0
_printf
@Lte0
gcc2_compiled.
```

This command lists all strings (with 4 or more characters) contained in the COFF object file, `file.o`.

```
$ strings file.o ↵

hello world
gcc2_compiled.
.text
.data
.tdesc
file.c
```

```
.text
.data
.tdesc
@LC0
main
@Ltb0
printf
@Lte0
@Lfe1
.rel
.symtab
.strtab
```

This command lists all strings (with 4 or more characters) contained in the ELF object file, `file.o`.

SEE ALSO

od(1).

NAME

stty – set the options for a terminal

SYNOPSIS

`stty [-a] [-g] [`*options*`]`

`stty [all|everything]`

DESCRIPTION

stty sets certain terminal I/O *options* for the device associated with the current standard input. Without arguments, or with the all argument, it reports the settings of certain options (options with non-default values and those of general interest). With the -a option, or the everything argument, it reports the settings of all line discipline options. stty obsoletes the att_stty and berk_stty commands; see the **NOTES** section below for details.

In the reports produced by stty, if a character is preceded by a caret (^), then the value of that option is the corresponding control character. (For example, "^h" is CTRL-h; in this case, recall that CTRL-h is the same as the "backspace" key.) The sequence "^-" means that an option has a null value.

Report options are:

-a	report all of the option settings.
-g	report current settings in a form that can be used as an argument to another stty command.
all	produce the same report as specifying no options, for Berkeley compatibility.
everything	produce the same report as the -a option, for Berkeley compatibility.

For detailed information about the modes listed from **Control Modes** through **Local Modes**, below, see termio(7). For detailed information about the modes listed under **Hardware Flow Control Modes** and **Clock Modes**, below, see termiox(7). Options described in the **Combination Modes** and **Berkeley Modes** sections are implemented using options in the earlier sections. Note that many combinations of options make no sense, but no sanity checking is performed. Hardware flow control and clock mode options may not be supported by all hardware interfaces.

Mode options are as follows:

Control Modes

parenb (-parenb)	enable (disable) parity generation and detection.
parext (-parext)	enable (disable) extended parity generation and detection for mark and space parity.
parodd (-parodd)	select odd (even) parity, or mark (space) parity if parext is enabled.
cs5 cs6 cs7 cs8	select character size [see termio(7)].
0	hang up line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
ispeed 0 110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal input baud rate to the number given, if possible. (Not all hardware supports split baud rates.) If the input baud rate is set to zero, the input baud rate will be specified by the value of the output baud rate.

`ospeed 0 110 300 600 1200 1800 2400 4800 9600 19200 38400`	Set terminal output baud rate to the number given, if possible. (Not all hardware supports split baud rates.) If the output baud rate is set to zero, the line will be hung up immediately.
`hupcl (-hupcl)`	hang up (do not hang up) connection on last close.
`hup (-hup)`	same as `hupcl (-hupcl)`.
`cstopb (-cstopb)`	use two (one) stop bits per character.
`cread (-cread)`	enable (disable) the receiver for input.
`clocal (-clocal)`	assume a line without (with) modem control.
`loblk (-loblk)`	block (do not block) output from a non-current layer.

Input Modes

`ignbrk (-ignbrk)`	ignore (do not ignore) break on input.
`brkint (-brkint)`	signal (do not signal) INTR on break.
`ignpar (-ignpar)`	ignore (do not ignore) parity errors.
`parmrk (-parmrk)`	mark (do not mark) parity errors [see `termio`(7)].
`inpck (-inpck)`	enable (disable) input parity checking.
`istrip (-istrip)`	strip (do not strip) input characters to seven bits.
`inlcr (-inlcr)`	map (do not map) NL to CR on input.
`igncr (-igncr)`	ignore (do not ignore) CR on input.
`icrnl (-icrnl)`	map (do not map) CR to NL on input.
`iuclc (-iuclc)`	map (do not map) upper case alphabetics to lower case on input.
`ixon (-ixon)`	enable (disable) START/STOP output control. Output is stopped by sending the STOP control character (default DC3, `^S`) and started by sending the START control character (default DC1, `^Q`).
`ixany (-ixany)`	allow any character (only DC1) to restart output.
`ixoff (-ixoff)`	request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.
`imaxbel (-imaxbel)`	echo (do not echo) BEL and refuse (flush then accept) more input when the input line gets too long.

Output Modes

`opost (-opost)`	post-process output (do not post-process output; ignore all other output modes).
`olcuc (-olcuc)`	map (do not map) lower case alphabetics to upper case on output.
`onlcr (-onlcr)`	map (do not map) NL to CR-LF on output.
`ocrnl (-ocrnl)`	map (do not map) CR to NL on output.
`onocr (-onocr)`	do not (do) output CRs at column zero.
`onlret (-onlret)`	on the terminal NL performs (does not perform) the CR function.
`ofill (-ofill)`	use fill characters (use timing) for delays.
`ofdel (-ofdel)`	fill characters are DELs (NULs).
`cr0 cr1 cr2 cr3`	select style of delay for carriage returns [see `termio`(7)].
`nl0 nl1`	select style of delay for line-feeds [see `termio`(7)].
`tab0 tab1 tab2 tab3`	select style of delay for horizontal tabs [see `termio`(7)].
`bs0 bs1`	select style of delay for backspaces [see `termio`(7)].
`ff0 ff1`	select style of delay for form-feeds [see `termio`(7)].

vt0 vt1	select style of delay for vertical tabs [see termio(7)].

Local Modes

isig (-isig)	enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (-icanon)	enable (disable) canonical input (ERASE and KILL processing).
xcase (-xcase)	canonical (unprocessed) upper/lower-case presentation.
echo (-echo)	echo back (do not echo back) every character typed.
echoe (-echoe)	echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does not keep track of control characters and, as a result, may be confusing on escaped characters and backspaces, unless the echoctl mode is also set.
echok (-echok)	echo (do not echo) NL after KILL character.
lfkc (-lfkc)	the same as echok (-echok); obsolete.
echonl (-echonl)	echo (do not echo) NL.
noflsh (-noflsh)	disable (enable) flush after INTR, QUIT, or SWTCH.
tostop (-tostop)	send (do not send) SIGTTOU when background processes write to the terminal.
echoctl (-echoctl)	echo (do not echo) control characters as ^*char*, delete as ^?.
echoprt (-echoprt)	echo (do not echo) erase character as character is "ERASEed".
echoke (-echoke)	backspace-space-backspace erase (do not erase) entire line on line kill.
flusho (-flusho)	output is (is not) being flushed.
pendin (-pendin)	retype (do not retype) pending input at next read or input character.
iexten (-iexten)	enable (disable) extended functions for input data [see termio(7)].

Hardware Flow Control Modes

rtsxoff (-rtsxoff)	enable (disable) RTS hardware flow control on input.
ctsxon (-ctsxon)	enable (disable) CTS hardware flow control on output.
dtrxoff (-dtrxoff)	enable (disable) DTR hardware flow control on input.
cdxon (-cdxon)	enable (disable) CD hardware flow control on output.
isxoff (-isxoff)	enable (disable) isochronous hardware flow control on input.

Clock Modes

xcibrg	get transmit clock from internal baud rate generator.
xctset	get transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
xcrset	get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
rcibrg	get receive clock from internal baud rate generator.
rctset	get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
rcrset	get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.

tsetcoff	transmitter signal element timing clock not provided.
tsetcrbrg	output receive baud rate generator on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
tsetctbrg	output transmit baud rate generator on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
tsetctset	output tranmitter signal element timing (DCE source) on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
tsetcrset	output receiver signal element timing (DCE source) on transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24.
rsetcoff	receiver signal element timing clock not provided.
rsetcrbrg	output receive baud rate generator on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.
rsetctbrg	output transmit baud rate generator on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.
rsetctset	output transmitter signal element timing (DCE source) on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.
rsetcrset	output receiver signal element timing (DCE source) on receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin.

Control Assignments

control-character c	set *control-character* to *c*, where *control-character* is dsusp, eof, eol, eol2, erase, flush, intr, kill, lnext, quit, rprnt, start, stop, susp, swtch, or werase. If *c* is preceded by a (escaped from the shell) caret (^) then the value used is the corresponding control character (e.g., "^d" is a CTRL-d). "^?" is interpreted as DEL and "^-" is interpreted as undefined.
min|time *number*	set the value of VMIN or VTIME, respectively,. to *number*. VMIN and VTIME are used in Non-Canonical (-icanon) mode input processing [see termio(7)].
line *i*	set line discipline to *i* (0 <= *i* <= 127).

Combination Modes

evenp or parity	enable parenb and cs7.
oddp	enable parenb, cs7, and parodd.
spacep	enable parenb, cs7, and parext.
markp	enable parenb, cs7, parodd, and parext.
-parity or -evenp	disable parenb, and set cs8.
-oddp	disable parenb and parodd, and set cs8.
-spacep	disable parenb and parext, and set cs8.
-markp	disable parenb, parodd, and parext, and set cs8.
raw (-raw or cooked)	enable (disable) raw input and output (no input editing, end-of-file detection, or signal processing, and no output post-processing).

`nl (-nl)`	unset (set) `icrnl` and `onlcr`. In addition, `-nl` unsets `inlcr`, `igncr`, `ocrnl`, and `onlret`.
`lcase (-lcase)`	set (unset) `xcase`, `iuclc`, and `olcuc`.
`LCASE (-LCASE)`	same as `lcase (-lcase)`.
`tabs (-tabs or tab3)`	preserve (expand to spaces) tabs when printing.
`ek`	reset ERASE and KILL characters back to default # and @.
`sane`	reset all modes to some reasonable values.
`tty33`	set all modes suitable for the Teletype Corp. Model 33 terminal.
`tty37`	set all modes suitable for the Teletype Corp. Model 37 terminal.
`vt05`	set all modes suitable for the Digital Equipment Corp. VT05 terminal.
`tn300`	set all modes suitable for the General Electric TermiNet 300.
`ti700`	set all modes suitable for a Texas Instruments 700 Series terminal.
`tek`	set all modes suitable for the Tektronix model 4014 terminal.
`async`	set normal asynchronous communications where clock settings are `xcibrg`, `rcibrg`, `tsetcoff` and `rsetcoff`.

Berkeley Modes

`even` allow even parity input (`-parodd`).

`-even` disallow even parity input (`parenb`, `parodd`, `cs7`, and `inpck`).

`odd` allow odd parity input (`parodd`).

`-odd` disallow odd parity input (`parenb`, `-parodd`, `cs7`, and `inpck`).

`cbreak`
make each character available to `read`(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed (`ixon`, `isig`, `-icanon`, `opost`, `vmin 1`, and `vtime 1`).

`-cbreak`
make characters available to `read`(2) only when NL is received (`ixon`, `isig`, `icanon`, `opost`, `eof ^d`, and `eol ^-`).

`tandem`
enable flow control, so that the system sends out the STOP character when its internal queue is in danger of overflowing on input, and sends the START character when it is ready to accept further input (`ixoff`).

`-tandem`
disable flow control (`-ixoff`).

`brk` *c* set break character to *c*; this character is an extra line delimiter (`veol` *c*).

`nl2 nl3`
unimplemented styles of delay for line-feed (approximated as `nl1` and `nl0`, respectively).

`dec` set all modes suitable for Digital Equipment Corp. operating systems users (`erase ^?`, `kill ^U`, `intr ^C`, `-ixany`, `echoe`, `echoke`, and `echoctl`).

`exta` set terminal baud rate to 19200 bits per second (`19200`).

`extb` set terminal baud rate to 38400 bits per second (`38400`).

`new` use line discipline number 1 (`line 1`).

`old` use line discipline number 0 (`line 0`).

`crt` set display options for a CRT (`echoe`, `echok`, and `echoctl`; `echoke`, if >= 1200 baud).

`crtbs` echo backspaces on ERASE characters (a no-op).

`prterase`
echo ERASEed characters backwards within "\" and "/" (`echoprt` and

-echoe).

`crterase`
: wipe out ERASEed characters with "backspace-space-backspace" (`echoe`).

`-crterase`
: leave ERASEed characters visible; just backspace (`-echoe`).

`crtkill`
: wipe out input on line KILL ala `crterase` (`echoke`).

`-crtkill`
: just echo KILL character (optional) and NL on line KILL (`echok` and `-echoke`).

`ctlecho`
: echo control characters as `^`*x*, DEL as `^?`; print two backspaces following the EOT character CTRL-D (`echoctl`).

`-ctlecho`
: control characters echo as themselves; in Canonical mode EOT (CTRL-D) is not echoed (`-echoctl`).

`decctlq`
: after output is suspended (normally by `^S`), only a start character (default `^Q`) will restart it (`-ixany`).

`-decctlq`
: after output is suspended, any character typed will restart it; the start character will restart output without providing any input (`ixany`).

`mdmbuf`
: start/stop output on carrier transitions (not implemented).

`-mdmbuf`
: return error if write attempted after carrier drops.

`litout`
: send output characters without any processing (`-opost`).

`-litout`
: do normal output processing, inserting delays, etc. (`opost`).

`pass8` do not strip input characters to seven bits (`-istrip`).

`-pass8`
: strip input characters to seven bits (`istrip`).

`nohang`
: don't send hangup signal if carrier drops (`clocal`).

`-nohang`
: send hangup signal to control process group when carrier drops (`-clocal`).

Window Size

`rows` *n*	set window size to *n* rows.
`columns` *n*	set window size to *n* columns.
`ypixels` *n*	set vertical window size to *n* pixels.
`xpixels` *n*	set horizontal window size to *n* pixels.

SEE ALSO

`tabs`(1), `ioctl`(2), `read`(2), `tcsetattr`(3C), `termio`(7), `termiox`(7), `ttycompat`(7).

NOTES

Beginning in Revision 5.4, the DG/UX System provides a single unified STREAMS line discipline. Prior to Revision 5.4, the DG/UX System supported two separate terminal interface conventions. The AT&T terminal interface convention was line discipline number 0, while the Berkeley (BSD) terminal interface convention was line discipline number 1. [A line discipline governs the manner in which terminal I/O is processed. For example, the editing operations (backspace, line kill, etc.) and echoing operations (no echo, echo erase, etc.) are part of the line discipline.] The merging of the two separate line disciplines results in several visible effects, which are covered in the remaining notes below.

The `att_stty` and `berk_stty` commands have been merged into a single `stty` command. This change parallels the unification of the AT&T and BSD line disciplines. Links to the original names have been retained for backwards compatibility; however, there are no longer any functional differences between the three commands.

The quantity of output from `stty` is much greater than before, and contains elements from both the `att_stty` and `berk_stty` commands' output formats. However, there is no way to obtain either of the previous output formats.

Changing the `line` discipline number no longer has any effect on the behavior of the terminal interface. For example, job control features in the C shell are available regardless of the line discipline number.

The `berk_stty` options `tilde` and `-tilde` are not supported because the merged STREAMS line discipline does not support tilde remapping.

The following combinations of Berkeley options are not supported: `even` and `odd`, and `-even` and `-odd`.

NAME

su – become super-user or another user

SYNOPSIS

su [–] [*name* [*arg* ...]]

DESCRIPTION

su lets you become another user without logging off. The default user *name* is root (i.e., superuser).

To use su, supply the appropriate password (unless it's already root). If the password is correct, su will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see passwd(4)), or /bin/sh if none is specified (see sh(1)). To restore normal user ID privileges, type an EOF (Ctrl-D) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like sh(1), an *arg* of the form -c *string* executes *string* via the shell and an arg of -r will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like sh(1):

If the first argument to su is a -, the environment is changed as if you actually logged in as the specified user. You invoke the program used as the shell with an *arg0* value whose first character is -, thus executing first the system's profile (/etc/profile) and then the specified user's profile (.profile in the new HOME directory). Otherwise, the environment is passed along with the possible exception of $PATH, which is set to /bin:/etc:/usr/bin for root.

If the optional program used as the shell is /bin/sh, the user's .profile can check *arg0* for -sh or -su to determine if it was invoked by login(1) or su(1), respectively. If the user's program is other than /bin/sh, then .profile is invoked with an *arg0* of *-program* by both login(1) and su(1).

All attempts to become another user using su are logged in the log file /usr/adm/sulog. This file contains the time and date when su was invoked, a plus sign or a minus sign indicating the success or failure (respectively) of the su command, the user's tty, the user's login name, and the name to which the user attempted to change.

For example, the following entry shows that user morris, at tty06, became root at 4:41pm on June 30.

```
SU 06/30 16:41 + tty06 morris-root
```

The following entry shows an unsuccessful attempt to become root.

```
SU 06/24 13:55 - tty11 morris-root
```

EXAMPLES

To become user bin while retaining your previously exported environment, execute:

```
su bin
```

To become user bin but change the environment to what would be expected if bin had originally logged in, execute:

```
        su - bin
```

To execute *command* with the temporary environment and permissions of user `bin`, type:

```
        su - bin -c "command args"
```

FILES

`/etc/passwd`	System's password file
`/etc/profile`	System's profile
`$HOME/.profile`	User's profile
`/usr/adm/sulog`	Log file

SEE ALSO

`env`(1), `login`(1), `sh`(1).
`passwd`(4), `profile`(4), `environ`(5) in the *Programmer's Reference for the DG/UX System*

NAME

sum – print checksum and block count of a file

SYNOPSIS

`sum [ -r ]` *file*

DESCRIPTION

`Sum` calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option `-r` causes an alternate algorithm to be used in computing the checksum.

EXAMPLES

```
sum record
```

Prints the file name and checksum for "record", and the number of 512 byte blocks of disk space that "record" occupies. If you transmit "record" to another system and then execute `sum` again, you can check that "record" was transmitted without any errors.

```
sum -r record labels
```

Calculates the checksum and the number of 512 byte blocks of disk space for the files "record" and "labels". You use the alternate algorithm because "record" and "label" were transferred to your system from a UNIX System Version 7 system. The algorithm used with the `-r` option is compatible with the algorithm used by `sum` on UNIX System Version 7 systems. This allows you to compare the checksum values both before and after the files were transmitted to check that there are no errors. `Sum` prints the information for each file on a separate line.

```
grep -l account /SYSTEMS/* | xargs sum
```

`Grep`(1) searches through the directory "SYSTEMS" for files with the string "account" in them. The command `xargs`(1) passes each file found as an argument. `Sum` prints a checksum, the number of 512 byte blocks of disk space, and the file name of each file. This command is useful for checking that many files transferred without errors from one system to another.

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

SEE ALSO

`wc`(1).

NAME

`tabs` – set tabs on a terminal

SYNOPSIS

`tabs` [*tabspec*] [`-T`*type*] [`+m`*n*]

DESCRIPTION

`Tabs` sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec Four types of tab specification are accepted for *tabspec*. They are described below: canned (*-code*), repetitive (*-n*), arbitrary (*n1,n2,...*), and file (*--file*). If no *tabspec* is given, the default value is `-8`. The lowest column number is 1. Note that for `tabs`, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are as follows:

`-a`	1,10,16,36,72 Assembler, IBM S/370, first format
`-a2`	1,10,16,40,72 Assembler, IBM S/370, second format
`-c`	1,8,12,16,20,55 COBOL, normal format
`-c2`	1,6,10,14,49 COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see `fspec`(4)): `<:t-c2 m6 s66 d:>`
`-c3`	1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67 COBOL compact format (columns 1-6 omitted), with more tabs than `-c2`. This is the recommended format for COBOL. The appropriate format specification is (see `fspec`(4)): `<:t-c3 m6 s66 d:>`
`-f`	1,7,11,15,19,23 FORTRAN
`-p`	1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61 PL/I
`-s`	1,10,55 SNOBOL
`-u`	1,12,20,44 UNIVAC 1100 Assembler

-n A *repetitive* specification requests tabs at columns 1+*n*, 1+2**n*, etc. Of particular importance is the value `8`: this represents the DG/UX system usual tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value `0`, implying no tabs at all.

n1,n2,...
The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an

increment to be added to the previous value. Thus, the formats `1,10,20,30`, and `1,10,+10,+10` are considered identical.

--file If the name of a *file* is given, `tabs` reads the first line of the file, searching for a format specification (see `fspec`(4)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as `-8`. This type of specification may be used to make sure that a file with tabs is printed with correct tab settings, and would be used with the `pr`(1) command:

```
tabs --file; pr file
```

Any of the following also may be used. If a given flag occurs more than once, the last value given takes effect:

`-T`*type* `tabs` usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in `term`(5). If no `-T` flag is supplied, `tabs` uses the value of the environment variable `TERM`. If `TERM` is not defined in the environment (see `environ`(5)), `tabs` tries a sequence that usually works for different terminals.

`+m`*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If `+m` is given without a value of *n*, the value assumed is `10`. For a TermiNet, the first value in the tab list should be `1`, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by `+m0`. The margin for most terminals is reset only when the `+m` flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

`tabs -a` example using *-code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

`tabs -8` example of using *-n* (*repetitive* specification), where *n* is `8`, causes tabs to be set every eighth position:
1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

`tabs 1,8,36`
example of using *n1,n2,...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

`tabs --$HOME/fspec.list/dgux18`
example of using *--file* (*file* specification) to indicate that tabs should be set according to the first line of `$HOME/fspec.list/dgux18`" (see `fspec`(4)).

DIAGNOSTICS

`illegal tabs`
when arbitrary tabs are ordered incorrectly

`illegal increment`
when a zero or missing increment is found in an arbitrary specification

`unknown tab code`
when a *canned* code cannot be found

`can't open`
if *--file* option used, and file can't be opened

`file indirection`
if *--file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted

`unable to set tabs`
: $TERM is not a valid terminal type or its terminfo entry does not contain operations for setting tabs.

SEE ALSO

`newform`(1), `pr`(1), `tput`(1).
`fspec`(4), `terminfo`(4), `environ`(5), `term`(5) in the *Programmer's Reference for the DG/UX System*.

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

`Tabs` clears only 20 tabs (on terminals requiring a long sequence), but can set 64.

CAUTION

Data General terminals do not support hardware tabbing. The *tabspec* used with the `tabs` command is different from the one used with the `newform`(1) command. For example, `tabs -8` sets every eighth position; whereas `newform -i-8` indicates that tabs are set every eighth position.

NAME

taccess – initiate access to labeled tape

SYNOPSIS

taccess [-a *ADN*] [-d *density*] [-f *length*] [-l IBM|ANSI|UN] [-p *pseudo*] [-z] -v *vsnlist*

DESCRIPTION

Taccess must be used before any other REELexchange commands can be used. It initializes access to the specific tapeset and tape drive. Through it, the user identifies the tapeset and its characteristics: density, length, tape format. Also, the user specifies the Volume Serial Number(s) (VSNs) for the tape volume(s) in the tapeset. If a multi-volume tapeset is being processed, then the full list of VSNs must be named.

OPTIONS

-a *ADN*
: "Tape Drive." Specifies which tape drive to use. The name specified should be one of the ADN files from the /var/reelexchange/Adn directory. If this option is not specified, the tape drive named in the file /var/reelexchange/default.adn will be used. See reelexchange_intro(1) for more information on the format of ADN files.

-d *density*
: "Density." The density in *bpi* of the tape. Common values are 1600, 3200, and 6250. The density specified here should match the density field in the ADN file of the tape drive that is being accessed. Default density is 1600.

-f *length*
: "Length." The length of the tape in feet less 100 feet error margin. Common values are 700, 1100 and 2300. Default length is 1400.

-l "Format." Specifies the format of the tape – either IBM, ANSI, or UN for unlabeled tapes. Default format is ANSI.

-p *pseudo*
: "Tapesetname." Defines an arbitrary alphanumeric name by which the tapeset is referenced during the current session. By default,REELexchange calls each tape default.

-v *vsnlist*
: "Tapeset." *vsnlist* is a comma separated list of VSNs. This list should be the order of tapes in the tapeset if the current tape being introduced by taccess is part of a multivolume tapeset.

-z "No Prompt." Normally, after a session has been established via the taccess command, the first command of the session that requires physical access to the tape will display a prompt which instructs you to mount the tape. If -z is specified, this prompt will be suppressed.

FILES

/var/reelexchange/default.adn	File containing name of default tape drive.
/var/reelexchange/tapecap	File containing descriptive entries for all tape drives supported by REELexchange.
/var/reelexchange/Adn/*	Each file in this directory describes a tape drive on the system. The name of one of these files is given with the -a option to specify a particular tape drive

	for I/O.
`/var/reelexchange/d_ibm`	IBM label defaults.
`/var/reelexchange/d_ansi`	ANSI label defaults.

SEE ALSO

`reelexchange_intro`(1), `tdisplay`(1), `tkey`(1), `tlabel`(1), `tposn`(1), `tread`(1), `trelease`(1), `tsniff`(1), `twrite`(1).

NAME

`tail` – deliver the last part of a file

SYNOPSIS

`tail` [± *number* `lbcr`] [*file*]
`tail` [`-lbcr`] [*file*]
`tail` [± *number* `lbcf`] [*file*]
`tail` [`-lbcf`] [*file*]

DESCRIPTION

`Tail` copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *–number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option `l`, `b`, or `c`. When no units are specified, counting is by lines.

With the `-f` (follow) option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied. Instead, it enters an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus, you can use `tail` to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f fred
```

prints the last ten lines of the file `fred`, followed by any lines that are appended to `fred` between the time `tail` is initiated and killed. As another example, the command:

```
tail -15cf fred
```

will print the last 15 characters of the file `fred`, followed by any lines that are appended to `fred` between the time `tail` is initiated and killed.

The `r` option copies lines from the specified starting point in the file in reverse order. The default for `r` is to print the entire file in reverse order.

The `r` and `f` options are mutually exclusive.

International Features

`tail` can process files containing characters from supplementary code sets.

SEE ALSO

`cat`(1), `dd`(1), `head`(1), `more`(1), `pg`(1), `tail`(1).

NOTES

Tails relative to the end of the file are saved in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

The `tail` command will only tail the last 4096 bytes of a file regardless of its length.

NAME

tar – tape file archiver

SYNOPSIS

tar -c[vwfbLkFhienA[*#s*]] *device block tapesize incfile* [*file* . . .]
tar -c[vwfbLkXhienA[*#s*]] *device block tapesize excfile* [[-I *incfile* | *file*] . . .]
tar -r[vwfbLkFhienA[*#s*]] *device block tapesize incfile* [*file* . . .]
tar -r[vwfbLkXhienA[*#s*]] *device block tapesize excfile* [[-I *incfile* | *file*] . . .]
tar -t[vfLFien[*#s*] *device incfile* [*file* . . .]
tar -t[vfLXien[*#s*] *device excfile* [[-I *incfile* | *file*] . . .]
tar -u[vwfbLkFhienA[*#s*]] *device block tapesize incfile* [*file* . . .]
tar -u[vwfbLkXhienA[*#s*]] *device block tapesize excfile* [[-I *incfile* | *file*] . . .]
tar -x[lmovwfLFpienA[*#s*]] *device incfile* [*file* . . .]
tar -x[lmovwfLXpienA[*#s*]] *device excfile* [[-I *incfile* | *file*] . . .]

DESCRIPTION

tar saves and restores files on magnetic tape. Its actions are controlled by a string of characters containing one function letter (c, r, t, u, or x), and possibly followed by one or more function modifiers (v, w, f, b, L, k, F, X, h, i, e, n, A, l, m, o, p, and *#s*). Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. A file name which follows a -I is interpreted as an include file whose contents is a list files or directories to be included in the file list. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function letters are as followings:

-c Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This function letter implies the r function letter.

-r Replace. The named *files* are written on the end of the tape. The c and u function letters imply this function letter.

-t Table. The names and other information for the specified files are listed each time they occur on the tape. The listing is similar to the format produced by the ls -l command [see ls(1)]. If no *files* argument is given, all the names on the tape are listed.

-u Update. The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape. This function letter implies the r function letter.

-x Extract. The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. Use the file or directory's relative path when appropriate, or tar will not find a match. If tar is invoked as superuser, the ownership and permissions for the file are restored. If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

The characters below may be used in addition to the letter that selects the desired function. Use them in the order shown in the synopsis.

[0-9] Digit modifiers determine the drive on which the tape is mounted. The digit modifier tells tar to use a drive other than the default drive. Drive numbers are mapped to actual devices by entries in the file /etc/default/tar. In addition, entries in /etc/default/tar specify a default record size and default media capacity for each drive number. The format for each entry is

```
archiven=device-file     blocking-factor     capacity
```

where *n* is the drive number, *device-file* is the full pathname for the device (e.g. /dev/rmt/0), *blocking-factor* is the default record size in blocks, and *capacity* is the media capacity in kilobytes. If neither a digit function modifier nor the `f` function modifier is specified, drive 0 is assumed by default.

`v` Verbose. Normally, `tar` does its work silently. The `v` (verbose) function modifier causes `tar` to echo to the screen, the name of each file it treats, preceded by the function letter. With the `t` function letter, `v` gives more information about the tape entries than just the name.

`w` What. This causes `tar` to print the action to be taken, followed by the name of the file, and then wait for your confirmation. If a word beginning with `y` is given, the action is performed. Any other input means no. This is not valid with the `t` function letter.

`f` File. This causes `tar` to use the *device* argument as the name of the archive instead of `/dev/rmt/0`. If the name of the file is `-`, `tar` writes to the standard output or reads from the standard input, whichever is appropriate. Thus, `tar` can be used as the head or tail of a pipeline. `tar` can also be used to move hierarchies with the command:

```
cd fromdir; tar cf - . | (cd todir; tar xf -)
```

`b` Blocking Factor. This causes `tar` to use the *block* argument as the blocking factor for tape records rather than the default. The default blocking factor is determined as follows: If a drive number is specified via function modifiers `0` through `9`, the default is taken from the `/etc/default/tar` entry for that drive number. If an output device is specified via the `f` function modifier, the default is 32. If neither a drive number nor the `f` function modifier is specified, drive number 0 is assumed and the blocking factor is obtained from the entry for drive 0 in the `/etc/default/tar` file. The maximum blocking factor is 32. This function letter should not be supplied when operating on regular archives or block special devices. The block size is determined automatically when reading tapes created on block special devices (function letters `x` and `t`).

`l` Link. This tells `tar` to complain if it cannot resolve all of the links to the files being dumped. If `l` is not specified, no error messages are printed.

`m` Modify. This tells `tar` to not restore the modification times. The modification time of the file will be the time of extraction.

`o` Ownership. This causes extracted files to take on the user and group identifier of the user running the program, rather than those on tape. This is only valid with the `x` function letter.

`L` Follow symbolic links. This causes symbolic links to be followed. By default, symbolic links are not followed. This function is identical to that of the `h` function modifier.

`k` This function modifier uses the *tapesize* argument as the size in bytes per volume for non-tape devices (such as a floppy drive). If this argument is not given, or if *tapesize* is 0, multi-volume mode is disabled (the volume is assumed to be of infinite length). This option allows you to create tar archives which span more than one tape. For example, to create a tar image on 150 Mbyte tapes, use a command line similar to

```
tar -cvfk /dev/rmt/0 145000 dir
```

This function modifier may be used with the `c`, `r`, and `u` function letters.

`F` This function modifier uses the *incfile* argument as a file containing a list of named files (or directories) to be included on the tape. This modifier can often be very helpful for including a list of files generated by the `find`(1) command. For example, if you wanted to archive the files under the directory *mydir* to `/dev/rmt/0` but wanted to exclude its subdirectories, you could do so with the following commands:

```
find mydir \! -type d -print >/tmp/list

tar cF /tmp/list
```

This function modifier may not be used with the `X` function modifier; however, the `-I` option, which performs the same function, is permitted with the `X` modifier.

`X` This function modifier uses the *excfile* argument as a file containing a list of named files (or directories) to be excluded, and may be used with all function letters. This function modifier may not be used with the `F` function modifier.

`h` This function modifier causes `tar` to follow symbolic links as if they were normal files or directories. Normally `tar` does not follow symbolic links. The `h` function modifier may be used with the function letters `c`, `r`, and `u`.

`p` This function modifier restores the named *file* arguments to their original modes, ignoring the present value returned by `umask` [see `umask`(2) in the *Programmer's Reference Manual*]. This option is not necessary if the the effective user ID is root. For users with root privileges, the `umask` is always ignored and file modes are restored exactly as they were recorded in the archive. This function modifier may only be used with the `x` function letter.

`i` This function modifier causes `tar` to ignore directory checksum errors.

`e` This function modifier causes `tar` to quit when certain minor errors are encountered. examples of "minor errors" are: a file name that is too long, and a file that changes size while it is being backed up. Without this modifier, `tar` will continue when errors such as these are encountered.

`n` This function modifier must be used when the *device* argument is for a non-tape device (for example, a floppy drive).

`A` This function modifier causes absolute pathnames for files to be suppressed, and may be used with the `r`, `c`, `u`, and `x` function letters. This causes all pathnames to be interpreted as relative to the current working directory.

The `-I` option is similar in function to the `F` function modifier. The argument following `-I` is assumed to be an include file containing a list of named files (or directories) to be included in the archive. Unlike the `F` function modifier, `-I` may be used with the `X` function modifier; it may also be specified more than once on the command line. The `-I` option and the `F` function modifier may not specified together.

FILES

```
/etc/default/tar
/dev/rmt/*
/tmp/tar*
```

DIAGNOSTICS

Complains about bad key characters and tape read/write errors.

Complains if not enough memory is available to hold the link tables.

SEE ALSO

`ar`(1), `cpio`(1), `ls`(1), `umask`(2), `tar`(5).

NOTES

`Tar` maintains the block allocation limit of control point directories.

The `r` and `u` options are not supported when writing to tape drives; they can be used only when writing tar images to disk.

`Tar` does not maintain the file node allocation limit of control point directories due to limitations in the `tar`(4) header format.

The limit on path name length is 255 characters (155 characters for the directory portion and 100 characters for the file name portion).

The block and seek sizes reported when verbose mode (function modifier `v`) is specified are rounded to the nearest kilobyte.

NAME

tdisplay – display label and record translation settings

SYNOPSIS

tdisplay [-a | -d | -r | -t | -w] [-p *pseudo*]

DESCRIPTION

tdisplay reports the current settings of file labels and record translation parameters. The tkey command is used to modify label and translation settings.

The -r option reports the file labels found after a tposn -r or a tread command has been issued.

The -w option reports the file labels which would be written by a subsequent twrite command. When a tposn command is executed, these labels always revert to the default values. The tkey command allows the labels to be modified before writing.

The -d option displays the default file labels used during file writes. They can be changed with the tkey -d command.

Current record translation settings are reported via the -t option:

```
Translation processing:

Key       Value
---       -----
pchar     40(octal)
tchar     12(octal)
conv      none
```

Options

-a "All." Display everything.

-d "Default." Display default write labels.

-p *pseudo*
"Tapesetname." Tapeset pseudo name as specified by the preceding taccess command which initiated access to the tapeset.

-r "Read." Display current read labels.

-t "Translation." Display current record translation settings.

-w "Write." Display current write labels.

FILES

/var/reelexchange/d_ibm	IBM label defaults.
/var/reelexchange/d_ansi	ANSI label defaults.

SEE ALSO

reelexchange_intro(1), taccess(1), tkey(1), tlabel(1), tposn(1), tread(1), trelease(1), tsniff(1), twrite(1).

NAME

`tee` – pipe fitting

SYNOPSIS

`tee` [`-i`] [`-a`] [*file* ...]

where:

file	An output file

DESCRIPTION

`Tee` transcribes the standard input to the standard output and makes copies in the specified files. If you specify the names of already-existing files, they will be overwritten.

`Tee` is used mainly as part of a pipeline. If you execute it on a command line by itself, it will take input from the standard input until it receives a control D (Ctrl-D). It will echo each line to the standard output.

Options

`-i`	Ignore interrupts.
`-a`	Append output to the specified files rather than overwrite them.

EXAMPLES

```
$ ls | tee file.list
```

This command sends a list of all files in the current directory to the standard output (the screen) and makes a copy in `file.list`.

```
$ pr file | tee file.out | lp
```

This command sends a print-formatted version of `file` through a pipe to `lp`, and also writes it to the file `file.out`.

SEE ALSO

`csh`(1), `ksh`(1), `sh`(1).

NAME

`telnet` – log in to another host over network

SYNOPSIS

`telnet [-i] [-d] [-o] [-bi] [-bo] [-e] [-s] [-l] [host [port] ]`

DESCRIPTION

The `telnet` command logs in to another host using the TELNET protocol. If you invoke `telnet` without arguments, it enters command mode, indicated by its prompt (`telnet>`). In this mode, it accepts and executes the commands listed below. When you invoke `telnet` with arguments, it first checks to see if the arguments match the switches. TELNET then checks to see if the `host` and `port` number are legal. If the arguments are legal, TELNET performs an `open` command (see below) with those arguments. If the remote host requests TERM or window size information, TELNET supplies it.

After a connection has been opened, `telnet` enters input mode. The text you type is sent directly to the remote host. The `telnet` command provides an eight-bit data path to the network if the local stty settings provide one. You can toggle the scanning of some special characters for NVT translation using the `localchars` command (see *Using TCP/IP on the DG/UX System* for more details on character translation). You cannot invoke TELNET commands in input mode; however, you can use the escape character (initially ^]) to enter local mode. While in local mode, you can invoke a TELNET command. After TELNET executes the command, it returns you to input mode.

Commands

The following commands are available. You only need to type the first three letters of any command to uniquely identify it.

`bye`	Close any open TELNET session and exit `telnet`. (Go back to the shell, or to the program that called `telnet`.) This is the same as the `quit` command.
`close`	Close a TELNET connection and return to where the connection was opened from (for example, the shell, or TELNET command mode).
`crmod`	Toggle carriage return mode. When enabled, this mode changes the current setting for `CR` in `stty`(1). If carriage return characters received from the remote host are mapped to `CR`; for example, `crmod` changes the setting to `NL`, where `NL` is a line feed and a carriage return. If the current setting in `stty` is `NL`, `crmod` changes the setting to `CR`.
`debug`	Toggle debug mode. Also, toggle the ability to use certain commands. With debug on, the following commands are enabled: `listen`, `map`, `send`, and `mode`. These commands let you wait for an incoming connection, change character mappings, and change modes without negotiating options.
CAUTION:	Beware when using the `debug` command. TELNET commands can be unpredictable when used in debug mode.
`escape` [*string*]	Set the TELNET escape character. Use the same syntax as for strings in the C programming language (for example, use double quotes). If the string is not specified, `telnet` will prompt for it.

`help` [*command*]	Get help. With no arguments, `help` prints a help summary. If a command is specified, `help` print the help information available about the command only. The `?` command is identical to the `help` command.
`listen` [*port*]	Listen on a port for an incoming connection. This command works only in debug mode. Use the escape character to abort the connection, or the Interrupt key (often Ctrl-C) if the connection has not yet been made.
`localchars`	Toggle the local translation of specific character sequences into TELNET NVT keyboard sequences. Translation is off by default. In the default mode, a special key such as the `stty` interrupt key (often set to Ctrl-C) is passed unmodified to the remote host for possible processing. For remote UNIX systems, this is usually preferable. If `localchars` is toggled on, the Ctrl-C is intercepted locally and translated into the standard TELNET sequence for Interrupt Process (IP). The remote host then decodes this sequence and performs whatever actions are required to interrupt the process. The `status` command shows what translations are currently being done.
`log` [*logfile*]	Instruct TELNET to put the data that the remote host sends to your terminal to a *logfile*. Logging starts when you enter `log` with the name of a *logfile*. Logging stops when you enter `log` without an argument. If the *logfile* does not exist, it will be created. If it does, TELNET appends data to the end of the file. Due to buffering, data may not be fully written to the logfile until logging is stopped.
`map` *string NVT_char*	Substitute a string of your choice for an NVT character. You can use this command only in debug mode. NVT characters are: IP, AO, AYT, EC, EL, BRK, and EOR (see **Definitions** below for descriptions of these characters).
`mode` *option type*	Change mode regardless of negotiated option. You can use this command only in debug mode. *Option* can be one of the following: EC, BI, BO, or LI (see **Definitions** below for descriptions of these options). *Type* can be one of the following: `on`, `off`, `always`, `never` (see **Definitions** below for descriptions of these types).
CAUTION:	When using the option types `never` and `always`, you could ask for a particular option that the remote server does not want. In such a case, you can expect unusual results during the connection.
`negotiate` *option type*	Request negotiation on an option. You can only request a negotiation; you cannot send an announcement of the current mode. TELNET does not notify you that the change has been made or not. Use the command `status` to see the results. *Option* can be one of the following: SGA, EC, BI, BO, ST, TM, or EX (see **Definitions** below for descriptions of these options). *Type* can be one of the following: `on`, `off`,

	always, never (see **Definitions** below for descriptions of these types).
open *host* [*port*]	Open a connection to the named host. If no port number is specified, telnet will attempt to contact a TELNET server at the default port. The host specification may be either a hostname (see hosts(4)) or an Internet address specified in the dot notation (see inet(3N)).
options	Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations will be displayed. Options sent by telnet are displayed as SENT, while options received from the TELNET server are displayed as RCVD.
prompt *string*	Substitute a string of your choice in place of the normal command prompt.
quit	Close any open TELNET session and exit telnet. (Go back to the shell, or to the program that called telnet.)
resume	Exit local mode and continue any suspended remote mode. Returning to remote mode does not automatically produce a shell prompt, refresh the screen, or enter any characters. You may take these actions yourself.
send *NVT_char*	Send NVT special characters across your network connection. (You can use this command only in debug mode.) You can substitute any of the following for *NVT_char*: Sync, IP, AO, AYT, EC, EL, BRK, or EOR (see **Definitions** below for descriptions of these characters).
shell [*command*]	Create a shell process without terminating TELNET. If you have a network connection, it will remain suspended until you terminate the shell process. Terminate the shell process by entering the exit command. The ! command is identical to the shell command.
status	Show the current status of telnet parameters. This includes the host to which you are connected, the local character translations, and the state of debugging.
terminator *string*	Add *string* to the list of terminators. Terminators determine when to ship characters in line mode. The list of terminators includes the default characters (see below) and any you specify. They cannot be the escape character or NVT special characters. The default terminators are: New Line End-of-file character Each character in the string you specify will be a terminator.
un-term *string*	Cancel terminator status for specified characters.
z	Suspend telnet. This command works only when the user is using DG/UX csh(1); it interacts with the C shell's job control facilities. When you issue a suspend command, you are then placed in the C shell. To return to the

	`telnet` session, type `fg job#`. `Job#` is the job number that was returned when you suspended the TELNET session. See `csh(1)` for more information on how the C shell handles job control.
`?` [*command*]	Get help. With no arguments, `?` prints a help summary. If a command is specified, `?` print the help information available about the command. This is a synonym for the `help` command.
`!` [*command*]	Create a shell process without terminating TELNET. If you have a network connection, it will remain suspended until you terminate the shell process. Terminate the shell process by entering the `exit` command. This command is identical to the `shell` command.

Switches

Some of the commands shown above are available as *switches*. Switches let you issue commands when you execute `telnet`, without having to enter command mode first. The following table shows the available switches, their corresponding commands, and definitions:

Switch	**Command**	**Definition**
`-i`*s_type*	Negotiate	Negotiate binary input option.
`-o`*s_type*	Negotiate	Negotiate binary output option.
`-d`	Debug mode	Turn debug mode on.
`-e`*s_type*	Negotiate	Negotiate remote echo option.
`-s`*s_type*	Negotiate	Negotiate remote side suppress-go-ahead option.
`-l`[*port*]	Listen	Listen for connections on the given port number.
`-bi, -bo`		These mode switches send and receive the data as is, with no translation from either side. All control characters are received and not ignored.

The argument *s_type* indicates whether or not you want the option. You must substitute either the letter `a`, for always, or the letter `n`, for never.

For the argument [*port*], you must substitute the port number of the connection you are monitoring. If you do not specify a port number, `telnet` will assign one to you.

Definitions

This section describes the NVT characters, options, and types that are used with the commands `map`, `mode`, `negotiate`, and `send`. The NVT characters are as follows:

NVT Character	Meaning
Sync	A TCP urgent notification with the command data mark (DM).
Interrupt process (IP)	Suspends, interrupts, aborts, or terminates a user process.
Break character (BRK)	Sends the appropriate break character to the remote process.
Abort output (AO)	Allows a process to run to completion, but does not send the output to the user's terminal.
Are you there (AYT)	Provides the user with visible evidence that the system is still up and running.
Erase character (EC)	Deletes the last preceding character or *printed position* the user types. Printed position means several characters that are a result of overstrikes.
Erase line (EL)	Deletes all the data on the current line of input.
End of record (EOR)	Allows the user to flush the input buffer before a terminator character is encountered.

The strings used for *option* in the commands `mode` and `negotiate` are as follows:

Option	Meaning
SGA	Suppress go-aheads
EC	Foreign echoing (for `negotiate`)
EC	Local echoing (for `mode`)
BI	Binary input
BO	Binary output
ST	Status
TM	Timing mark
EX	Extended option
LI	Line mode

When using the command `mode`, substitute one of the following strings for *type*:

Type	**Function**
`on`	Turns on the mode regardless of the option that normally controls it. However, if the option is negotiated after your change, the mode changes to correspond with the change in the option.
`off`	Turns off the mode regardless of the option that normally controls it. However, if the option is negotiated after your change, the mode changes to correspond with the change in the option.
`always`	Turns the mode on and leaves it on regardless of the option. Mode is changed when you invoke the change or when you close the connection. When you close the connection, the mode returns to the default setting.
`never`	Turns the mode off and leaves it off regardless of the option. Mode changes when you invoke the change or when you close the connection. When you close the connection, the mode returns to the default setting.

When using the command `negotiate`, substitute one of the following strings for *type*:

String	**Function**
`on`	Try to negotiate an option `on`. All future requests to negotiate the option `off` will be honored.
`off`	Try to negotiate an option `off`. All future requests to negotiate the option `on` will be honored.
`always`	Try to negotiate an option `on`. All future requests from the server program to negotiate the option `off` will be honored. However, the user program immediately sends one request to the server to negotiate the option back `on`.
`never`	Demands an option to be negotiated `off` and left `off`. All future requests to negotiate it `on` will be refused. You must know whether or not the server will abort when that option is refused.

SEE ALSO

`rlogin`(1), `telnetd`(1M), `inet`(3N), `hosts`(4).

NAME

`termprinter` – print a file using the 40014A Terminal Server

SYNOPSIS

`/var/spool/lp/model/termprinter`

DESCRIPTION

This script takes the file to be printed from standard input. This script combines print header information with the file to be printed and then sends the file to a printer connected to a DG model 40014A Terminal Server through the use of the `/usr/lib/lptermprinter` utility. This routine does not perform any conversion except for tab expansion before sending it to the print routine.

This routine will store any output from the `/usr/lib/lptermprinter` utility in a file called:

`/tmp/lptermprinter-`*printer-name-job-number*

where *printer-name* and *job-number* are the same as those returned by the `lp` program when the print job is queued. This file is not deleted when `lptermprinter` returns with a status value greater than 0.

This routine takes the name of the printer, which will be the name of the script after it has been copied by the `sysadm` utility, and uses this as the host name for the `lptermprinter` utility. For this reason, the printer should have been set up as a host using the `sysadm` utility under the same name assigned to the printer.

Refer to the "40014A Terminal Server / AViiON Release Notice" for further information.

FILES

`/var/spool/lp/model/termprinter`
`/usr/lib/lptermprinter`

SEE ALSO

`lptermprinter`(1), `sysadm`(1M).

NAME

test – condition evaluation command

SYNOPSIS

test *expr*
[*expr*]

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; test also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

-r *file*	True if *file* exists and is readable.
-w *file*	True if *file* exists and is writable.
-x *file*	True if *file* exists and is executable.
-f *file*	True if *file* exists and is a regular file.
-d *file*	True if *file* exists and is a directory.
-% *file*	True if *file* exists and is a control point directory.
-c *file*	True if *file* exists and is a character-special file.
-b *file*	True if *file* exists and is a block-special file.
-h *file*	True if *file* exists and is a symbolic-link.
-p *file*	True if *file* exists and is a named pipe (fifo).
-u *file*	True if *file* exists and its set-user-ID bit is set.
-g *file*	True if *file* exists and its set-group-ID bit is set.
-k *file*	True if *file* exists and its sticky bit is set.
-s *file*	True if *file* exists and has a size greater than zero.
-t *[fildes*]	True if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
-z *s1*	True if the length of string *s1* is zero.
-n *s1*	True if the length of the string *s1* is non-zero.
s1 = *s2*	True if strings *s1* and *s2* are identical.
s1 != *s2*	True if strings *s1* and *s2* are *not* identical.
s1	True if *s1* is *not* the null string.
n1 -eq *n2*	True if the integers *n1* and *n2* are algebraically equal. Any of the comparisons -ne, -gt, -ge, -lt, and -le may be used in place of -eq.

You can combine these primaries with the following operators:

!	Unary negation operator.
-a	Binary and operator.
-o	Binary or operator (-a has higher precedence than -o).
(*expr*)	Parentheses for grouping.

All the operators and flags are separate arguments to test. Parentheses are meaningful to the shell and, therefore, must be escaped.

International Features

test can process characters from supplementary code sets in expr.

EXAMPLES

```
$ test -r exercise.1
```

Tests the expression -r exercise.1. Checks if the file exercise.1 exists and is readable. If it exists and is readable, then test sets the exit status to zero.

```
$ [ "$a" = "yes" ]
```

Checks if the value for the variable a is the string "yes". If it is, test sets the exit status to zero. Note that the square brackets must be delimited by blanks for this form of test to work.

```
$ if
      test -s exercise.1
  then
      pr exercise.1
  fi
```

This is an example of using test in a shell program. The test in this shell program tests the expression -s exercise.1, which checks if exercise.1 is a file with at least one character. If exercise.1 is a file and has at least one character, the program prints the file. Otherwise, the program does not print exercise.1.

SEE ALSO

find(1), sh(1).

CAUTION

In the second form of the command (i.e., the one that uses [ ], rather than the word test), the square brackets must be delimited by blanks.

NAME

`tftp` – DARPA trivial file transfer protocol

SYNOPSIS

`tftp` [*host* [*port*]]

where:

host	A hostname or a dot-format Internet address
port	A small integer identifying a port

DESCRIPTION

The `tftp` program is the user interface to a very simple network file transfer protocol. The program lets a user transfer files to and from a remote network site.

This is a much simpler program than `ftp(1C)`. It does not let you display a remote directory, invoke a shell, or do other kinds of file and directory manipulation. Generally, you would use it only when communicating with a remote host that does not support `ftp`.

If *port* is not specified (the usual case), a default port is assumed. When invoked, `tftp` displays a prompt, `tftp>`. You may then issue any of the following commands:

`?` Help. Displays the `tftp` command list.

`connect` [*host* [*port*]]
Identifies a remote host with which to communicate. *Host* is a hostname or a dot-format Internet address. `Port` is an integer.

`quit` Terminates `tftp`.

`mode` [*name*] Sets the file transfer mode. File transfers (`get`, `put`) made after the mode is set will be performed in that mode. *Name* is one of the following:

`ascii` transfers standard ASCII text files.
`binary` transfers binary files, such as compiled programs.
`mail` sends files as mail to a user rather than to a file (this mode is not implemented).

`get` Transfers a file from the remote host to a local file or directory. If `connect` has not been issued for the remote host desired, then *host*: must be specified explicitly. It is used much like `cp`(1).

`get`, executed without an argument, will prompt for both filenames.

`get` [*host*:]*/rdirpath/file* copies remote file *file* from the remote *host* into a file named *file* in the current local directory. It will overwrite an existing file of the same name.

`get` [*host*:]*/rdirpath/file1* [*/ldirpath/*]*file2* copies remote file *file1* into local file *file2*.

`get` [*host*:]*/rdirpath/file* ... [*/ldirpath/*]*dir* copies the remote file *file* into the specified local directory. More than one remote file can be specified; `tftp` assumes that the last name given is the appropriately specified local directory that you want the remote files moved to.

All *rdirpath*s must be absolute pathnames, i.e., specified all the way from the root directory `/` on the remote host. *Ldirpath*s may be absolute or relative pathnames (if not specified, the current local directory is assumed). All *host*s are specified by hostnames or dot-format Internet addresses.

`put`
: Transfers a file from the local host to the remote host. If `connect` has not been issued for the remote host desired, then *host*: must be specified explicitly. Otherwise, it is used much like `cp`(1).

`put`, executed without an argument, prompts for filenames.

`put` [*host*:]*/rdirpath/file* copies local file *file* to the remote host into a file of the same name in the specified remote directory. It will overwrite an existing file of the same name.

`put` [*/ldirpath/*]*file1* [*host*:]*/rdirpath/file2* copies local file *file1* into remote file *file2*.

`put` [*/ldirpath/*]*file*... [*host*:]*/rdirpath* copies local file *file* into the specified remote directory. More than one local file can be specified; `tftp` assumes that the last name given is an appropriately specified remote directory.

All *rdirpath*s must be absolute pathnames, i.e., specified all the way from the root directory `/`on the remote host. *Ldirpath*s may be absolute or relative pathnames (if not specified, the local current working directory is assumed). *Host*s are specified by hostnames or by dot-format Internet addresses.

`rexmt`
: Sets the amount of time in seconds to wait before a retry is sent. Default is 5 seconds. You might want to increase the amount of time if the network is very slow.

`status`
: Displays the settings for `host`, `mode`, `trace`, `verbose`, `rexmt`, and `timeout`.

`timeout`
: Sets the amount of time in seconds to wait before giving up on a file transfer. Default is 25 seconds; you might want to increase this number if the network is very slow.

`trace`
: Turns on trace mode. When on, packet transfers are displayed on the screen. During `put` and `get`, each packet transfer prints the packet header on the screen. The packet header contains information such as type of packet and the packet number. Retries show up as multiple lines with the same packet number.

`verbose`
: Turns on verbose mode. Verbose mode has no meaning in the DG/UX System; nonetheless, it is visible to `status`.

This protocol trades flexibility for absolute simplicity. It uses a reliable, lockstep packet mechanism. Security depends on file permissions and how much outsiders know about your directories and files. Remote users are governed by the "others," or `o` section of a file permission.

DIAGNOSTICS

Most errors terminate the command, including:

* file not found,
* user not found,
* access violation (you don't have access to a directory to which you tried to send a file),

* internal errors (a server receives a badly formed packet).

SEE ALSO

chmod(1), chown(1), cp(1), ftp(1C), ftpd(1M), tftpd(1M), inet(3N), hosts(4).

NAME

time – time a command

SYNOPSIS

time *command*

DESCRIPTION

The *command* is executed; after it is complete, time prints the elapsed time during the command, the time spent in the system, and the time spent executing the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

csh(1), timex(1).
times(2) in the *Programmer's Reference for the DG/UX System (Volume 1)*

NAME

`timex` – time a command; report process data and system activity

SYNOPSIS

`timex` [*options*] *command*

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of `timex` is written on standard error.

Options are:

`-p` List process accounting records for *command* and all its children. Suboptions `f, h, k, m, r,` and `t` modify the data items reported, as defined in `acctcom`(1). The suboptions are:

- `-f` Print the *fork/exec* flag and system exit status columns in the output.
- `-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).
- `-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. The number of blocks read or written and the number of characters transferred are always reported.

`-o` Report the total number of blocks read or written and total characters transferred by *command* and all its children.

`-s` Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in `sar`(1) are reported.

EXAMPLES

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex -opskmt sh
        session commands
EOT
```

SEE ALSO

`acctcom`(1), `sar`(1), `time`(1).

CAUTION

Process records associated with *command* are selected from the accounting file `/usr/adm/pacct` by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

NAME

tkey – set label and data translation parameters

SYNOPSIS

tkey [-d] [-f *filename*] [-p *pseudo*] [*key* = *val* ...]

DESCRIPTION

Tkey modifies file labels and record translation parameters. If write label parameters are modified, they will be remain in effect only until the next tposn or twrite command (tposn and twrite reset write label parameters to their default values). If you would like the write label values to remain in effect for multiple twrite operations, use the -d option.

Options

-d	"Default." Resets the default write labels according to the keyword assignments provided. If this option is specified, changes to the write labels will remain effective for any succeeding commands in the access session. If -d is not specified, the changes to the write labels will affect only the command which immediately follows tkey.
-f *filename*	"Keywords." The file *filename* contains the keyword assignments. See REELexchange_intro(1) for more information on the format of this file.
-p *pseudo*	"Tapesetname." Tapeset pseudo name as specified by the preceding taccess command which initiated access to the tapeset.
key = *val*	"Keywords." Set keyword *key* to value *val*. For information about keywords and values, see REELexchange_intro(1).

EXAMPLE

```
tkey blen = 8192
```

sets the block length to 8192 bytes (possible range = 1 to 32760).

FILES

/var/reelexchange/d_ibm	IBM label defaults.
/var/reelexchange/d_ansi	ANSI label defaults.

SEE ALSO

reelexchange_intro(1), taccess(1), tdisplay(1), tlabel(1), tposn(1), tread(1), trelease(1), tsniff(1), twrite(1).

NAME

tlabel – initialize a tape with a volume label

SYNOPSIS

label [-a *ADN*] [-d *density*] [-l *IBM|ANSI*] [-r] -v *VSN*

DESCRIPTION

tlabel writes a volume label onto a tape. This initializes the tape for use with the other REELexchange commands. All previous data on the tape is lost after application of this command. This is the only REELexchange command which need not be sandwiched by the commands taccess and trelease. A tape must be labeled before it can be written or read by twrite and tread. The command tlabel -r reports the VSN of the labeled tape.

OPTIONS

-a *ADN*
: "Tape Drive." Specifies which tape drive to use. The name specified should be one of the ADN files from the /var/reelexchange/adn directory. If this option is not specified, the tape drive named in the file /var/reelexchange/default.adn will be used. See **reelexchange_intro**(1) for more information on the format of ADN files.

-d *density*
: "Density." The density in *bpi* of the tape. Common values are 1600, 3200, and 6250. The density specified here should match the density field in the ADN file of the tape drive that is being accessed. Default density is 1600.

-l "Format." Specifies the format of the tape – either IBM or ANSI. Default format is ANSI.

-r "Read." Read and display the volume serial number of a tape.

-v *VSN* "VSN." The Volume Serial Number(VSN). All labeled tapes possess volume serial numbers by which they are identified. Good tape practice is to give each tape a unique volume serial number. The VSN is a six character alphanumeric symbol.

FILES

/var/reelexchange/default.adn	File containing name of default tape drive.
/var/reelexchange/Adn/*	Each file in this directory describes a tape drive on the system. The name of one of these files is given with the –a option to specify a particular tape drive for I/O.
/var/reelexchange/d_ibm	IBM label defaults.
/var/reelexchange/d_ansi	ANSI label defaults.

SEE ALSO

reelexchange_intro(1), taccess(1), tdisplay(1), tkey(1), tposn(1), tread(1), trelease(1), tsniff(1), twrite(1).

NAME

touch − update access and modification times of a file

SYNOPSIS

touch [-amc] [*mmddhhmm*[*yy*]] *files*

DESCRIPTION

Touch updates the access and modification times of each argument. You must be the owner of the file or superuser to access an existing file. The filename is created if it does not exist. If no time is specified (see date(1)), the current time is used. The −a and −m options update only the access or modification times respectively (default is −am). The −c option silently prevents touch from creating the file if it did not previously exist.

The return code from touch is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1).
utime(2) in the *Programmer's Reference for the DG/UX System*

NAME

tposn – position tape to specified file

SYNOPSIS

`tposn [-n` *num* `] [-p` *pseudo* `] {-r | -w }`

DESCRIPTION

`tposn` moves the tape to be correctly positioned for either a subsequent read (`tread`) or write (`twrite`). Read and write start at different positions for a file, so the position must be made specifically for the planned operation. With the `-r` option, `tposn` positions for a read. The `-w` option positions for a subsequent write.

`tposn` resets file labels. If positioning for a write, `tposn` resets the current file labels to default values.

Tape mount and unmount prompts are generated as needed to traverse the volumes that make up the tapeset.

Options

-n *num* "File Number." Specifies a file by its absolute order position from the beginning of the tapeset. *num* is a positive integer.

-p *pseudo*
"Tapesetname." Tapeset pseudo name as specified by the preceding `taccess` command which initiated access to the tapeset. By default, REELexchange calls each tape `default`.

-r "Read." Position for subsequent read.

-w "Write." Position for subsequent write.

SEE ALSO

`reelexchange_intro(1)`, `taccess(1)`, `tdisplay(1)`, `tkey(1)`, `tlabel(1)`, `tread(1)`, `trelease(1)`, `tsniff(1)`, `twrite(1)`.

NAME

tput – initialize a terminal or query terminfo database

SYNOPSIS

`tput` [`-T`*type*] *capname* [*parameter* ...]

`tput` [`-T`*type*] `init`

`tput` [`-T`*type*] `reset`

`tput` [`-T`*type*] `longname`

`tput` [`-T`*type*] `-S`

where:

capname	The attribute from the `terminfo` database.
parameters	An argument to be instantiated into the string.

DESCRIPTION

`tput` uses the `terminfo`(4) database to make the values of terminal-dependent capabilities and information available to users, to initialize or reset the terminal, or to return the long name of the requested terminal type. If the attribute (*cap*ability *name*) is of type string, `tput` outputs the string value of the attribute. If the attribute is of type integer, `tput` outputs a string representation of the integer value of the attribute. If the attribute is of type boolean, `tput` simply sets the exit code (`0` for TRUE if the terminal has the capability, `1` for FALSE if it does not), and produces no output. Before using a value returned on standard output, shell scripts should test the exit code to be sure it is `0`. (See the **EXIT CODES** section.)

If *capname* represents a string that takes parameters, the arguments *parameters* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number. For a complete list of capabilities and the *capname* associated with each, see `terminfo`(4).

Options are:

`-T` *type* Find information for the terminal named *type* in the `terminfo`(4) database. Normally this option is unnecessary, because the default is taken from the environment variable `TERM`. If `-T` is specified, then the environment variables `LINES` and `COLUMNS` and the dimensions stored in the line discipline (see `stty`(1)) will not be used to override the defined size of the terminal screen.

`-S` Allow more than one capability per invocation of `tput`. The capabilities must be passed to `tput` from the standard input instead of from the command line (see the **EXAMPLES** section). Only one *capname* is allowed per line. The `-S` option changes the meaning of the `0` and `1` boolean and string exit codes (see the EXIT CODES section).

Special Capabilities

`init` If the `terminfo` database is present and an entry for the user's terminal exists (see `-T` *type*, above), the following will occur: (1) if present, the terminal's initialization strings will be output (`is1, is2, is3, if, iprog`), (2) any delays (e.g., newline) specified in the entry will be set in the TTY driver, (3) tab expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

`reset` Instead of putting out initialization strings, the terminal's reset strings will be output if present (`rs1`, `rs2`, `rs3`, `rf`). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, `reset` acts identically to `init`.

`longname` If the `terminfo`(4) database is present and an entry for the user's terminal exists (see `-T` *type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the `terminfo`(4) database (see `term`(5)).

EXAMPLES

```
tput init
```

Initialize your terminal for normal use, according to the terminal type given by the environment variable `TERM`. This command would normally appear in your `.profile` or `.login` after the environment variable `TERM` has been exported (see `profile`(4)).

```
tput -T vt100 reset
```

Reset a VT100 terminal, overriding the type of terminal given by the environment variable `TERM`.

```
tput cup 0 0
```

Send the sequence to move the cursor to row `0`, column `0` (the upper left corner of the screen, usually known as the "home" cursor position).

```
tput clear
```

Echo the clear-screen sequence for your type of terminal.

```
tput cols
```

Print the number of columns for your type of terminal.

```
tput -T D462-unix cols
```

Print the number of columns for a Data General D462+ terminal.

```
bold=`tput smso`
offbold=`tput rmso`
```

Set the shell variables `bold` and `offbold` to the sequences to begin and end standout mode (respectively) for the current terminal. These variables could be used in a shell script to highlight a prompt:

```
      echo "${bold}Please type in your name: ${offbold}\c"
```

```
tput hc
```

Set the exit code to indicate if the current terminal is a hardcopy terminal.

```
tput cup 23 4
```

Send a sequence to move the cursor to row 23, column 4.

```
tput longname
```

Print the long name from the `terminfo` database for the type of terminal specified by the environment variable `TERM`.

```
tput -S <<!
> clear
> cup 10 10
> bold
> !
```

This example shows `tput` processing several capabilities in one invocation. This example clears the screen; moves the cursor to position 10, 10; and turns on bold (extra bright) mode. The list is terminated by an exclamation mark (`!`) on a line by itself.

FILES

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

`/usr/include/curses.h`
`curses`(3X) header file.

`/usr/include/term.h`
`terminfo`(4) header file.

`/usr/share/lib/tabset/*`
Tab setting files for certain terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs). For more information, see the "Tabs and Initialization" section of `terminfo`(4).

EXIT CODES

0 *capname* is of type boolean, and the *capname* is TRUE for this terminal *type*.

0 *capname* is of type integer; an exit status of `0` is always returned, whether or not *capname* is defined for this terminal *type*. The user must check standard output to determine if *capname* is defined. The message `-1` means that *capname* is not specified in the `terminfo`(4) database for this terminal *type* (e.g., `tput -T450 lines` and `tput -T2621 xmc`).

0 *capname* is of type string, and the *capname* is defined for this terminal *type*.

1 *capname* is of type boolean, and the *capname* is FALSE for this terminal *type*. (See `-S` **Option** below.)

1 *capname* is of type string, and *capname* is not defined for this terminal *type*. (See `-S` **Option** below.)

2 The command line contains invalid syntax, or some other usage error was found.

3 Terminal *type* is unknown, or the `terminfo`(4) database is inaccessible.

4 *capname* is not a valid `terminfo`(4) capability.

-S Option

If *capname* is of type boolean or string and the `-S` option is used, a value of `0` is returned to indicate that all lines were successful. No indication of which line failed can be given so exit code `1` will never appear. Exit codes `2`, `3`, and `4` retain their usual interpretation.

SEE ALSO

`clear`(1), `stty`(1), `tabs`(1), `curses`(3X), `profile`(4), `terminfo`(4), `term`(5)

NAME

`tr` – translate characters

SYNOPSIS

`tr [ -cds ] [`*string1* `[`*string2* `] ]`

DESCRIPTION

`Tr` copies the standard input to the standard output, substituting or deleting selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. You can use any combination of these options:

`-c`	Complements the set of characters in *string1* with the universe of characters whose ASCII codes are 001 through 377 octal.
`-d`	Deletes all input characters in *string1*.
`-s`	Squeezes all strings of repeated output characters in *string2* to single characters.

The following abbreviation conventions introduce ranges of characters or repeated characters into the strings:

`[a-z]`	Stands for the string of characters whose ASCII codes run from character `a` to character `z`, inclusive.
`[a*`*n*`]`	Stands for *n* repetitions of `a`. If the first digit of *n* is `0`, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

Use the escape character \ as in the shell to remove special meaning from any character in a string. In addition, \ followed by one, two, or three octal digits stands for the character whose ASCII code is given by those digits.

International Features

`tr` can process characters from supplementary code sets. Characters specified are searched for and translated in character units, not bytes.

The semantics of the "`[x-y]`" notation takes after the range specification of the regular expression syntax.

EXAMPLES

```
$ cat infile
aaaabbbccccccc
$ tr -s "[a-z]" "[A-Z]" <infile > outfile
$ cat outfile
ABC
$
```

This example causes all lower case letters in the infile to be converted to capital letters in the outfile. The -s switch causes repeated output characters to be "squeezed".

```
$ cat infile2
Mary Wadsmith 23 11/10
Tim Simon 28 1/15
Karen Adams 24 3/9
$ tr -d "[0-9]/" <infile2 >outfile2
Mary Wadsmith
Tim Simon
Karen Adams
```

```
$
```

This example causes all numeric values and slashes to be deleted from infile1. All other values are left alone. The output goes to outfile1.

```
$ cat infile3
Jim Wang - employee number 32465
Grant Stanley - employee number 98757
Cindy Eddy - employee number 76578
Mark Hoopes - employee number 78657
$ tr -cs "[0-9]" "[\012*]" <infile3 >outfile3
$ cat outfile3
32465
98757
76578
78657
$
```

This example causes all values in the infile that are not in string1, [0-9], to be converted to newlines (12 is the ascii value for newline). All of the newlines are "squeezed", and all values that are in string1 are left alone (because of the -c option).

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for new-line.

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

CAUTION

When octal notation with the backslash (\) escape character is used, a backslash is placed before each byte of characters from supplementary code set.

SEE ALSO

ed(1), sh(1).

BUGS

Will not handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

`tread` – read file(s) from tape

SYNOPSIS

`tread [ -p` *pseudo* `]`

DESCRIPTION

The `tread` command reads a file from tape and sends the output to standard output (stdout). It automatically reads file labels making them available for review via the `tdisplay` command. `tread` automatically detects the tape file format and translates records according to the current translation settings as configured by the `tkey` command.

OPTIONS

-p *pseudo*

"Tapesetname." Tapeset pseudo name as specified by the preceding `taccess` command which initiated access to the tapeset.

SEE ALSO

`reelexchange_intro`(1), `taccess`(1), `tdisplay`(1), `tkey`(1), `tlabel`(1), `tposn`(1), `trelease`(1), `tsniff`(1), `twrite`(1).

NAME

trelease – terminate access to a tape

SYNOPSIS

trelease [-p *pseudo*]

DESCRIPTION

trelease terminates access to a tapeset.

OPTIONS

-p *pseudo*

"Tapename." The name specified by the preceding taccess command which initiated access to the tape. By default, REELexchange calls each tape default.

SEE ALSO

reelexchange_intro(1), taccess(1), tdisplay(1), tkey(1), tlabel(1), tposn(1), tread(1), tsniff(1), twrite(1).

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

true does nothing, successfully. false does nothing, unsuccessfully. They are typically used in input to sh such as:

```
while true
do
        command
done
```

DIAGNOSTICS

true has exit status zero, false nonzero.

SEE ALSO

sh(1).

NAME

`tsniff` – summary report of tape contents

SYNOPSIS

`tsniff` [`-p` *pseudo*]

DESCRIPTION

`tsniff` generates a summary report of the current tapeset's contents including volume label information, file labels, and file data.

OPTIONS

`-p` *pseudo*

"Tapesetname." Tapeset pseudo name as specified by the preceding `taccess` command which initiated access to the tapeset.

SEE ALSO

`reelexchange_intro`(1), `taccess`(1), `tdisplay`(1), `tkey`(1), `tlabel`(1), `tposn`(1), `tread`(1), `trelease`(1), `twrite`(1).

NAME

`tty` – get the name of the terminal

SYNOPSIS

`tty [ -s ]`

DESCRIPTION

`tty` prints the path name of the user's terminal.

Option

`-s`	Inhibit printing of the terminal path name, allowing one to test just the exit code.

DIAGNOSTICS

``not a tty'' if the standard input is not a terminal and `-s` is not specified.

Exit Codes

2	if invalid options were specified
0	if standard input is a terminal
1	otherwise.

SEE ALSO

`ps`(1), `stty`(1), `who`(1).

NAME

twrite – writes a file to tape

SYNOPSIS

twrite [-p *pseudo*]

DESCRIPTION

The twrite command writes a file to the tape receiving input from standard input (stdin). It also writes file labels as configured by the tkey command. twrite automatically translates the UNIX data into tape records according to the current translation settings, also configured by the tkey command.

OPTIONS

-p *pseudo*

"Tapesetname." Tapeset pseudo name as specified by the preceding taccess command which initiated access to the tapeset.

SEE ALSO

reelexchange_intro(1), taccess(1), tdisplay(1), tkey(1), tlabel(1), tposn(1), tread(1), trelease(1), tsniff(1).

NAME

ul – do underlining

SYNOPSIS

`ul [ -i ] [ -t` *terminal* `] [` *name* ... `]`

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. Ul also translates overstruck characters to bold, where appropriate. The -t option overrides the terminal kind specified in the environment. Ul uses information from the terminfo database to determine the appropriate sequences for underlining and emboldening. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, ul degenerates to cat(1). If the terminal cannot underline, underlining is ignored.

The -i option causes ul to indicate underlining onto by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an nroff output stream on a crt-terminal.

EXAMPLES

```
$ ul file01
```

This command reads the file "file01" and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use. If the terminal does not support underlining, ul displays the file just as cat would.

SEE ALSO

cat(1), man(1), more(1), pg(1), curses(3X), terminfo(4).
nroff(1) in the Documenter's Toolkit.

BUGS

For users of the Documenter's Toolkit software: nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to screen permissions for files and directories created by the user. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see chmod(2) and umask(2)). The value of each specified digit is subtracted from the corresponding digit specified by the system when it creates a file (see creat(2)). For example, umask 022 removes *group* and *others* write permission; files normally created with mode 666 become mode 644.

If *ooo* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell (sh(1)) and the C shell (csh(1)).

Note that the shell and any programs running under it always create files with 666 permissions. This means that if you need execution permissions, you must add them explicitly with chmod.

Shell programs must have read and execute permissions in order to run.

EXAMPLES

Under the shell, the user file-creation mode mask has this effect:

Umask	Mode of all files created
000	666
101	666
102	664
026	640

SEE ALSO

chmod(1), csh(1), sh(1).
chmod(2), creat(2), umask(2) in the *Programmer's Reference for the DG/UX System (Volume 1)*

NAME

uname – print name of current system

SYNOPSIS

```
uname [ -amnprsv ]
```

DESCRIPTION

Uname prints the current system name of the DG/UX system on the standard output file. It is mainly useful to determine which system you are using. The options print selected information returned by uname(2) and sysinfo(2):

-a	Print all information.
-m	Print the machine hardware name.
-n	Print the nodename (the nodename may be a name that the system is known by to a communications network).
-p	Print the current host's processor type.
-r	Print the operating system release.
-s	Print the name of the operating system. This is the default.
-v	Print the operating system version.

EXAMPLES

```
$ uname
dgux
$
```

Prints the system name on the standard output.

```
$ uname -a
dgux sys08 5.4 generic AViiON mc88100
$
```

Prints the system name, the nodename, the operating system release, the operating system version, the machine hardware name, and the processor type on the standard output.

```
$ PS1=‘uname‘; export PS1
dgux
```

Sets the primary shell prompt to the name of the system and makes the new value of the variable available to all the shells that you initiate. Note that the uname command is in backquotes.

SEE ALSO

hostname(1C) in TCP/IP.
uname(2) in the *Programmer's Reference for the DG/UX System.*

NOTES

The command hostname(1C) and the system call sethostname(2) modify the system's nodename, and thus change the value returned by uname -n.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [-udc [+*n*] [-*n*]] [*input* [*output*]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see sort(1).

International Features

uniq can process characters from supplementary code sets.

OPTIONS

-u	Output just the lines that are not repeated in the original file.
-d	Writes one copy of just the repeated lines. The normal mode output is the union of the -u and -d mode outputs.
-c	Ignore the -u and -d options and generate an output report in default style, but precede each line with the number of times the line occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-*n*	The first *n* fields, together with any blanks before each, are ignored. A field is a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.
+*n*	The first *n* characters are ignored. Fields are skipped before characters.

International Features

+*n*	*n* must be in columns, not in characters.

EXAMPLES

```
$ cat in_file
This is not a unique file.
This is not a unique file.
This line occurs once.
The next line will not count as the third occurence.
This is not a unique file.
$ uniq -c in_file
   2 This is not a unique file.
   1 This line occurs once.
   1 The next line will not count as the third occurence.
   1 This is not a unique file.
```

The above example finds repeated lines in a file. NOTE: Only repeated adjacent lines are considered repeated.

SEE ALSO

comm(1), sort(1).

NAME

`units` – conversion program

SYNOPSIS

`units`

DESCRIPTION

`Units` converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

`Units` makes only multiplicative scale changes; thus, it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized. So are a few constants of nature, including:

`pi`	Ratio of circumference to diameter
`c`	Speed of light
`e`	Charge on an electron
`g`	Acceleration of gravity
`force`	Same as `g`
`mole`	Avogadro's number
`water`	Pressure head per unit height of water
`au`	Astronomical unit

`Pound` is not recognized as a unit of mass; `lb` is. Compound names are run together, (e.g., `lightyear`). British units that differ from their U.S. counterparts are prefixed thus: `brgallon`. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

`/usr/lib/unittab`

SEE ALSO

`bc(1), dc(1), expr(1).`

NAME

usage – retrieve a command description and usage examples

SYNOPSIS

[help] usage [-d] [-e] [-o] [*command_name*]

DESCRIPTION

The DG/UX system Help Facility command usage retrieves information about DG/UX system commands. With no argument, usage displays a menu screen prompting the user for the name of a command, or allows the user to retrieve a list of commands supported by usage. The user may also quit and exit to the shell by typing q.

After a command is selected, the user is asked to choose among a description of the command, examples of typical usage of the command, or descriptions of the command's options. Then, based on the user's request, the appropriate information will be printed.

A command name may also be entered at shell level as an argument to usage. To receive information on the command's description, examples, or options, the user may use the -d, -e, or -o options respectively. (The default option is -d.)

From any screen in the Help Facility, a user may execute a command via the shell (sh(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), set the shell environment variable SCROLL to no. In the Bourne shell, sh(1), do this by adding the following line to your .profile file (see profile(4)):

```
SCROLL=no; export SCROLL
```

In the csh(1), add the following line to your .login file:

```
setenv SCROLL no
```

If you later decide that you prefer scrolling, set SCROLL to yes.

Information on each of the Help Facility commands (starter, locate, usage, glossary, and help) is located on their respective manual pages.

SEE ALSO

csh(1), glossary(1), help(1), locate(1), sh(1), starter(1).
term(5) in the *Programmer's Reference for the DG/UX System*.

CAUTIONS

If the shell variable TERM (see sh(1)) is not set in the user's .profile or .login file, TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to term(5).

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [*options*] *source-files destination-file*
uulog [*options*] -s*system*
uulog [*options*] -f*system*
uuname [-l] [-c]

DESCRIPTION

uucp

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names that uucp knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, to the destination. See **CAUTIONS** and **BUGS** below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information (see **CAUTIONS** below for restrictions).

The shell metacharacters ?, * and [. . .] appearing in *path-name* will be expanded on the appropriate system.

Pathnames may be one of:

(1) a full path name;

(2) a path name preceded by ~*user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by ~/*destination* where *destination* is appended to /usr/spool/uucppublic; (NOTE: This destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ~/dan/ as the destination will create the directory /usr/spool/uucppublic/dan if it does not exist and put the requested file(s) in that directory):

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see chmod(2)).

The following options are interpreted by uucp:

-c	Do not copy local file to the spool directory for transfer to the remote machine (default).
-C	Force the copy of local files to the spool directory for transfer.
-d	Make all necessary directories for the file copy (default).
-f	Do not make intermediate directories for the file copy.
-g*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

`-j`	Output the job identification ASCII string on the standard output. This job identification can be used by `uustat` to obtain the status or terminate a job.
`-m`	Send mail to the requester when the copy is completed.
`-n`*user*	Notify *user* on the remote system that a file was sent.
`-r`	Do not start the file transfer, just queue the job.
`-s`*file*	Report status of the transfer to *file*. Note that the *file* must be a full path name.
`-x`*debug_level*	Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

uulog

`uulog` queries a log file of `uucp` or `uuxqt` transactions in a file `/usr/spool/uucp/.Log/uucico/`*system*, or `/usr/spool/uucp/.Log/uuxqt/`*system*.

The options cause *uulog* to print logging information:

`-s`*sys*	Print information about file transfer work involving system *sys*.
`-f`*system*	Does a "tail `-f`" of the file transfer log for *system*. (You must press ^D to exit this function.)
`-x`	Look in the `uuxqt` log file for the given system.
-*number*	Indicates that a "tail" command of *number* lines should be executed (see **CAUTIONS** section)..

uuname

`uuname` lists the names of systems known to `uucp`. The `-c` option lists the names of systems known to `cu`. (The two lists are the same, unless your machine is using different *Systems* files for `cu` and `uucp`. See the *Sysfiles* file.) The `-l` option returns the local system name.

EXAMPLES

```
uucp -d -m file? sys_2!/usr/spool/uucppublic
```

The above example will send any files that begin with `file` and have one other character at the end of the name (such as `file1`, `file2`, `file3`, etc.) to `sys_2`. All of the files will be deposited into the `/usr/spool/uucppublic` on `sys_2`. With the included options, directories will be created as needed, and the user will be notified by mail about the file transfer.

FILES

`/usr/spool/uucp`	spool directories
`/usr/spool/uucppublic`	public directory for receiving and sending (`/usr/spool/uucppublic`)
`/usr/lib/uucp`	directory containing program files
`/etc/uucp`	directory containing data and program files

SEE ALSO

mail(1), uustat(1), uux(1).
uucico(1M) and uuxqt(1M) in the *System Manager's Reference for the DG/UX System.*
chmod(2) in the *Programmer's Reference for the DG/UX System (Volume 1).*

CAUTIONS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin /usr/spool/uucppublic (equivalent to ~/).

All files received by uucp will be owned by uucp.
The -m option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [. . .] will not activate the -m option.

The forwarding of files through other systems may not be compatible with the previous version of uucp. If forwarding is used, all systems in the route must have the same version of uucp.

The value 0 will print out the entire file when using uulog-number.

BUGS

Protected files and files that are in protected directories that are owned by the requester can be sent by uucp. However, if the requester is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uuencode, uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [*source*] *remotedest* | mail *sys1*!*sys2*!..!decode
uudecode [*file*]

DESCRIPTION

Uuencode and uudecode are used to send a binary file via uucp (or other) mail.

Uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters. It includes the mode of the file and the *remotedest* for recreation on the remote system.

Uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user "decode" should be filtered through the uudecode program. This way, the file is created automatically without human intervention. This is possible on the uucp network by either using sendmail or by making rmail be a link to mailx instead of mail. In each case, an alias must be created in a master file to get the automatic invocation of uudecode.

If these facilities are not available, the file can be sent to a user on the remote machine who can uudecode it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

SEE ALSO

uucp(1), uux(1), mail(1), mailx(1).

NOTES

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking uudecode (often uucp) must have write permission on the specified file.

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [-a]
uustat [-m]
uustat [-p]
uustat [-q]
uustat [-k*jobid*]
uustat [-r*jobid*]
uustat [-s*system*] [-u*user*]

DESCRIPTION

Uustat will display the status of, or cancel, previously specified uucp commands, or provide general status on uucp connections to other systems. Only one of the following options can be specified with uustat per command execution:

-a Output all jobs in queue.

-m Report the status of accessibility of all machines.

-p Execute a "ps -flp" for all the process-ids that are in the lock files.

-q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of C or X files, it is the age in days of the oldest C./X. file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the -q option:

```
eagle 3C     04/07-11:07 NO DEVICES AVAILABLE
mh3bs32C     07/07-10:42 SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

-k*jobid* Kill the uucp request whose job identification is *jobid*. The killed uucp request must belong to the person issuing the uustat command unless one is the super-user.

-r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with uustat:

-s*sys* Report the status of all uucp requests for remote system *sys*.

-u*user* Report the status of all uucp requests issued by *user*.

Output for both the -s and -u options has the following format:

```
eaglen0000  4/07-11:01:03  (POLL)
eagleN1bd7  4/07-11:07     S      eagle  dan  522 /usr/dan/A
eagleC1bd8  4/07-11:07     S      eagle  dan  59 D.3b2al2ce4924
            4/07-11:07     S      eagle  dan  rmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (`rmail` - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., `D.3b2alce4924`) that is created for data files associated with remote executions (`rmail` in this example).

When no options are given, `uustat` outputs the status of all `uucp` requests issued by the current user.

FILES

`/usr/spool/uucp/*`	spool directories

SEE ALSO

uucp(1).

NAME

uuto, uupick – public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [*options*] *source-files destination*
uupick [-s *system*]

DESCRIPTION

Uuto sends *source-files* to *destination*. Uuto uses the uucp(1) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that uucp knows about (see uuname). *User* is the login name of someone on the specified system.

Two *options* are available:

-p	Copy the source file into the spool directory before transmission.
-m	Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the uucp source. By default, this directory is /usr/spool/uupublic. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by mail(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, uupick searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

new-line	Go on to next entry.
d	Delete the entry.
m [*dir*]	Move the entry to named directory *dir* (current directory is default). If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as m except moving all the files sent from *system*.
p	Print the content of the files.
q	Stop.
EOT (control-d)	Same as q.
!*command*	Escape to the shell to do *command*.
*	Print a command summary.

Uupick invoked with the -s*system* option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR	/usr/spool/uucppublic	public directory

NOTES

In order to send files that begin with a dot (e.g., `.profile`) the files must by qualified with a dot. For example: `.profile`, `.prof*`, `.profil?` are correct; whereas `*prof*`, `?profile` are incorrect.

SEE ALSO

`mail`(1), `uucp`(1), `uustat`(1), `uux`(1).
`uucleanup`(1M) in the *System Manager's Reference for the DG/UX System*

NAME

uux – UNIX-to-UNIX system command execution

SYNOPSIS

uux [*options*] *command-string*

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

NOTE: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from uux, permitting only the receipt of mail (see `mail`(1)). (Remote execution permissions are defined in `/etc/uucp/Permissions`.)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*`!`. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

As an example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

will get the `file1` and `file2` files from the "usg" and "pwba" machines, execute a `diff`(1) command and put the results in `file.diff` in the local `PUBDIR/dan/` directory.

Any special shell characters such as `<>;` | should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
    uux a!cut -f1 b!/usr/file \(c!/usr/file\)
```

gets /usr/file from system "b" and sends it to system "a", performs a `cut` command on that file and sends the result of the `cut` command to system "c".

uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the `-n` option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by uux:

- `-` The standard input to uux is made the standard input to the *command-string*.
- `-a`*name* Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
- `-b` Return whatever standard input was provided to the uux command if the exit status is non-zero.
- `-c` Do not copy local file to the spool directory for transfer to the remote machine (default).

`-C`	Force the copy of local files to the spool directory for transfer.
`-g`*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.
`-j`	Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by `uustat` to obtain the status or terminate a job.
`-n`	Do not notify the user if the command fails.
`-p`	Same as -: The standard input to `uux` is made the standard input to the *command-string*.
`-r`	Do not start the file transfer, just queue the job.
`-s`*file*	Report status of the transfer in *file*.
`-x`*debug_level*	Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.
`-z`	Send success notification to the user.

FILES

`/usr/lib/uucp/spool`	spool directories
`/etc/uucp/Permissions`	remote execution permissions
`/usr/lib/uucp`	directory containing program files
`/etc/uucp`	directory containing data and program files

SEE ALSO

`cut(1)`, `mail(1)`, `uucp(1C)`, `uustat(1C)`.

CAUTIONS

Only the first command of a shell pipeline may have a *system-name*`!`. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do.
The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the `uucp` system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the `uux` request. The following command will NOT work:

```
     uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

```
     uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work (if `diff` is a permitted command).

BUGS

Protected files and files that are in protected directories that are owned by the requester can be sent in commands using `uux`. However, if the requester is root, and the directory is not searchable by "other", the request will fail.

NAME

vacation – automatically respond to incoming mail messages

SYNOPSIS

vacation [-l *logfile*] [-m *mailfile*] [-M *canned_msg_file*] [-F *failsafe*] [-d]

DESCRIPTION

When a new mail message arrives, the mail command first checks if the recipient's mailbox indicates that the message is to be forwarded elsewhere (to some other recipient or as the input to some command). vacation is used to set up forwarding on the user's mailbox so that the new message is saved into an alternative mailbox and a canned response is sent to the message's originator.

Options

-l *logfile* File to keep track of which originators have already seen the canned response. If not specified, it defaults to $HOME/.maillog.

-m *mailfile* Alternate mailbox to save new messages into. If not specified, it defaults to $HOME/.mailfile.

-M *canned_msg_file*
File to send back as the canned response. If *canned_msg_file* is not specified, it defaults to /usr/share/lib/mail/std_vac_msg, which contains:

```
Subject: AUTOANSWERED!!!

I am on vacation. I will read (and answer if necessary)
your e-mail message when I return.

This message was generated automatically and you will
receive it only once, although all messages you send
me while I am away WILL be saved.
```

-F *failsafe* If mail has troubles delivering to the mailfile specified, it may optionally be forwarded to another login id (*failsafe*) instead of being returned to the sender.

-d The log file will have the day's date appended.

Unsetting the Automatic Response

To remove the vacation functionality, use

```
mail -F ""
```

FILES

/tmp/notif*	Temporary file
/usr/share/lib/mail/std_vac_msg	Default canned response
/var/mail/*	Users' standard mailboxes
/usr/lib/mail/vacation2	Program that actually sends back the canned response

SEE ALSO

mail(1), mailx(1).

NOTES

Because `vacation` uses the "`Forward to` | *command*" facility of `mail` to implement notifications, `/var/mail/`*username* should `not` be specified as the place to put newly arrived messages via the `-m` invocation option. The `mail` command uses `/var/mail/`*username* to hold either mail messages or indications of mail forwarding, but not both simultaneously.

NAME

vi, view, vedit – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [-t *tag*] [-r *file*] [-l] [-L] [-w*n*] [-R] [-x] [-C] [-c *command*] *file*...
view [-t *tag*] [-r *file*] [-l] [-L] [-w*n*] [-R] [-x] [-C] [-c *command*] *file*...
vedit [-t *tag*] [-r *file*] [-l] [-L] [-w*n*] [-R] [-x] [-C] [-c *command*] *file*...

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor ex. It is possible to use the command mode of ex from within vi and vice-versa. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all ex line editor commands are described on the ex(1) manual page.

When using vi, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

Invocation Options

The following invocation options are interpreted by vi (previously documented options are discussed in the NOTES section of this manual page):

-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	Set up for editing LISP programs.
-L	List the name of all files saved as the result of an editor or system crash.
-w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
-R	Readonly mode; the readonly flag is set, preventing accidental overwriting of the file.
-x	Encryption option; when used, vi simulates the X command of ex 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 WARNING section at the end of this manual page.
-C	Encryption option; same as the -x option, except that vi simulates the C command of ex. The C command is like the X command of ex, 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.

The *view* invocation is the same as vi except that the readonly flag is set.

The *vedit* invocation is intended for beginners. It is the same as vi except that the report flag is set to 1, the showmode and novice flags are set, and magic is turned off. These defaults make it easier to learn how to use `vi`.

vi Modes

Command	Normal and initial mode. Other modes return to command mode upon completion. `ESC` (escape) is used to cancel a partial command.
Input	Entered by setting any of the following options: `a A i I o O c C s S R` . Arbitrary text may then be entered. Input mode is normally terminated with `ESC` character, or, abnormally, with an interrupt.
Last line	Reading input for `:` `/` `?` or `!`; terminate by typing a carriage return; an interrupt cancels termination.

COMMAND SUMMARY

In the descriptions, `CR` stands for carriage return and `ESC` stands for the escape key.

Sample commands

← ↓ ↑ →	arrow keys move the cursor
`h j k l`	same as arrow keys
`i`*text*`ESC`	insert *text*
`cw`*new*`ESC`	change word to *new*
`ea`*s*`ESC`	pluralize word (end of word; append `s`; escape from input state)
`x`	delete a character
`dw`	delete a word
`dd`	delete a line
`3dd`	delete 3 lines
`u`	undo previous change
`ZZ`	exit `vi`, saving changes
`:q!CR`	quit, discarding changes
/text`CR`	search for *text*
`^U ^D`	scroll up or down
`:`*cmd*`CR`	any `ex` or `ed` command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	`z G |`
scroll amount	`^D ^U`
repeat effect	most of the rest

Interrupting, canceling

`ESC`	end insert or incomplete cmd
`DEL`	(delete or rubout) interrupts

File manipulation

`ZZ`	if file modified, write and exit; otherwise, exit
`:wCR`	write back changes
`:w ! CR`	forced write, if permission originally not valid
`:qCR`	quit
`:q ! CR`	quit, discard changes

`:e` *name*CR	edit file *name*
`:e!`CR	reedit, discard changes
`:e +` *name*CR	edit, starting at end
`:e +`*n*CR	edit starting at line *n*
`:e #`CR	edit alternate file
`:e! #`CR	edit alternate file, discard changes
`:w` *name*CR	write file *name*
`:w!` *name*CR	overwrite file *name*
`:sh`CR	run shell, then return
`:!`*cmd*CR	run *cmd*, then return
`:n`CR	edit next file in arglist
`:n` *args*CR	specify new arglist
`^G`	show current file and line
`:ta` *tag*CR	position cursor to *tag*

In general, any `ex` or `ed` command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

`^F`	forward screen
`^B`	backward screen
`^D`	scroll down half screen
`^U`	scroll up half screen
n`G`	go to the beginning of the specified line (end default), where *n* is a line number
/pat	next line matching *pat*
?pat	previous line matching *pat*
`n`	repeat last `/` or `?` command
`N`	reverse last `/` or `?` command
/pat/+n	nth line after *pat*
?pat?-n	nth line before *pat*
`]]`	next section/function
`[[`	previous section/function
`(`	beginning of sentence
`)`	end of sentence
`{`	beginning of paragraph
`}`	end of paragraph
`%`	find matching `( ) {` or `}`

Adjusting the screen

`^L`	clear and redraw window
`^R`	clear and redraw window if `^L` is → key
`z`CR	redraw screen with current line at top of window
`z-`CR	redraw screen with current line at bottom of window
`z.`CR	redraw screen with current line at center of window
/pat/`z-`CR	move *pat* line to bottom of window
`z`*n*`.`CR	use *n*-line window
`^E`	scroll window down 1 line
`^Y`	scroll window up 1 line

Marking and returning

`` ` ` ``	move cursor to previous context
`' '`	move cursor to first non-white space in line
m*x*	mark current position with the ASCII lower-case letter *x*
`` ` ``*x*	move cursor to mark *x*
'*x*	move cursor to first non-white space in line marked by *x*

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
−	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character positioning

^	first non white-space character
0	beginning of line
$	end of line
h or →	forward
l or ←	backward
^H	same as ← (backspace)
space	same as → (space bar)
f*x*	find next *x*
F*x*	find previous x
t*x*	move to character prior to next *x*
T*x*	move to character following previous *x*
;	repeat last f F t or T
,	repeat inverse of last f F t or T
n|	move to column *n*
%	find matching ([ ) or]

Words, sentences, paragraphs

w	forward a word
b	back a word
e	end of word
)	to next sentence
]	to next paragraph
(	back a sentence
[	back a paragraph
W	forward a blank-delimited word
B	back a blank-delimited word
E	end of a blank-delimited word

Corrections during insert

`^H`	erase last character (backspace)
`^W`	erase last word
erase	your erase character, same as `^H` (backspace)
kill	your kill character, erase this line of input
\	quotes your erase and kill characters
`ESC`	ends insertion, back to command mode
`DEL`	interrupt, terminates insert mode
`^D`	backtab one character; reset left margin of *autoindent*
`^^D`	caret (^) followed by control-d (^D); backtab to beginning of line; do not reset left margin of *autoindent*
`0^D`	backtab to beginning of line; reset left margin of *autoindent*
`^V`	quote non-printable character

Insert and replace

`a`	append after cursor
`A`	append at end of line
`i`	insert before cursor
`I`	insert before first non-blank
`o`	open line below
`O`	open above
`r`*x*	replace single char with *x*
`R`*text*`ESC`	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since `w` moves over a word, `dw` deletes the word that would be moved over. Double the operator, e.g., `dd` to affect whole lines.

`d`	delete
`c`	change
`y`	yank lines to buffer
`<`	left shift
`>`	right shift
`!`	filter through command

Miscellaneous Operations

`C`	change rest of line (`c$`)
`D`	delete rest of line (`d$`)
`s`	substitute chars (`cl`)
`S`	substitute lines (`cc`)
`J`	join lines
`x`	delete characters (`dl`)
`X`	delete characters before cursor (`dh`)
`Y`	yank lines (`yy`)

Yank and Put

Put inserts the text most recently deleted or yanked; however, if a buffer is named (using the ASCII lower-case letters `a` - `z`), the text in that buffer is put instead.

`3yy`	yank 3 lines
`3yl`	yank 3 characters
`p`	put back text after cursor
`P`	put back text before cursor
`"`*x*`p`	put from buffer *x*
`"`*x*`y`	yank to buffer *x*
`"`*x*`d`	delete into buffer *x*

Undo, Redo, Retrieve

`u`	undo last change
`U`	restore current line
`.`	repeat last change
`"`*d*`p`	retrieve *d*'th last delete

International Features

`vi` can process and display characters from supplementary character sets using a consistent user interface.

All processing is in character units, not columns or bytes. Accordingly, in *command mode*, `vi` recognizes arguments to indicate the number of characters.

In regular expressions, also, processing is performed on characters, not bytes.

Multi-column characters are split over two lines when using the full screen width. `vi` displays the same number of ASCII `>` characters as the split character's display width.

Commands:

`r`*x*, `f`*x*, `F`*x*, `t`*x*, `T`*x*
: Accompanying argument *x* must be a single-byte character.

AUTHOR

`vi` and `ex` were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

`$HOME/.exrc`	Editor initialization file
`./.exrc`	Editor initialization file
`/tmp`	default directory where temporary work files are placed; it can be changed using the `directory` option [see the `ex`(1) `set` command]
`/usr/share/lib/terminfo/?/*`	compiled terminal description database
`/usr/lib/.COREterm/?/*`	subset of compiled terminal description database

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard [see `intro`(1)]. 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 are provided with the Security Administration Utilities package, which is available only in the United States.

Tampering with entries in `/usr/share/lib/terminfo/?/*` or `/usr/share/lib/terminfo/?/*` (for example, changing or removing an entry) can affect programs such as `vi` that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

Software tabs using `^T` work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

If you use a Data General terminal, make sure that the emulation mode is a VT mode or ANSI mode or that UNIX mode is enabled. For example, you can run in VT100 emulation mode; if you run in D217 mode, UNIX mode must be enabled. AViiON DG/UX `vi` does not support terminals in a DG mode unless UNIX mode is enabled.

If you are using a Data General terminal in ANSI mode, make sure your `stty` settings include `-onlcr`, `-icrnl`, and `tab0`. These can easily be selected by issuing `stty nl -tabs`.

SEE ALSO

`ed`(1), `edit`(1), `ex`(1), `term`(5).
Using the DG/UX System.
Using the DG/UX Editors.
`curses` (ETI) and `terminfo` chapters of *Programmer's Guide: Character User Interface (FMLI and ETI)*.

NAME

`wait` – await completion of process

SYNOPSIS

`wait` [*n*]

DESCRIPTION

Wait for your background process whose process id is *n* and report its termination status. The optional process id argument, *n*, is available only through the `sh` and `ksh`. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes `wait`, without creating a new process.

SEE ALSO

`csh`(1), `ksh`(1), `sh`(1).

BUGS

Not all the processes of a pipeline with three or more stages are children of the shell; thus, they cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

wc – word count

SYNOPSIS

`wc [-lwc]` [*names*]

DESCRIPTION

`wc` counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options `l`, `w`, and `c` may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is `-lwc`.

When *names* are specified on the command line, they will be printed along with the counts.

SEE ALSO

`nawk(1)`, `nl(1)`.

NAME

`whatis` – display a one-line summary about a topic

SYNOPSIS

`whatis` *topic* ...

where:

topic	A reference manual entry name to identify

DESCRIPTION

`whatis` looks up a given *topic* in the `whatis` database and displays the **NAME** line associated with the manual entry. You can then run the `man`(1) command to get more information. If a `whatis` output line starts with *filename*`(`*section*[*x*]`)`, where *section* is a digit and *x* is a lowercase letter, you can enter the following command to get the documentation for it:

`man` *section filename*

EXAMPLES

To find out what information is available on the topic `passwd`, and then display the documentation directly concerning the "password file":

```
whatis passwd
```

followed by

```
man 4 passwd
```

FILES

`/usr/catman/?_man/whatis`
Table of contents data bases

SEE ALSO

`man`(1), `apropos`(1).

NOTES

`whatis` is actually just the `-f` option to the `man`(1) command.

NAME

whereis – locate source, binary, and or manual for program

SYNOPSIS

whereis [-sbm] [-u] [-SBM *dir* ... -f] *name* ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". Prefixes of "s." resulting from use of source code control are also dealt with. Whereis then attempts to locate the desired program in a list of standard places. If any of the -b, -s or -m flags are given then whereis searches only for binaries, sources or manual sections respectively (or any two thereof). The -u flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type.

Finally, the -B -M and -S flags may be used to change or otherwise limit the places where whereis searches. The -f flag is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following example finds all files in /usr/bin that are undocumented in /usr/catman/u_man/man1.

```
cd /usr/bin
whereis -M /usr/catman/u_man/man1 -f -m -u *
```

FILES

```
/usr/src/*
/usr/catman/?_man/man? (DG/UX)
/bin, /lib, /etc, /usr/{lib,bin}
```

SEE ALSO

which(1).

NAME

`which` – locate a program file for csh(1) users

SYNOPSIS

`which` [*name*] ...

DESCRIPTION

`Which` takes a list of names and looks for the files which would be executed had these names been given as commands. This command only works for `csh`(1) users. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's `.cshrc` file.

FILES

`~/.cshrc`	source of aliases and path values

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

SEE ALSO

`csh`(1), `sh`(1), `whereis`(1).

NOTES

Only aliases and paths from `~/.cshrc` are used; importing from the current environment is not attempted.

`Which` must be executed from `csh` because aliases only exist in the `csh`.

To compensate for `~/.cshrc` files in which aliases depend upon the `prompt` variable being set, `which` sets this variable. If the `~/.cshrc` produces output or prompts for input when `prompt` is set, `which` may produce some strange results.

NAME

who – who is on the system

SYNOPSIS

who [-uTHlpdbrtasqh] [*file*]

who -qn *x* [*file*]

who am i

who am I

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process ID of the shell for each current system user. It examines the /etc/utmp file to get this information, or it examines *file* if it is given. Usually, *file* will be /etc/wtmp, which contains a history of all the logins since the file was last created.

Who with the am i or am I option identifies the invoking user.

Except for the default -s option, the general format for output from this command is:

name [*state*] *line time activity pid* [*comment*] [*exit*] [*hostname*]

With options, who can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the init process. These options are:

-u Lists only those users currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory /dev.

The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not.

The *pid* is the process ID of the user's shell.

The *comment* is the comment field associated with this line as found in /etc/inittab (see inittab(4)). This can contain information such as where the terminal is located, the telephone number of the dataset, and the type of terminal if it is hard-wired.

-T Same as the -u option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a – appears if it is not. Root can write to all lines having a + or a – in the *state* field. If a bad line is encountered, a ? is printed.

-l Lists only those lines on which the system is waiting for someone to login. The *name* field in this case is LOGIN. Other fields are the same as for user entries, except that the *state* field does not exist.

-p Lists any other active process that was also spawned by init. The *name* field is the name of the program executed by init as found in /etc/inittab. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from /etc/inittab that spawned this process. See inittab(4).

-d Displays all processes that have expired and not been respawned by init. The *exit* field appears for a dead process and contains its termination and exit values, as returned by wait(2) . This can be useful in determining why a process terminated.

-b Indicates the time and date of the last reboot.

-r Indicates the current *run-level* of the init process.

-t Indicates the last change to the system clock (via the date(1) command) by root. See su(1).

-a Processes /etc/utmp or the named *file* with all options turned on.

-s The default--lists only the *name*, *line*, and *time* fields.

-H Prints column headings above the regular output.

-q This is a quick who, displaying only the names and the number of users currently logged on. When this option is used, all other options except n are ignored.

-n *x* This option should be used with -q. It lets you specify the number of users you want to be displayed per line.

-h This option prints host names for remote users

EXAMPLES

```
$ who am i
user1      tty21       Nov 19 09:54
```

The above example displays the user who invoked the who command.

```
$ who -u
sys_mgr    tty01       Nov 19 09:00  1:53    690
donnelly   tty02       Nov 19 08:56  0:20    144
crumley    tty20       Nov 19 09:32   .      151
user1      tty21       Nov 19 09:54   .      974
user2      tty12       Nov 19 13:19  0:06   1728
```

The above example shows all users logged onto the system, which terminal they have logged onto, the date and time, the amount of time since last activity ("." indicates less than one minute), and the user's PID.

```
$ who -b
    .      system boot  Nov 18 21:34
```

The above example shows the date and time that the system was last booted.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), login(1), mesg(1), ps(1), su(1).
wait(2), inittab(4), utmp(4) in the *Programmer's Reference for the DG/UX System*
init(1M) in the *System Manager's Reference for the DG/UX System*

NAME

`write` – write to another user

SYNOPSIS

`write` *user* [*line*]

where:

user	A user name (login name)
line	A terminal line (e.g., `tty00`) to which *user* is connected

DESCRIPTION

`Write` copies lines from your terminal to that of another user. When first called, it sends this message to the person you want to talk to:

`Message from` *yourname* `(tty` *??*`)` [*date*]. . .

When it has successfully completed the connection, `write` also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n." At that point, `write` writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, use the *line* argument to indicate which line or terminal to send to; otherwise, the first writable instance of the user found in `/etc/utmp` is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted with the `mesg`(1) command. Writing to others is normally allowed by default. Certain commands, in particular `nroff`(1) and `pr`(1) , disallow messages to prevent interference with their output. However, if the sender has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character `!` is found at the beginning of a line, `write` calls the shell to execute the rest of the line as a command.

We suggest the following protocol for using `write`: when you first write to another user, wait for them to write back before starting to send. Each person should end a message with a distinctive signal (e.g., `o` for "over") so that the other person knows when to reply. The signal `oo` (for "over and out") is useful when you're terminating the conversation.

International Features

`write` can send characters from supplementary code sets.

`write` uses the locale of the sender to determine printability.

FILES

`/etc/utmp`	To find user
`/bin/sh`	To execute `!`

DIAGNOSTICS

`user is not logged on`
: The person you are trying to write to is not logged on.

`Permission denied`
: The person you are trying to write to denies that permission (with `mesg`).

`Warning: cannot respond, set mesg -y`
: Your terminal is set to `mesg n`; therefore the recipient cannot respond to you.

`Can no longer write to user`
: The recipient has denied permission (`mesg n`) after you have started writing.

SEE ALSO

`mail`(1), `mesg`(1), `nroff`(1), `pr`(1), `sh`(1), `who`(1).

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

`xargs` [*flags*] [*command* [*initial-arguments*]]

DESCRIPTION

`Xargs` combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The flags you specify determine the number of arguments read for each *command* invocation and how they are combined.

Command, which may be a shell file, is searched for, using one's `$PATH`. If *command* is omitted, `/bin/echo` is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see `-i` flag). Flags `-i`, `-l`, and `-n` determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full. Then, *command* is executed with the accumulated arguments. This process is repeated until there are no more arguments. When there are flag conflicts (e.g., `-l` vs. `-n`), the last flag has precedence. *Flag* values are:

`-l`*number* *Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option `-x` is forced.

`-i`*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single argument and inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of five arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option `-x` is also forced. `{ }` is assumed for *replstr* if not specified.

`-n`*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option `-x` is also coded, each *number* arguments must fit in the *size* limitation, or `xargs` terminates execution.

`-t` Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just before their execution.

`-p`	Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (`-t`) is turned on to print the command instance to be executed, followed by a `?...` prompt. A reply of `y` (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.
`-x`	Causes `xargs` to terminate if any argument list would be greater than *size* characters; `-x` is forced by the options `-i` and `-l`. When none of the options `-i`, `-l`, or `-n` are coded, the total length of all arguments must be within the *size* limit.
`-s`*size*	The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If `-s` is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument, plus the count of characters in the command name.
`-e`*eofstr*	*Eofstr* is taken as the logical end-of-file string. Underscore (_) is assumed for the logical EOF string if `-e` is not coded. The value `-e` with no *eofstr* coded turns off the logical EOF string capability (underscore is taken literally). `Xargs` reads standard input until either end-of-file or the logical EOF string is encountered.

`Xargs` will terminate if it receives a return code of `-1` from *command*, or if it cannot execute *command*. When *command* is a shell program, it should explicitly *exit* (see `sh`(1)) with an appropriate value to avoid accidentally returning with `-1`.

EXAMPLES

In the Bourne shell the following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

In the Bourne shell the following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived. `Xargs` archives them into *arch* one at a time (1.), or many at a time (2.).

```
1. ls | xargs -p -l ar r arch
2. ls | xargs -p -l | xargs ar r arch
```

The following will execute `diff`(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

DIAGNOSTICS

Self-explanatory.

SEE ALSO

`sh`(1).

NAME

`ypcat` – print values in an NIS data base

SYNOPSIS

`ypcat [ -kt ] [ -d` *domainname* `]` *mname*

`ypcat -x`

DESCRIPTION

`ypcat` prints out values in a Network Information Service (NIS) map specified by *mname*, which may be either a map name or a map nickname . Since `ypcat` uses the NIS network services, no NIS server is specified.

To look at the network-wide password database, `passwd.byname` (with the nickname `passwd`), type:

```
ypcat passwd
```

Refer to `ypfiles`(4) and `ypserv`(1M) for an overview of the Network Information Service.

OPTIONS

`-k` Display the keys for those maps in which the values are null or the key is not part of the value.

`-t` Inhibit translation of *mname* to map name. For example, '`ypcat -t passwd`' will fail because there is no map named `passwd`, whereas '`ypcat passwd`' will be translated to '`ypcat passwd.byname`'.

`-d` *domainname*
Specify a domain other that the default domain. The default domain is returned by *domainname*.

`-x` Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

`domainname`(1), `ypmatch`(1), `ypserv`(1M), `ypfiles`(4).

NAME

`ypmatch` – print the value of one or more keys from an NIS map

SYNOPSIS

`ypmatch` [`-d` *domain*] [`-k`] [`-t`] *key* ... *mname*

`ypmatch -x`

DESCRIPTION

`ypmatch` prints the values associated with one or more keys from the Network Information Service (NIS) map specified by *mname*, which may be either a *mapname* or an map nickname.

Multiple keys can be specified; the same map will be searched for all . The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

OPTIONS

`-d` Specify a domain other than the default domain.

`-k` Before printing the value of a key, print the key itself, followed by a '`:`' colon . This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

`-t` Inhibit translation of nickname to *mapname*. For example, '`ypmatch -t zippy passwd`' will fail because there is no map named `passwd`, while '`ypmatch zippy passwd`' will be translated to `ypmatch zippy passwd.byname ' .

`-x` Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

`ypcat`(1), `ypfiles`(4).

NAME

`yppasswd` – change your network password in the Network Information Service

SYNOPSIS

`yppasswd` [*name*]

DESCRIPTION

`yppasswd` changes (or installs) a network password associated with the user *name* (your own name by default) in the Network Information Service. The Network Information Service password may be different from the one on your own machine.

`yppasswd` prompts for the old Network Information Service password, and then for the new one. You must type in the old password correctly for the change to take effect. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long, if they use a sufficiently rich alphabet, and at least six characters long if monocase. These rules are relaxed if you are insistent enough. Only the owner of the name or the super-user may change a password; in either case you must prove you know the old password.

The Network Information Service password daemon, `yppasswdd`(1M) must be running on the master NIS server in order for the new password to take effect.

NOTE:

Secure RPC using DES Authentication is an additional feature that must be purchased separately from the DG/UX™ ONC™/NFS® package.

If Secure RPC is installed, `yppasswd`(1) updates the public/secret key pair in the public key database and returns a message indicating that the public key database has been updated.

SEE ALSO

`passwd`(1), `yppasswdd`(1M), `ypfiles`(4).

BUGS

The update protocol passes all the information to the server in one RPC call, without ever looking at it. Thus if you type in your old password incorrectly, you will not be notified until after you have entered your new password.

NAME

`ypwhich` – which host is the NIS server or map master?

SYNOPSIS

`ypwhich` [`-d` [*domain*]] [`-V1` | `-V2`] [*hostname*]
`ypwhich` [`-t` *mapname*] [`-d` *domain*] `-m` [*mname*]
`ypwhich -x`

DESCRIPTION

`ypwhich` tells which NIS server supplies the Network Information Service to an NIS client, or which is the master for a map. If invoked without arguments, it gives the NIS server for the local machine. If *hostname* is specified, that machine is queried to find out which NIS master it is using.

Refer to `ypfiles`(4) and `ypserv`(1M) for an overview of the Network Information Service.

OPTIONS

`-d` Use *domain* instead of the default domain.

`-V1` Which server is serving v.1 NIS protocol client processes.

`-V2` Which server is serving v.2 NIS protocol client processes.

If neither version is specified, `ypwhich` attempts to locate the server that supplies the (current) v.2 services. If there is no v.2 server currently bound, `ypwhich` then attempts to locate the server supplying the v.1 services. Since NIS servers and NIS clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

`-t` *mapname*
Inhibit nickname translation; useful if there is a *mapname* identical to a nickname. This is not true of any DG/UX ONC/NFS-supplied map.

`-m` Find the master NIS server for a map. No *hostname* can be specified with `-m`. *mname* can be a mapname, or a nickname for a map. When *mname* is omitted, produce a list of available maps.

`-x` Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

`rpcinfo`(1M), `ypserv`(1M), `ypset`(1M), `ypfiles`(4).

End of Chapter

Chapter 2
Miscellaneous Features

This chapter contains miscellaneous manual pages of interest to general users. Most miscellaneous man pages deal with specific programming topics and are therefore in the *Programmer's Reference to the DG/UX System*.

For a more detailed discussion of the Editread command-line editor, see *Using the DG/UX Editors*.

NAME

editread – command line editor

DESCRIPTION

Editread is an optional interface that you can use in sh(1), csh(1), and crash(1M) for editing command lines. Editread also offers a history facility that saves command lines for later recall, editing, and execution. The editread history facility is separate from the C shell history facility; their functions are similar but their implementations and use are different.

Editread is disabled by default. To enable the facility, you need a file named .editreadrc in your home directory or current directory. You can create an empty file named .editreadrc, which causes the default editread values to be recognized. Alternatively, you can copy /usr/lib/editreadrc.proto to $HOME/.editreadrc. After you have a .editreadrc file, it will be in effect for all child processes of a parent process (but not for the parent itself). Editread will be active the next time you log in.

To find out the current values for the editread functions, use your reconfig key (<Ctrl-R> by default). The following screen shows the default values.

```
E D I T R E A D   C O N F I G U R A T I O N   D I S P L A Y

CURSOR CONTROL          LINE EDITING            HISTORY

backward     = ^b       insert       = ^n       hist_display   = 23
forward      = ^f       insert_space = OFF      hist_save      = 100
goto_end     = ^e       erase        = DEL      hist_recall    = ESC
goto_end_ov  =          word_erase   = ^t       hist_scan      = ^p
home         = ^a       kill         = ^u       hist_up        = UP
left         = LEFT     delete_end   = ^k       hist_down      = DOWN
right        = RIGHT

PROCESS CONTROL         CONFIGURING EDITREAD    MISCELLANEOUS

eof          = ^d       enable       = ON       prompt         = OFF
intr         = ^c       reconfig     = ^r       refresh        =
quit         = ^\                               verbatim       = ^v
susp         = ^z                               term           = ansi
```

Note: The caret (^) in this configuration display represents the <Ctrl> key.

Functions and Default Values

Each function and its default value is described in the table below:

Keyword	Description	Default Value
	Editread Configuration	
enable	Turns editread off or on.	ON
reconfig	Displays the current editread values and concludes a redefinition of a single function or multiple functions.	`<Ctrl-R>`
	Cursor Control	
right	Moves the cursor one position to the right.	Right Arrow key
left	Moves the cursor one position to the left.	Left Arrow key
goto_end	Moves the cursor to the end of the line.	`<Ctrl-E>`
goto_end_ov	Retrieves the previous command line and appends it to the current command line at the cursor position.	unassigned
home	Returns the cursor to the beginning of the line.	`<Ctrl-A>`
forward	Moves the cursor to the beginning of the next word.	`<Ctrl-F>`
backward	Moves the cursor backward to the space after the previous word.	`<Ctrl-B>`
	Line Editing	
erase	Erases a character one position to the left of the cursor.	`<Del>`
word_erase	Deletes from the current cursor position through the end of the word.	`<Ctrl-T>`
delete_end	Deletes from the cursor position to the end of the line.	`<Ctrl-K>`
kill	Erases the entire line.	`<Ctrl-U>`
insert	Enables and disables (toggles) insert mode.	`<Ctrl-N>`
insert_space	When insert mode is in effect, a leading space always appears to the left of the cursor.	OFF

(Continued)

Keyword	Description	Default
	History	
hist_recall	Displays the history list.	`<Esc>`
hist_scan	Searches the history list for a given regular expression and shows matches.	`<Ctrl-P>`
hist_up	Recalls the previous command in the history list.	Up Arrow key
hist_down	Recalls the next command in the history list.	Down Arrow key
hist_save	Sets the maximum number of commands to be saved in history.	100
hist_display	Sets the number of commands to be displayed at one time when you press the `reconfig` key.	23
write_hist	Is not a function you can configure but a command you can use to write history commands to a file.	`write_hist` *file*`<reconfig>`
read_hist	Is not a function you can configure but a command you can use to read a file containing a history list to the current history list.	`read_hist` *file*`<reconfig>`
	Process Control	
eof	Sets the end-of-file character.	`<Ctrl-D>`
intr	Sets the interrupt key.	`<Del>`
quit	Sets the quit key.	`<Ctrl-\>`
susp	Sets the suspend key.	`<Ctrl-Z>`
	Miscellaneous	
prompt	Prints an editread prompt ($) preceded by the current history number.	OFF
refresh	Refreshes the current line.	unassigned
verbatim	Nullifies (escapes) the meaning of a key to editread.	`<Ctrl-V>`
term	Identifies your terminal type.	`ansi`

Assigning Values to Functions

You can reassign values to editread functions in a `.editreadrc` file (in either the home or current directory), in the `EDITREAD` environment variable, or interactively. The following order of precedence is followed when enabling editread:

— editread value set interactively,

— `EDITREAD` environment variable,

— `.editreadrc` in the current directory,

— `.editreadrc` in the home directory.

Pressing the `reconfig` key (default is `<Ctrl-R>`) gives a display of the current editread values, which are comprised of the defaults, those set interactively, and those set in an `.editreadrc` file or the `EDITREAD` environment variable.

Any keyboard key can be assigned to any editread function with the following exceptions. Functions `enable`, `insert_space`, and `prompt` take a toggle value, ON or OFF; two commands, `hist_save` and `hist_display`, take numeric values; and the `term` function takes a string value. To disable any function, set its value to OFF. On the configuration display, a blank setting implies a disabled condition.

In addition to literal keyboard keys, you can set editread functions to mnemonic names. These names represent special keys that generate unprintable characters or that vary from terminal to terminal; for example, control characters, function keys, and cursor control keys. See the *Special Keys* section for a list of the available names and other related information.

You reassign values to functions using this format:

function-keyword = *value* [*function-keyword* = *value*]

You can use any amount of spacing between keywords and values for readability. Interactive assignments are terminated with the `reconfig` key. When using the `EDITREAD` environment variable, put single quotation marks ('') around the entire list of function keywords and values.

You can temporarily disable editread (for the remainder of a log-in session) by interactively assigning the OFF value to the `enable` function. Or, you can disable editread in one shell (such as the C shell) while keeping it enabled in the Bourne shell. To disable editread permanently, you can delete your `.editreadrc` file(s), delete or unset the `EDITREAD` environment variable, or set the `enable` function to OFF (either interactively, in the `EDITREAD` environment variable, or in a `.editreadrc` file).

By default, the line-editing and control keys defined in editread are copied from your terminal's line discipline. For example, both editread and your terminal define `<Del>` as the erase key and `<Ctrl-U>`> as the kill (delete-line) key. If you redefine a key in editread, the change will be automatically propagated to your line discipline. Similarly, if you make a change to your line discipline with the `stty`(1) command, editread will reconfigure the appropriate function to match the line discipline change.

Special Keys

Editread recognizes the special keys listed in the two tables below. Some special keys are provided for all terminals and are part of the editread default values; these are listed in the first table. The remainder of the special keys are not available on all terminals; these are listed in the second table. For keys listed in the first table, you can always set an editread function to the mnemonic name of the key. For keys listed in the second table, however, you can use mnemonic names for only those keys supported by your terminal.

Editread accesses the `terminfo`(4) database to determine which special keys your terminal supports. For more information, consult `term`(5) and `terminfo`(4).

The following mnemonic names are provided for all terminals:

^*C*	The control form of character *C* (`<Ctrl-`*C*`>`).
`CR`	The carriage return (enter) key.
`DEL`	The delete (`<Ctrl-?>`) key.
`DOWN`	The cursor down (down arrow) key.
`ESC`	The escape (`<Ctrl-[>`) key.
`HOME`	The home key.
`LEFT`	The cursor left (left arrow) key.
`NL`	The newline (linefeed) key.
`RIGHT`	The cursor right (right arrow) key.
`UP`	The cursor up (up arrow) key.

The following mnemonic names are terminal dependent:

`BREAK`	The break key.
`BACKSPACE`	The backspace key.
`F`*n*	Function key *n*, where *n* is between 0 and 63.
`DL`	The delete line key.
`IL`	The insert line key.
`DC`	The delete character key.
`IC`	The insert character (enter insert mode) key.
`EIC`	The end insert mode key.
`CLEAR`	The clear screen (erase) key.
`EOS`	The clear to end of screen key.
`EOL`	The clear to end of line key.
`SF`	The scroll forward (scroll down) key.
`SR`	The scroll backward (scroll up) key.
`NPAGE`	The next page key.
`PPAGE`	The previous page key.
`STAB`	The set tab key.
`CTAB`	The clear tab key.
`CATAB`	The clear all tabs key.
`ENTER`	The enter (send) key.
`SRESET`	The soft (partial) reset key.
`RESET`	The (hard) reset key.
`PRINT`	The print (copy) key.
`LL`	The home-down key.
`A1`	The upper left keypad key.
`A3`	The upper right keypad key.
`B2`	The center keypad key.
`C1`	The lower left keypad key.
`C3`	The lower right keypad key.
`BTAB`	The back tab key.
`BEG`	The beginning (beg) key.
`CANCEL`	The cancel key.
`CLOSE`	The close key.
`COMMAND`	The command (cmd) key.
`COPY`	The copy key.
`CREATE`	The create key.
`END`	The end key.
`EXIT`	The exit key.
`FIND`	The find key.
`HELP`	The help key.
`MARK`	The mark key.
`MESSAGE`	The message key.
`MOVE`	The move key.
`NEXT`	The next object key.
`OPEN`	The open key.
`OPTIONS`	The options key.
`PREVIOUS`	The previous object key.
`REDO`	The redo key.
`REFERENCE`	The reference (ref) key.

REFRESH	The refresh key.
REPLACE	The replace key.
RESTART	The restart key.
RESUME	The resume key.
SAVE	The save key.
SBEG	The beginning (beg) key (shifted).
SCANCEL	The cancel key (shifted).
SCOMMAND	The command (cmd) key (shifted).
SCOPY	The copy key (shifted).
SCREATE	The create key (shifted).
SDC	The delete character key (shifted).
SDL	The delete line key (shifted).
SELECT	The select key.
SEND	The end key (shifted).
SEOL	The clear to end of line key (shifted).
SEXIT	The exit key (shifted).
SFIND	The find key (shifted).
SHELP	The help key (shifted).
SHOME	The home key (shifted).
SIC	The insert character (enter insert mode) key (shifted).
SLEFT	The cursor left (left arrow) key (shifted).
SMESSAGE	The message key (shifted).
SMOVE	The move key (shifted).
SNEXT	The next object key (shifted).
SOPTIONS	The options key (shifted).
SPREVIOUS	The previous object key (shifted).
SPRINT	The print (copy) key (shifted).
SREDO	The redo key (shifted).
SREPLACE	The replace key (shifted).
SRIGHT	The cursor right (right arrow) key (shifted).
SRSUME	The resume key (shifted).
SSAVE	The save key (shifted).
SSUSPEND	The suspend key (shifted).
SUNDO	The undo key (shifted).
SUSPEND	The suspend key.
UNDO	The undo key.

EXAMPLES

The first and second examples show interactive assignments. The third example shows the assignment of function-value pairs to the EDITREAD environment variable in the C shell. The final example shows the contents of a .editreadrc file.

```
$ erase = ^?<Ctrl-R>

$ intr=^c hist_recall=^z  prompt = OFF<Ctrl-R>

% setenv EDITREAD 'prompt=ON  goto_end = OFF  goto_end_ov = ^a'<NL>

% cat .editreadrc<NL>
erase = ^?
intr = ^c  hist_recall = Esc  prompt = ON
hist_disp = 12  reconfig=f1
```

FILES

`/usr/lib/editreadrc.proto`	Offers a prototype of a `.editreadrc` file.
`.editreadrc`	Is read to enable editread at login for each shell and for other programs using editread. Multiple `.editreadrc` files can reside in your file system.
`.profile`	Is read to initialize the Bourne shell at login; may contain key definitions.
`.login`	Is read to initialize the C shell at login; may contain key definitions.
`/usr/lib/terminfo/?/*`	Provides information about your terminal and keyboard.

SEE ALSO

crash(1M), csh(1), sh(1), stty(1), terminfo(4), term(5).

NOTE

Editread can run out of internal dynamic memory before reaching the number of lines of history requested by the `hist_save` setting. If that happens, editread will, on an on-going basis, retain as many history lines as will fit in its internal memory pool. As each new line is entered, editread will discard as many of the oldest history line(s) as it needs to in order to save the most recent line. This may cause the actual number of saved history lines to vary.

If editread is enabled when an xterm is put into the background, then there is a race condition that can cause the the xterm to have undefined quit and intr stty settings. This appears most often with a csh. The work around to this problem is to either disable editread or to invoke xterm in the following manner.

```
xterm&; sleep 1
```

NAME

term – conventional names for terminals

DESCRIPTION

Terminal names are maintained as part of the shell environment in the environment variable TERM [see sh(1), csh(1), profile(4), and environ(5)]. These names are used by certain commands [for example, man(1), tabs(1), tput(1), and vi(1)] and certain functions [for examples, see curses(3X) and termcap(3X)].

Files under /usr/share/lib/terminfo are used to name terminals and describe their capabilities. These files are in the format described in terminfo(4). To print a source description for a terminal *term*, use the following command:

```
infocmp -I term
```

[See infocmp(1M) for more information.]

Entries in TERMINFO source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the TERMINFO database gives the names by which TERMINFO knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal. This is the preferred one to use to set the environment variable TERM; see profile(4). The last name given should be a long name fully identifying the terminal. All others are understood as synonyms for the terminal name. All names but the last should contain no blanks and should be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. Select a root name based on the particular piece of hardware making up the terminal; for example, for the Data General D463 terminal, d463. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from the set a through z and 0 through 9, make up a basic terminal name. Names should generally be based on original vendors rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes of the hardware, or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, a Data General D463 terminal in 132 column mode is d463-w. The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	d463-w
-am	With automatic margins (usual default)	vt100-am
-nam	Without automatic margins	vt100-nam
-*n*	*n* number of lines on the screen	aaa-60
-na	No arrow keys (leave them in local)	c100-na
-*n*p	*n* number of pages of memory	c100-4p
-rv	Reverse video	att4415-rv

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (e.g., -w), it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the TERMINFO database unique. Terminal entries that are present only for inclusion in other entries via the use= facilities should have a '+' in their name, as in 4415+nl.

The most useful known terminal names are summarized in the table below. For a complete list, enter the following command:

```
ls -C /usr/share/lib/terminfo/?
```

`2621,hp2621`	Hewlett-Packard 2621 series
`2640,hp2640`	Hewlett-Packard 2640 series
`2645,hp2645`	Hewlett-Packard 2645 series
`33,tty33`	AT&T Teletype Model 33 KSR
`35,tty35`	AT&T Teletype Model 35 KSR
`37,tty37`	AT&T Teletype Model 37 KSR
`4014,tek4014`	TEKTRONIX 4014
`40,tty40`	AT&T Teletype Dataspeed 40/2
`43,tty43`	AT&T Teletype Model 43 KSR
`4410,5410`	AT&T 4410/5410 in 80-column mode, version 2
`4410-nfk,5410-nfk`	AT&T 4410/5410 without function keys, version 1
`4410-nsl,5410-nsl`	AT&T 4410/5410 without pln defined
`4410-w,5410-w`	AT&T 4410/5410 in 132-column mode
`4410v1,5410v1`	AT&T 4410/5410 in 80-column mode, version 1
`4410v1-w,5410v1-w`	AT&T 4410/5410 in 132-column mode, version 1
`4415,5420`	AT&T 4415/5420 in 80-column mode
`4415-nl,5420-nl`	AT&T 4415/5420 without changing labels
`4415-rv,5420-rv`	AT&T 4415/5420 80 columns in reverse video
`4415-rv-nl,5420-rv-nl`	AT&T 4415/5420 reverse video w/o changing labels
`4415-w,5420-w`	AT&T 4415/5420 in 132-column mode
`4415-w-nl,5420-w-nl`	AT&T 4415/5420 132-column mode w/o changing labels
`4415-w-rv,5420-w-rv`	AT&T 4415/5420 132 columns in reverse video
`4418,5418`	AT&T 5418 in 80-column mode
`4418-w,5418-w`	AT&T 5418 in 132-column mode
`4420`	AT&T Teletype Model 4420
`4424`	AT&T Teletype Model 4424
`4425,5425`	AT&T 4425/5425
`4425-fk,5425-fk`	AT&T 4425/5425 without function keys
`4425-nl,5425-nl`	AT&T 4425/5425 w/o changing labels, 80-column mode
`4425-w,5425-w`	AT&T 4425/5425 in 132-column mode
`4425-w-fk,5425-w-fk`	AT&T 4425/5425 w/o function keys, 132-column mode
`4425-nl-w,5425-nl-w`	AT&T 4425/5425 w/o changing labels 132-column mode
`4426`	AT&T Teletype Model 4426S
`500,att500`	AT&T-IS 500 terminal
`510a`	AT&T 510a in 80-column mode
`513bct,att513`	AT&T 513 bct terminal
`5420_2`	AT&T 5420 model 2 in 80-column mode
`5420_2-w`	AT&T 5420 model 2 in 132-column mode
`5620,dmd`	AT&T 5620 terminal 88 columns
`5620-24,dmd-24`	AT&T Teletype Model DMD 5620 in a 24x80 layer
`5620-34,dmd-34`	AT&T Teletype Model DMD 5620 in a 34x80 layer
`605x,dg605x,dg`	Generic DG terminal (DG mode)
`6053,dg6053,d2`	DG 6053/D2 terminal (DG mode)
`610,610bct`	AT&T 610 bct terminal in 80-column mode
`610-w,610bct-w`	AT&T 610 bct terminal in 132-column mode
`630,630MTG`	AT&T 630 Multi-Tasking Graphics terminal
`7300,pc7300,unix_pc`	AT&T UNIX PC Model 7300

`d200`	DG Dasher D200 terminal (DG mode)
`d210`	DG Dasher D210 terminal (ANSI mode)
`d211`	DG Dasher D211 terminal (8 bit ANSI mode)
`d211-7b`	DG Dasher D211 terminal (7 bit ANSI mode)
`d214`	DG Dasher D214 terminal (ANSI mode)
`d215`	DG Dasher D215 terminal (8 bit ANSI mode)
`d215-7b`	DG Dasher D215 terminal (7 bit ANSI mode)
`d216`	DG D216/D216+ terminals (VT100 mode)
`d216-unix,d216+`	DG D216+ terminal (DG-UNIX mode)
`d216-unix-25,d216+25`	DG D216+ terminal (25-line DG-UNIX mode)
`d216e-unix,d216e+`	DG D216E+ terminal (DG-UNIX mode)
`d217`	DG D217 terminal (VT100 mode)
`d217-unix`	DG D217 terminal (DG-UNIX mode)
`d217-unix-25`	DG D217 terminal (25-line DG-UNIX mode)
`d220`	DG Dasher D220 color terminal (8 bit ANSI mode)
`d220-7b`	DG Dasher D220 color terminal (7 bit ANSI mode)
`d230c`	DG Dasher D230C color terminal (ANSI-LF mode)
`d400`	DG Dasher D400 terminal (DG mode)
`d410`	DG Dasher D410 terminal (8 bit ANSI mode)
`d410-w`	DG Dasher D410 (8 bit compressed ANSI mode)
`d410-7b`	DG Dasher D410 terminal (7 bit ANSI mode)
`d410-7b-w`	DG Dasher D410 (7 bit compressed ANSI mode)
`d411`	DG Dasher D411 terminal (8 bit ANSI mode)
`d411-w`	DG Dasher D411 (8 bit compressed ANSI mode)
`d411-7b`	DG Dasher D411 terminal (7 bit ANSI mode)
`d411-7b-w`	DG Dasher D411 (7 bit compressed ANSI mode)
`d412`	DG D412/D412+ terminals (VT220/VT320 mode)
`d412-w`	DG D412/D412+ (132-column VT220/VT320 mode)
`d412-unix,d412+,`	DG D412+ terminal (DG-UNIX mode)
`d412-unix-25,d412+25,`	DG D412+ terminal (25-line DG-UNIX mode)
`d412-unix-s,d412+s,`	DG D412+ terminal (DG-UNIX mode with status line)
`d412-unix-sr,d412+sr,`	DG D412+ (DG-UNIX mode with scrolling region)
`d412-unix-w,d412+w,`	DG D412+ terminal (132-column DG-UNIX mode)
`d413`	DG D413 terminal (VT320 mode)
`d413-w`	DG D413 terminal (132-column VT320 mode)
`d413-unix`	DG D413 terminal (DG-UNIX mode)
`d413-unix-25`	DG D413 terminal (25-line DG-UNIX mode)
`d413-unix-s`	DG D413 terminal (DG-UNIX mode with status line)
`d413-unix-sr`	DG D413 (DG-UNIX mode with scrolling region)
`d413-unix-w,`	DG D413 terminal (132-column DG-UNIX mode)
`d450`	DG Dasher D450 terminal (DG mode)
`d460`	DG Dasher D460 terminal (8 bit ANSI mode)
`d460-w`	DG Dasher D460 (8 bit compressed ANSI mode)
`d460-7b`	DG Dasher D460 terminal (7 bit ANSI mode)
`d460-7b-w`	DG Dasher D460 (7 bit compressed ANSI mode)
`d461`	DG Dasher D461 terminal (8 bit ANSI mode)
`d461-w`	DG Dasher D461 (8 bit compressed ANSI mode)
`d461-7b`	DG Dasher D461 terminal (7 bit ANSI mode)
`d461-7b-w`	DG Dasher D461 (7 bit compressed ANSI mode)

`d462`	DG D462/D462+ terminals (VT220/VT320 mode)
`d462-w`	DG D462/D462+ (132-column VT220/VT320 mode)
`d462e`	DG D462E terminal (VT220 mode)
`d462e-w`	DG D462E terminal (132-column VT220 mode)
`d462-unix,d462+`	DG D462+ terminal (DG-UNIX mode)
`d462-unix-25,d462+25`	DG D462+ terminal (25-line DG-UNIX mode)
`d462-unix-s,d462+s`	DG D462+ terminal (DG-UNIX mode with status line)
`d462-unix-sr,d462+sr`	DG D462+ (DG-UNIX mode with scrolling region)
`d462-unix-w,d462+w`	DG D462+ terminal (132-column DG-UNIX mode)
`d463`	DG D463 terminal (VT320 mode)
`d463-w`	DG D463 terminal (132-column VT320 mode)
`d463-unix`	DG D463 terminal (DG-UNIX mode)
`d463-unix-25`	DG D463 terminal (25-line DG-UNIX mode)
`d463-unix-s`	DG D463 terminal (DG-UNIX mode with status line)
`d463-unix-sr`	DG D463 (DG-UNIX mode with scrolling region)
`d463-unix-w`	DG D463 terminal (132-column DG-UNIX mode)
`d470c,d470`	DG Dasher D470C color terminal (8 bit ANSI mode)
`d470c-7b,d470-7b`	DG Dasher D470C color terminal (7 bit ANSI mode)
`d555`	DG Dasher D555 phone terminal (8 bit ANSI mode)
`d555-w`	DG Dasher D555 (8 bit compressed ANSI mode)
`d555-7b`	DG Dasher D555 terminal (7 bit ANSI mode)
`d555-7b-w`	DG Dasher D555 (7 bit compressed ANSI mode)
`d577`	DG Dasher D577 console terminal (8 bit ANSI mode)
`d577-w`	DG Dasher D577 (8 bit compressed ANSI mode)
`d577-7b`	DG Dasher D577 terminal (7 bit ANSI mode)
`d577-7b-w`	DG Dasher D577 (7 bit compressed ANSI mode)
`d578`	DG D578 console terminal (8 bit ANSI mode)
`d578-7b`	DG D578 terminal (7 bit ANSI mode)
`dumb`	generic name for terminals that lack reverse line-feed and other special escape sequences
`hp`	generic Hewlett-Packard terminal
`pt505`	AT&T Personal Terminal 505 (22 lines)
`pt505-24`	AT&T Personal Terminal 505 (24-line mode)
`vt100,vt100-am`	DEC VT100 terminal and compatibles
`vt100-fk`	DEC VT100 compatible with VT220 function keys
`vt100-nam`	DEC VT100 compatible without automatic margins
`vt100-nav`	DEC VT100 compatible without advanced video
`vt100-s,vt100-s-top`	DEC VT100 terminal with status line (top)
`vt100-s-bot`	DEC VT100 terminal with status line (bottom)
`vt100-w,vt100-w-am`	DEC VT100 terminal (132-column mode)
`vt220`	DEC VT220 terminal and compatibles
`vt220-w`	DEC VT220 terminal (132-column mode)
`xterm,xterm-65`	X Window System `xterm` terminal emulator
`xterms,xterm-24`	`xterm` terminal emulator (24 line window)
`xterm-fk,xterm-65-fk`	`xterm` with VT220-style function keys
`xterms-fk,xterm-24-fk`	`xterm` with function keys (24 line window)
`xterm-dg,xterm-65-dg`	`xterm` on a DG AViiON workstation keyboard
`xterms-dg,xterm-24-dg`	`xterm` on a DG AViiON (24 line window)

Commands whose behavior depends on the type of terminal should accept arguments of the form `-T`*term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable `TERM`, which, in turn, should contain *term*.

FILES

`/usr/share/lib/terminfo/?/*`	compiled terminal descriptions
`/usr/src/cmd/terminfo/*.ti`	source terminal descriptions

SEE ALSO

`csh`(1), `man`(1), `sh`(1), `stty`(1), `tabs`(1), `tput`(1), `vi`(1), `infocmp`(1M), `curses`(3X), `termcap`(3X). `profile`(4), `terminfo`(4), `editread`(5), `environ`(5), `termcap`(5), `ttycompat`(7).

NOTES

Not all programs follow the above naming conventions.

The following line discipline `stty`(1) settings are recommended for Data General terminals:

`nl -tabs -istrip`	for eight-bit ANSI and DG modes.
`nl -tabs`	for seven-bit ANSI and DG modes.
`-nl tabs`	for VT100 mode.
`-nl tabs -istrip`	for VT220, VT320, and DG-UNIX modes.
`-nl -tabs -istrip`	for eight-bit ANSI-LF mode.
`-nl -tabs`	for seven-bit ANSI-LF mode.

In addition to supporting the Data General D216 terminal in its VT100 mode, the `d216` entry supports the D216+ terminal in its VT100 mode. The `vt100` entry can also be used with these terminals; it makes the numeric keypad work as an additional set of function keys (but of course prevents the keypad from being used for data entry).

In addition to supporting the Data General D412 and D462 terminals in their VT220 modes, the `d412` and `d462` entries support the D412+ and D462+ terminals in their VT320 modes. The `vt220` entry can also be used with these terminals; it maps the function keys so that they are more suitable for a touch-typist familiar with a VT220/VT320 keyboard (but then the function key labels do not match the keys' effects).

The default entries for the `xterm` terminal emulator, `xterm` and `xterms`, are compatible with the `vt100` entry. They make the numeric keypad work as an additional set of function keys, but do not take advantage of the large number of function keys found on modern keyboards (such as the PC-style keyboard used on AViiON workstations). Additional entries, `xterm-fk` and `xterms-fk`, are provided to enable use of these function keys and to allow data entry from the keypad; they are usable with any keyboard that has modern function keys. Two more entries, `xterm-dg` and `xterms-dg`, are provided; they are similar to `xterm-fk` and `xterms-fk`, but match the keys and layout of an AViiON keyboard exactly.

The `d230c` terminal entry supports both seven-bit and eight-bit character sizes in a single entry. Separate entries are required for other DG terminals in ANSI mode, one to support each character size.

Support for Data General terminals in DG mode is limited because this mode does not in general work well on UNIX systems. The problem is that the ANSI standard backspace character (Ctrl-H) causes the cursor to "home" to the upper-left corner of a DG terminal in DG mode. This behavior is incompatible with the way characters and lines are erased from the screen by the `tty`(7) driver; the cursor will go to the home position whenever the erase key is pressed. This problem can be partially avoided by using `editread`(5) or by using `stty`(1) to turn off echoing for the ERASE and KILL characters.

The 6053, D200, D400, and D450 terminals provide only DG mode.

International Notes

The Data General D216, D216E, D216+, and D216E+ terminals do not support eight-bit characters in their VT100 modes.

The `d412`, `d413`, `d462`, and `d463` entries support eight-bit characters but require that the terminal be set for "seven-bit control" sequences.

Data General terminals in (eight-bit) ANSI, DG-UNIX, and DG modes use the proprietary "DG International" code set, instead of an standard code set such as ISO 8859-1. Data General terminals in VT220 and VT320 modes use the "DEC Multinational" code set, which is a compatible subset of ISO 8859-1.

The 6053, D200, D210, D214, D400, and D450 terminals do not support eight-bit characters at all.

End of Chapter

Index

Note: Boldfaced page numbers (e.g., **1-5**) indicate definitions of terms or other key information.

Related Documents

The following list of related manuals gives titles of Data General manuals followed by nine-digit numbers used for ordering. You can order any of these manuals via mail or telephone (see the TIPS Order Form in the back of this manual).

For a complete list of AViiON® and DG/UX™ manuals, see the *Guide to AViiON® and DG/UX™ Documentation* (069-701085). The on-line version of this manual found in **/usr/release/doc_guide** contains the most current list.

Data General Software Manuals

User's Manuals

Using the DG/UX™ Editors
Describes the text editors **vi** and **ed**, the batch editor **sed**, and the command line editor **editread**. Ordering Number — 069-701036

Using the DG/UX™ System
Describes the DG/UX system and its major features, including the C and Bourne shells, typical user commands, the file system, and communications facilities such as **mailx**. Ordering Number — 069-701035

Using TCP/IP on the DG/UX™ System
Introduces Data General's implementation of the TCP/IP family of protocols and describes how to use the package. Ordering Number — 093-701023

Installation and Administration Manuals

Managing ONC™/NFS® and Its Facilities on the DG/UX™ System
Explains how to manage and use the DG/UX ONC™/NFS® product. Contains information on the Network File System (NFS), the Network Information Service (NIS), Remote Procedure Calls (RPC), and External Data Representation (XDR). Ordering Number — 093-701049

System Manager's Reference for the DG/UX™ System
Contains an alphabetical listing of DG/UX, TCP/IP, and ONC/NFS manual pages for commands relating to system administration or operation. Ordering Number — 093-701050

Programming Manuals

Porting and Developing Applications on the DG/UX™ System
A compendium of useful information for experienced programmers developing or porting applications to the DG/UX™ system. It includes information on how to: set up your environment, use the software development tools, compile and link programs, port to the windowing environment, and build BCS applications. It also describes available debuggers and the various industry standards the DG/UX system supports. Ordering Number — 069-701059

Programmer's Reference for the DG/UX™ System, (Volume 1)
Alphabetical listing of manual pages for DG/UX programming commands and system calls. This is part of a three-volume set. Ordering Number — 093-701055

Programmer's Reference for the DG/UX™ System, (Volume 2)
Alphabetical listing of manual pages for DG/UX and ONC/NFS subroutines and libraries. This is part of a three-volume set. Ordering Number — 093-701056

Programmer's Reference for the DG/UX™ System, (Volume 3)
Alphabetical listing of manual pages for DG/UX, TCP/IP, and ONC/NFS file formats, miscellaneous features, and networking protocols. Part of a three-volume set, this volume contains the table of contents and index (**contents** (0) and **index** (0)) for man pages. Ordering Number — 093-701102

End of Related Documents

TIPS ORDERING PROCEDURES

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a) MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

   Send your order form with payment to:

   Data General Corporation
   ATTN: Educational Services/TIPS G155
   4400 Computer Drive
   Westboro, MA 01581–9973

   b) TELEPHONE – Call TIPS at (508) 870–1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a) Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.
   b) Check or Money Order – Make payable to Data General Corporation.
   c) Credit Card – A minimum order of $20 is required for Mastercard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	**Shipping & Handling Charge**
1–4 Units	$5.00
5–10 Units	$8.00
11–40 Units	$10.00
41–200 Units	$30.00
Over 200 Units	$100.00

If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	**Discount**
$1–$149.99	0%
$150–$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

6. Allow at least two weeks for delivery.

RETURNS

7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870–1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	**SHIP TO:** (No P.O. Boxes - Complete Only If Different Address)
COMPANY NAME__________	COMPANY NAME__________
ATTN:__________	ATTN:__________
ADDRESS__________	ADDRESS (NO PO BOXES)__________
CITY__________	CITY__________
STATE ______ ZIP______	STATE______ZIP______

Priority Code __________ (See label on back of catalog)

Authorized Signature of Buyer (Agrees to terms & conditions on reverse side)	Title	Date	Phone (Area Code)	Ext.

ORDER #	QTY	DESCRIPTION	UNIT PRICE	TOTAL PRICE

A

☐ UPS	ADD
1-4 Items	$ 5.00
5-10 Items	$ 8.00
11-40 Items	$ 10.00
41-200 Items	$ 30.00
200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0 – $149.99	0%
$150 – $499.99	10%
Over $500.00	20%

Tax Exempt # or Sales Tax (if applicable)

____

ORDER TOTAL	
Less Discount See B	–
SUB TOTAL	
Your local* sales tax	+
Shipping and handling – See A	+
TOTAL – See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is__________. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ Expiration Date ☐☐☐☐

____

Authorized Signature

(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508–870–1600.

Form 702
Rev. 8/87

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE

TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. CUSTOMER CERTIFICATION
Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub–licensee of the software which is the subject matter of the publication(s) ordered hereunder.

2. TAXES
Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. DATA AND PROPRIETARY RIGHTS
Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

4. LIMITED MEDIA WARRANTY
DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY
EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY
A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. GENERAL
A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts, excluding its conflict of law rules. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)
Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision–locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

Cut here and insert in binder spine pocket