UNIX
User’s Manual
UNIX
User's Manual

Release 3.0

T. A. Dolotta
S. B. Olsson
A. G. Petruccelli
Editors

June 1980

Not for use or disclosure outside the
Bell System except under written agreement.

Laboratory 364
Bell Telephone Laboratories, Incorporated
Murray Hill, NJ 07974
Copyright © 1980 Bell Telephone Laboratories, Inc.

UNIX is a Trademark of Bell Telephone Laboratories, Inc.

This manual was set on an AUTOLOGIC, Inc. APS-5 phototypesetter driven by the TROFF formatter operating under the UNIX system.
ACKNOWLEDGEMENTS


*Murray Hill, New Jersey*

T.A.D.
S.B.O.
A.G.P.
INTRODUCTION

This manual describes the features of UNIX. It provides neither a general overview of UNIX (for that, see “The UNIX Time-Sharing System,” BSTJ, Vol. 57, No. 6, Part 2, pp. 1905-29, by D. M. Ritchie and K. Thompson), nor details of the implementation of the system (see “UNIX Implementation,” BSTJ, same issue, pp. 1931-46).

Not all commands, features, and facilities described in this manual are available in every UNIX system; for example, yacc(1) is usually not available in a UNIX system running on a PDP-11/23. When in doubt, consult your system's administrator.

This manual is divided into eight sections, some containing inter-filed sub-classes:

1. Commands and Application Programs:
   1. General-Purpose Commands.
   1C. Communications Commands.
   1G. Graphics Commands.
   1M. System Maintenance Commands.
2. System Calls.
3. Subroutines:
   3C. C and Assembler Library Routines.
   3M. Mathematical Library Routines.
   3S. Standard I/O Library Routines.
   3X. Miscellaneous Routines.
4. Special Files.
5. File Formats.
6. Games.
7. Miscellaneous Facilities.

Section 1 (Commands and Application Programs) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory /bin (for binary programs). Some programs also reside in /usr/bin, to save space in /bin. These directories are searched automatically by the command interpreter called the shell. Sub-class 1C contains communication programs such as cu, dpr, fget, etc. These entries may differ from system to system. Sub-class 1M contains system maintenance programs such as fsck, mkfs, etc., which generally reside in the directory /etc; these commands are not intended for use by the ordinary user due to their privileged nature. Some UNIX systems have a directory called /usr/sbin, containing local commands.

Section 2 (System Calls) describes the entries into the UNIX supervisor, including the C language interface.

Section 3 (Subroutines) describes the available subroutines. Their binary versions reside in various system libraries in the directories /lib and /usr/lib. See intro(3) for descriptions of these libraries and the files in which they are stored.

Section 4 (Special Files) discusses the characteristics of each system file that actually refers to an input/output device. The names in this section generally refer to the Digital Equipment Corporation's device names for the hardware, rather than to the names of the special files themselves.

Section 5 (File Formats) documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in a.out(5). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language struct declarations corresponding to these formats can be found in the directories /usr/include and /usr/include/sys.
Section 6 (Games) describes the games and educational programs that, as a rule, reside in the directory /usr/games.

Section 7 (Miscellaneous Facilities) contains a variety of things. Included are descriptions of character sets, macro packages, etc.

Section 8 (System Maintenance Procedures) discusses crash recovery and boot procedures, etc. Information in this section is not of great interest to most users.

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

The NAME part gives the name(s) of the entry and briefly states its purpose.

The SYNOPSIS part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (Commands):

**Boldface** strings are literals and are to be typed just as they appear.

*Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).

Square brackets [ ] around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Ellipses ... are used to show that the previous argument prototype may be repeated.

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

The DESCRIPTION part discusses the subject at hand.

The EXAMPLE(S) part gives example(s) of usage, where appropriate.

The FILES part gives the file names that are built into the program.

The SEE ALSO part gives pointers to related information.

The DIAGNOSTICS part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The WARNINGS part points out potential pitfalls.

The BUGS part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permuted index derived from that table precede Section 1. On each index line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

On most systems, all entries are available on-line via the man(1) command, q.v.
HOW TO GET STARTED

This discussion provides the basic information you need to get started on UNIX: how to log in and log out, how to communicate through your terminal, and how to run a program. (See UNIX for Beginners by B. W. Kernighan for a more complete introduction to the system.)

Logging in. You must dial up UNIX from an appropriate terminal. UNIX supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained (together with the telephone number(s) of your UNIX system) from the administrator of your system. Common terminal speeds are 10, 15, 30, and 120 characters per second (110, 150, 300, and 1,200 baud); occasionally, speeds of 240, 480, and 960 characters per second (2,400, 4,800, and 9,600 baud) are also available. On some UNIX systems, there are separate telephone numbers for each available terminal speed, while on other systems several speeds may be served by a single telephone number. In the latter case, there is one "preferred" speed; if you dial in from a terminal set to a different speed, you will be greeted by a string of meaningless characters (the login: message at the wrong speed). Keep hitting the "break" or "attention" key until the login: message appears. Hard-wired terminals usually are set to the correct speed.

Most terminals have a speed switch that should be set to the appropriate speed and a half-/full-duplex switch that should be set to full-duplex. When a connection (at the speed of the terminal) has been established, the system types login: and you then type your user name followed by the "return" key. If you have a password (and you should!), the system asks for it, but does not print ("echo") it on the terminal. After you have logged in, the "return", "new-line", and "line-feed" keys will give exactly the same result.

It is important that you type your login name in lower case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and that you mean all subsequent upper-case input to be treated as lower case. When you have logged in successfully, the shell will type a $ to you. (The shell is described below under How to run a program.)

For more information, consult login(1) and getty(8), which discuss the login sequence in more detail, and stty(1), which tells you how to describe the characteristics of your terminal to the system (profile(5) explains how to accomplish this last task automatically every time you log in).

Logging out. There are two ways to log out:

1. You can simply hang up the phone.
2. You can log out by typing an end-of-file indication (ASCII EOT character, usually typed as "control-d") to the shell. The shell will terminate and the login: message will appear again.

How to communicate through your terminal. When you type to UNIX, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a "return" (or "new-line"), as described above in Logging in.

UNIX terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have interspersed in it the input characters. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away all the saved characters.
On an input line from a terminal, the character \@ "kills" all the characters typed before it. The character \# erases the last character typed. Successive uses of \# will erase characters back to, but not beyond, the beginning of the line; \@ and \# can be typed as themselves by preceding them with \ (thus, to erase a \, you need two \#s). These default erase and kill characters can be changed; see stty(1).

The ASCII DC3 (control-s) character can be used to temporarily stop output. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when a DC1 (control-q) or a second DC3 (or any other character, for that matter) is typed. The DC1 and DC3 characters are not passed to any other program when used in this manner.

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

The quit signal is generated by typing the ASCII FS character. It not only causes a running program to terminate, but also generates a file with the "core image" of the terminated process. Quit is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent as to whether you have a terminal with the "new-line" function, or whether it must be simulated with a "carriage-return" and "line-feed" pair. In the latter case, all input "carriage-return" characters are changed to "line-feed" characters (the standard line delimiter), and a "carriage-return" and "line-feed" pair is echoed to the terminal. If you get into the wrong mode, the stty(1) command will rescue you.

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

How to run a program. When you have successfully logged into UNIX, a program called the shell is listening to your terminal. The shell reads the lines you type, splits them into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see The current directory below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

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

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

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

Path names. To refer to files not in the current directory, you must use a path
name. Full path names begin with /, which is the name of the root directory of the
whole file system. After the slash comes the name of each directory containing the
next sub-directory (followed by a /), until finally the file name is reached (e.g.,
/usr/ae/filex refers to file filex in directory ae, while ae is itself a subdirectory of
usr; usr springs directly from the root directory). See intro(2) for a formal
deinition of path name.

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

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

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

Writing a program. To enter the text of a source program into a UNIX file, use
ed(1). The four principal languages available under UNIX are C (see cc(1)), For-
tran (see f77(1)), bs (a compiler/interpreter in the spirit of Basic, see bs(1)), and
assembly language (see as(1)). After the program text has been entered with the
editor and written into a file (whose name has the appropriate suffix), you can give
the name of that file to the appropriate language processor as an argument. Nor-
mally, the output of the language processor will be left in a file in the current direc-
tory named a.out (if that output is precious, use mv(1) to give it a less vulnerable
name). If the program is written in assembly language, you will probably need to
load with it library subroutines (see ld(1)). Fortran and C call the loader automati-
cally; programs written in bs(1) are interpreted and, therefore, do not need to be
loaded.

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

If any execution (run-time) errors occur, you will need adb(1) to examine the
remains of your program. On the VAX-11/780, a second debugger sdb(1), which
allows you to step through C statements rather than assembler instructions, is
available.

Your programs can receive arguments from the command line just as system pro-
grams do; see exec(2).

Text processing. Almost all text is entered through the editor ed(1). The com-
mands most often used to write text on a terminal are cat(1), pr(1), and nroff(1).
The cat(1) command simply dumps ASCII text on the terminal, with no processing
at all. The pr(1) command paginates the text, supplies headings, and has a facility
for multi-column output. Nroff(1) is an elaborate text formatting program, and
requires careful forethought in entering both the text and the formatting commands
into the input file; it produces output on a typewriter-like terminal. Troff(1) is very
similar to *nroff*(1), but produces its output on a phototypesetter (it was used to typeset this manual). There are several “macro” packages (especially the so-called *mm* package) that significantly ease the effort required to use *nroff*(1) and *troff*(1); Section 7 entries for these packages indicate where you can find their detailed descriptions.

**Surprises.** Certain commands provide inter-user communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you. To communicate with another user currently logged in, *write*(1) is used; *mail*(1) will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

When you log in, a message-of-the-day may greet you before the first $. 
# TABLE OF CONTENTS

1. Commands and Application Programs

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intro</td>
<td>introduction to commands and application programs</td>
</tr>
<tr>
<td>300</td>
<td>handle special functions of DASI 300 and 300s terminals</td>
</tr>
<tr>
<td>4014</td>
<td>preparer for the Tektronix 4014 terminal</td>
</tr>
<tr>
<td>450</td>
<td>handle special functions of the DASI 450 terminal</td>
</tr>
<tr>
<td>acct</td>
<td>overview of accounting and miscellaneous accounting commands</td>
</tr>
<tr>
<td>accctcms</td>
<td>command summary from per-process accounting records</td>
</tr>
<tr>
<td>acctcom</td>
<td>search and print process accounting file(s)</td>
</tr>
<tr>
<td>acctcon</td>
<td>connect-time accounting</td>
</tr>
<tr>
<td>acctmerg</td>
<td>merge or add total accounting files</td>
</tr>
<tr>
<td>acctprc</td>
<td>process accounting</td>
</tr>
<tr>
<td>acctsh</td>
<td>shell procedures for accounting</td>
</tr>
<tr>
<td>adb</td>
<td>create and administer SCCS files</td>
</tr>
<tr>
<td>admin</td>
<td>archive and library maintainer</td>
</tr>
<tr>
<td>ar</td>
<td>convert archive files from PDP-11 to VAX-11/780 format</td>
</tr>
<tr>
<td>arcv</td>
<td>assembler for PDP-11</td>
</tr>
<tr>
<td>as.pdp</td>
<td>assembler for VAX-11/780</td>
</tr>
<tr>
<td>as.vax</td>
<td>a compiler/interpreter for modest-sized programs</td>
</tr>
<tr>
<td>awk</td>
<td>pattern scanning and processing language</td>
</tr>
<tr>
<td>banner</td>
<td>make posters</td>
</tr>
<tr>
<td>basename</td>
<td>deliver portions of path names</td>
</tr>
<tr>
<td>bc</td>
<td>arbitrary-precision arithmetic language</td>
</tr>
<tr>
<td>bcopy</td>
<td>interactive block copy</td>
</tr>
<tr>
<td>bdiff</td>
<td>big diff</td>
</tr>
<tr>
<td>bfs</td>
<td>big file scanner</td>
</tr>
<tr>
<td>bs</td>
<td>a compiler/interpreter for modest-sized programs</td>
</tr>
<tr>
<td>cal</td>
<td>print calendar</td>
</tr>
<tr>
<td>calendar</td>
<td>reminder service</td>
</tr>
<tr>
<td>cat</td>
<td>concatenate and print files</td>
</tr>
<tr>
<td>cb</td>
<td>C program beautifier</td>
</tr>
<tr>
<td>cc</td>
<td>C compiler</td>
</tr>
<tr>
<td>cd</td>
<td>change working directory</td>
</tr>
<tr>
<td>cdc</td>
<td>change the delta commentary of an SCCS delta</td>
</tr>
<tr>
<td>chmod</td>
<td>change mode</td>
</tr>
<tr>
<td>chown</td>
<td>change owner or group</td>
</tr>
<tr>
<td>chroot</td>
<td>change root directory for a command</td>
</tr>
<tr>
<td>cli</td>
<td>clear i-node</td>
</tr>
<tr>
<td>cmp</td>
<td>compare two files</td>
</tr>
<tr>
<td>col</td>
<td>filter reverse line-feeds</td>
</tr>
<tr>
<td>comb</td>
<td>combine SCCS deltas</td>
</tr>
<tr>
<td>comm</td>
<td>select or reject lines common to two sorted files</td>
</tr>
<tr>
<td>config</td>
<td>configure a UNIX system</td>
</tr>
<tr>
<td>cp</td>
<td>copy, link or move files</td>
</tr>
<tr>
<td>cpio</td>
<td>copy file archives in and out</td>
</tr>
<tr>
<td>crash</td>
<td>examine system images</td>
</tr>
<tr>
<td>cref</td>
<td>make cross-reference listing</td>
</tr>
<tr>
<td>cron</td>
<td>clock daemon</td>
</tr>
<tr>
<td>crypt</td>
<td>encode/decode</td>
</tr>
<tr>
<td>csplit</td>
<td>context split</td>
</tr>
<tr>
<td>ct</td>
<td>call terminal</td>
</tr>
<tr>
<td>cu</td>
<td>call another UNIX system</td>
</tr>
<tr>
<td>cut</td>
<td>cut out selected fields of each line of a file</td>
</tr>
<tr>
<td>cw</td>
<td>prepare constant-width text for troff</td>
</tr>
<tr>
<td>date</td>
<td>print and set the date</td>
</tr>
</tbody>
</table>

-1-
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dc</td>
<td>desk calculator</td>
</tr>
<tr>
<td>dd</td>
<td>convert and copy a file</td>
</tr>
<tr>
<td>delta</td>
<td>make a delta (change) to an SCCS file</td>
</tr>
<tr>
<td>deroff</td>
<td>remove nroff/troff, tbl, and eqn constructs</td>
</tr>
<tr>
<td>devnm</td>
<td>device name</td>
</tr>
<tr>
<td>df</td>
<td>report number of free disk blocks</td>
</tr>
<tr>
<td>diff</td>
<td>differential file comparator</td>
</tr>
<tr>
<td>diff3</td>
<td>3-way differential file comparison</td>
</tr>
<tr>
<td>diff/mlk</td>
<td>mark differences between files</td>
</tr>
<tr>
<td>dircmp</td>
<td>directory comparison</td>
</tr>
<tr>
<td>dpd</td>
<td>HONEYWELL sending daemons, line printer daemon</td>
</tr>
<tr>
<td>dpr</td>
<td>off-line print</td>
</tr>
<tr>
<td>du</td>
<td>summarize disk usage</td>
</tr>
<tr>
<td>dump</td>
<td>incremental file system dump</td>
</tr>
<tr>
<td>echo</td>
<td>echo arguments</td>
</tr>
<tr>
<td>ed</td>
<td>text editor</td>
</tr>
<tr>
<td>efl</td>
<td>Extended Fortran Language</td>
</tr>
<tr>
<td>env</td>
<td>set environment for command execution</td>
</tr>
<tr>
<td>eqn</td>
<td>format mathematical text for nroff or troff</td>
</tr>
<tr>
<td>errdead</td>
<td>extract error records from dump</td>
</tr>
<tr>
<td>errdem</td>
<td>error-logging daemon</td>
</tr>
<tr>
<td>erprt</td>
<td>process a report of logged errors</td>
</tr>
<tr>
<td>errstop</td>
<td>terminate the error-logging daemon</td>
</tr>
<tr>
<td>expr</td>
<td>evaluate arguments as an expression</td>
</tr>
<tr>
<td>f77</td>
<td>Fortran 77 compiler</td>
</tr>
<tr>
<td>factor</td>
<td>factor a number, generate large primes</td>
</tr>
<tr>
<td>fget</td>
<td>retrieve files from the HONEYWELL 6000</td>
</tr>
<tr>
<td>fget.demon</td>
<td>file retrieval daemons</td>
</tr>
<tr>
<td>file</td>
<td>determine file type</td>
</tr>
<tr>
<td>find</td>
<td>find files</td>
</tr>
<tr>
<td>fsck</td>
<td>file system consistency check and interactive repair</td>
</tr>
<tr>
<td>fscv</td>
<td>convert files between PDP-11 and VAX-11/780 systems</td>
</tr>
<tr>
<td>fsdb</td>
<td>file system debugger</td>
</tr>
<tr>
<td>fsend</td>
<td>send files to the HONEYWELL 6000</td>
</tr>
<tr>
<td>fwtmp</td>
<td>manipulate wtmp records</td>
</tr>
<tr>
<td>gcat</td>
<td>send phototypesetter output to the HONEYWELL 6000</td>
</tr>
<tr>
<td>gcosmail</td>
<td>send mail to HIS user</td>
</tr>
<tr>
<td>gdev</td>
<td>graphical device routines and filters</td>
</tr>
<tr>
<td>ged</td>
<td>graphical editor</td>
</tr>
<tr>
<td>get</td>
<td>get a version of an SCCS file</td>
</tr>
<tr>
<td>getopt</td>
<td>parse command options</td>
</tr>
<tr>
<td>graph</td>
<td>draw a graph</td>
</tr>
<tr>
<td>graphics</td>
<td>access graphical and numerical commands</td>
</tr>
<tr>
<td>greek</td>
<td>select terminal filter</td>
</tr>
<tr>
<td>grep</td>
<td>search a file for a pattern</td>
</tr>
<tr>
<td>gutil</td>
<td>graphical utilities</td>
</tr>
<tr>
<td>help</td>
<td>ask for help</td>
</tr>
<tr>
<td>hp</td>
<td>handle special functions of HP 2640 and 2621-series terminals</td>
</tr>
<tr>
<td>hyphen</td>
<td>find hyphenated words</td>
</tr>
<tr>
<td>id</td>
<td>print user and group IDs and names</td>
</tr>
<tr>
<td>install</td>
<td>install commands</td>
</tr>
<tr>
<td>join</td>
<td>relational database operator</td>
</tr>
<tr>
<td>kas</td>
<td>assembler for the KMC11 microprocessor</td>
</tr>
<tr>
<td>kill</td>
<td>terminate a process</td>
</tr>
<tr>
<td>kun</td>
<td>un-assembler for the KMC11/DMC11 microprocessor</td>
</tr>
<tr>
<td>ld</td>
<td>link editor</td>
</tr>
</tbody>
</table>
**Table of Contents**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lex</td>
<td>generate programs for simple lexical tasks</td>
</tr>
<tr>
<td>line</td>
<td>read one line</td>
</tr>
<tr>
<td>link</td>
<td>exercise link and unlink system calls</td>
</tr>
<tr>
<td>lint</td>
<td>a C program checker</td>
</tr>
<tr>
<td>login</td>
<td>sign on</td>
</tr>
<tr>
<td>logname</td>
<td>get login name</td>
</tr>
<tr>
<td>lorder</td>
<td>find ordering relation for an object library</td>
</tr>
<tr>
<td>lpr</td>
<td>line printer spooler</td>
</tr>
<tr>
<td>ls</td>
<td>list contents of directories</td>
</tr>
<tr>
<td>m4</td>
<td>macro processor</td>
</tr>
<tr>
<td>mail</td>
<td>send mail to users or read mail</td>
</tr>
<tr>
<td>make</td>
<td>maintain, update, and regenerate groups of programs</td>
</tr>
<tr>
<td>man</td>
<td>print entries in this manual</td>
</tr>
<tr>
<td>mesg</td>
<td>permit or deny messages</td>
</tr>
<tr>
<td>mkdir</td>
<td>make a directory</td>
</tr>
<tr>
<td>mkfs</td>
<td>construct a file system</td>
</tr>
<tr>
<td>mknod</td>
<td>build special file</td>
</tr>
<tr>
<td>mm</td>
<td>print out documents formatted with the MM macros</td>
</tr>
<tr>
<td>mmchek</td>
<td>check usage of mm macros and eqn delimiters</td>
</tr>
<tr>
<td>mmt</td>
<td>typeset documents, view graphs, and slides</td>
</tr>
<tr>
<td>mount</td>
<td>mount and dismount file system</td>
</tr>
<tr>
<td>mvdir</td>
<td>move a directory</td>
</tr>
<tr>
<td>ncheck</td>
<td>generate names from i-numbers</td>
</tr>
<tr>
<td>newgrp</td>
<td>log in to a new group</td>
</tr>
<tr>
<td>news</td>
<td>print news items</td>
</tr>
<tr>
<td>nice</td>
<td>run a command at low priority</td>
</tr>
<tr>
<td>nl</td>
<td>line numbering filter</td>
</tr>
<tr>
<td>nm</td>
<td>print name list</td>
</tr>
<tr>
<td>nohup</td>
<td>run a command immune to hangups and quits</td>
</tr>
<tr>
<td>od</td>
<td>octal dump</td>
</tr>
<tr>
<td>orjестат</td>
<td>RJE status and enquiries</td>
</tr>
<tr>
<td>pack</td>
<td>compress and expand files</td>
</tr>
<tr>
<td>passwd</td>
<td>change login password</td>
</tr>
<tr>
<td>paste</td>
<td>merge same lines of several files or subsequent lines of one file</td>
</tr>
<tr>
<td>pr</td>
<td>print files</td>
</tr>
<tr>
<td>prof</td>
<td>display profile data</td>
</tr>
<tr>
<td>profiler</td>
<td>operating system profiler</td>
</tr>
<tr>
<td>prs</td>
<td>print an SCCS file</td>
</tr>
<tr>
<td>ps</td>
<td>report process status</td>
</tr>
<tr>
<td>ptx</td>
<td>permuted index</td>
</tr>
<tr>
<td>pwck</td>
<td>password/group file checkers</td>
</tr>
<tr>
<td>pwd</td>
<td>working directory name</td>
</tr>
<tr>
<td>ratfor</td>
<td>rational Fortran dialect</td>
</tr>
<tr>
<td>reform</td>
<td>reformat text file</td>
</tr>
<tr>
<td>regcmp</td>
<td>regular expression compile</td>
</tr>
<tr>
<td>restor</td>
<td>incremental file system restore</td>
</tr>
<tr>
<td>rjestat</td>
<td>RJE status report and interactive status console</td>
</tr>
<tr>
<td>rm</td>
<td>remove files or directories</td>
</tr>
<tr>
<td>rmdel</td>
<td>remove a delta from an SCCS file</td>
</tr>
<tr>
<td>rsh</td>
<td>restricted shell (command interpreter)</td>
</tr>
<tr>
<td>runacct</td>
<td>run daily accounting</td>
</tr>
<tr>
<td>sact</td>
<td>print current SCCS file editing activity</td>
</tr>
<tr>
<td>sag</td>
<td>system activity graph</td>
</tr>
<tr>
<td>scc</td>
<td>C compiler for stand-alone programs</td>
</tr>
<tr>
<td>sccsdiff</td>
<td>compare two versions of an SCCS file</td>
</tr>
<tr>
<td>sdb</td>
<td>symbolic debugger</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdiff</td>
<td>side-by-side difference program</td>
</tr>
<tr>
<td>sed</td>
<td>stream editor</td>
</tr>
<tr>
<td>send</td>
<td>gather files and/or submit RJE jobs</td>
</tr>
<tr>
<td>setmnt</td>
<td>establish mnttab table</td>
</tr>
<tr>
<td>sh</td>
<td>shell, the standard command programming language</td>
</tr>
<tr>
<td>shutdown</td>
<td>terminate all processing</td>
</tr>
<tr>
<td>size</td>
<td>size of an object file</td>
</tr>
<tr>
<td>sleep</td>
<td>suspend execution for an interval</td>
</tr>
<tr>
<td>sno</td>
<td>SNOBOL interpreter</td>
</tr>
<tr>
<td>sort</td>
<td>sort and/or merge files</td>
</tr>
<tr>
<td>spell</td>
<td>find spelling errors</td>
</tr>
<tr>
<td>spline</td>
<td>interpolate smooth curve</td>
</tr>
<tr>
<td>split</td>
<td>split a file into pieces</td>
</tr>
<tr>
<td>st</td>
<td>synchronous terminal control</td>
</tr>
<tr>
<td>stat</td>
<td>statistical network useful with graphical commands</td>
</tr>
<tr>
<td>strip</td>
<td>remove symbols and relocation bits</td>
</tr>
<tr>
<td>stty</td>
<td>set the options for a terminal</td>
</tr>
<tr>
<td>su</td>
<td>become super-user or another user</td>
</tr>
<tr>
<td>sum</td>
<td>sum and count blocks in a file</td>
</tr>
<tr>
<td>sync</td>
<td>update the super block</td>
</tr>
<tr>
<td>sysdef</td>
<td>system definition</td>
</tr>
<tr>
<td>tabs</td>
<td>set tabs on a terminal</td>
</tr>
<tr>
<td>tail</td>
<td>deliver the last part of a file</td>
</tr>
<tr>
<td>tar</td>
<td>tape file archiver</td>
</tr>
<tr>
<td>tbl</td>
<td>format tables for nroff or troff</td>
</tr>
<tr>
<td>tc</td>
<td>phototypesetter simulator</td>
</tr>
<tr>
<td>tee</td>
<td>pipe fitting</td>
</tr>
<tr>
<td>test</td>
<td>condition evaluation command</td>
</tr>
<tr>
<td>time</td>
<td>time a command</td>
</tr>
<tr>
<td>timex</td>
<td>time a command and generate a system activity report</td>
</tr>
<tr>
<td>toc</td>
<td>graphical table of contents routines</td>
</tr>
<tr>
<td>touch</td>
<td>update access and modification times of a file</td>
</tr>
<tr>
<td>tp</td>
<td>manipulate tape archive</td>
</tr>
<tr>
<td>tplot</td>
<td>graphics filters</td>
</tr>
<tr>
<td>tr</td>
<td>translate characters</td>
</tr>
<tr>
<td>troff</td>
<td>typeset or format text</td>
</tr>
<tr>
<td>true</td>
<td>provide truth values</td>
</tr>
<tr>
<td>tsort</td>
<td>topological sort</td>
</tr>
<tr>
<td>tty</td>
<td>get the terminal's name</td>
</tr>
<tr>
<td>typo</td>
<td>find possible typographical errors</td>
</tr>
<tr>
<td>umask</td>
<td>set file-creation mode mask</td>
</tr>
<tr>
<td>uname</td>
<td>print name of current UNIX</td>
</tr>
<tr>
<td>unget</td>
<td>undo a previous get of an SCCS file</td>
</tr>
<tr>
<td>uniq</td>
<td>report repeated lines in a file</td>
</tr>
<tr>
<td>units</td>
<td>conversion program</td>
</tr>
<tr>
<td>uuclean</td>
<td>uucp spool directory clean-up</td>
</tr>
<tr>
<td>uucp</td>
<td>unix to unix copy</td>
</tr>
<tr>
<td>uustat</td>
<td>uucp status inquiry and job control</td>
</tr>
<tr>
<td>uusub</td>
<td>monitor uucp network</td>
</tr>
<tr>
<td>uuto</td>
<td>public UNIX-to-UNIX file copy</td>
</tr>
<tr>
<td>uux</td>
<td>unix to unix command execution</td>
</tr>
<tr>
<td>val</td>
<td>validate SCCS file</td>
</tr>
<tr>
<td>vc</td>
<td>version control</td>
</tr>
<tr>
<td>vlx</td>
<td>VAX-11/780 LSI console floppy interface</td>
</tr>
<tr>
<td>volcopy</td>
<td>copy file systems with label checking</td>
</tr>
<tr>
<td>vpmc</td>
<td>compiler for the virtual protocol machine</td>
</tr>
</tbody>
</table>
2. System Calls

- intro: introduction to system calls and error numbers
- access: determine accessibility of a file
- acct: enable or disable process accounting
- alarm: set a process’s alarm clock
- brk: change data segment space allocation
- chdir: change working directory
- chmod: change mode of file
- chown: change owner and group of a file
- chroot: change root directory
- close: close a file descriptor
- creat: create a new file or rewrite an existing one
- dup: duplicate an open file descriptor
- exec: execute a file
- exit: terminate process
- fcntl: file control
- fork: create a new process
- getpid: get process, process group, and parent process IDs
- getuid: get real user, effective user, real group, and effective group IDs
- ioctl: control device
- kill: send a signal to a process or a group of processes
- link: link to a file
- lseek: move read/write file pointer
- mknod: make a directory, or a special or ordinary file
- mount: mount a file system
- nice: change priority of a process
- open: open for reading or writing
- pause: suspend process until signal
- pipe: create an interprocess channel
- profil: execution time profile
- ptrace: process trace
- read: read from file
- setpgid: set process group ID
- setuid: set user and group IDs
- signal: specify what to do upon receipt of a signal
- stat: get file status
- strftime: set time
- sync: update super-block
- time: get time
- times: get process and child process times
- ulimit: get and set user limits
- umask: set and get file creation mask
- umount: unmount a file system
Table of Contents

uname ........................................ get name of current UNIX system
unlink ........................................ remove directory entry
ustat .......................................... get file system statistics
utime .......................................... set file access and modification times
wait ........................................... wait for child process to stop or terminate
write .......................................... write on a file

3. Subroutines

intro .......................................... introduction to subroutines and libraries
a641 ........................................... convert between long and base-64 ASCII
abort .......................................... generate an IOT fault
abs ............................................ integer absolute value
assert ......................................... program verification
atof ........................................... convert ASCII to numbers
bessel .......................................... bessel functions
bsearch ....................................... binary search
conv .......................................... character translation
crypt .......................................... DES encryption
ctermid ....................................... generate file name for terminal
ctime .......................................... convert date and time to ASCII
cctype ......................................... character classification
cuserid ....................................... character login name of the user
evt ............................................ output conversion
end ............................................. last locations in program
exp ............................................. exponential, logarithm, power, square root functions
fclose ......................................... close or flush a stream
ferror ......................................... stream status inquiries
floor .......................................... absolute value, floor, ceiling, remainder functions
fopen .......................................... open a stream
fptrap .......................................... floating point interpreter
fread .......................................... buffered binary input/output
frexp .......................................... split into mantissa and exponent
fseek .......................................... reposition a stream
gamma .......................................... log gamma function
getc ........................................... get character or word from stream
getenv ......................................... value for environment name
getgrent ..................................... get group file entry
getlogin ...................................... get login name
getopt .......................................... get option letter from argv
getpass ....................................... read a password
getpw .......................................... get name from UID
getpwent ..................................... get password file entry
gets ........................................... get a string from a stream
hypot .......................................... Euclidean distance
13tol .......................................... convert between 3-byte integers and long integers
l32name ....................................... log name of user
lsearch ....................................... linear search and update
malloc ........................................ main memory allocator
mksmem ......................................... make a unique file name
monitor ....................................... prepare execution profile
nlist .......................................... get entries from name list
perror ......................................... system error messages
plot ........................................... graphics interface subroutines
popen .......................................... initiate I/O to/from a process
printf ......................................... output formatters
putc .......................................... put character or word on a stream

- 6 -
Table of Contents

putpwent ........................................... write password file entry
puts ................................................ put a string on a stream
qsort ................................................ quicker sort
rand ................................................... random number generator
regex ................................................ regular expression compile/execute
scanf ............................................... formatted input conversion
setbuf ............................................. assign buffering to a stream
setjmp ............................................... non-local goto
sinh ................................................ hyperbolic functions
sleep ............................................... suspend execution for interval
ssignal .............................................. softw:orr.
stdio .............................................. standard buffered input/output package
string ................................................... string operations
swab ................................................... swap bytes
system ............................................... issue a shell command
tmpfile ............................................ create a temporary file
tmpnam ............................................ create a name for a temporary file
trig ................................................... trigonometric functions
ttyname ............................................. find name of a terminal
ungetc ............................................... push character back into input stream

4. Special Files
intro .............................................. introduction to special files
cat ................................................... phototypesetter interface
dj ..................................................... DJ-11 asynchronous multiplexor
dmc ................................................ communications link with built-in DDCMP protocol
dn ..................................................... DN-11 ACU interface
dqs ................................................... DQS-11 interface for two-point BSC
du ................................................... DU-11 synchronous line interface
dz .................................................... DZ-11, DZ-11/KMC-11, DH-11 asynchronous multiplexers
err .................................................... error-logging interface
hp .................................................... RP04/RP05/RP06 moving-head disk
hs ..................................................... RH11/RJS03-RJS04 fixed-head disk file
ht .................................................... TU16 magnetic tape interface
kl .................................................... KL-11 or DL-11 asynchronous interface
kmc ................................................ KMC11 microprocessor
lp ..................................................... line printer
mem ................................................ core memory
null ................................................... the null file
pcl ................................................ parallel communications link interface
prf ................................................ operating system profiler
rf .................................................... RF11/RS11 fixed-head disk file
rk .................................................... RK-11/RK03 or RK05 disk
rl ................................................... RL-11/RL01 disk
rp .................................................... RP-11/RP03 moving-head disk
st ................................................ synchronous terminal interface
tm ................................................ TM11/TU10 magnetic tape interface
trace ................................................ event-tracing driver
tty ................................................ general terminal interface
vp ................................................ Versatec printer
vpm ................................................ The Virtual Protocol Machine

5. File Formats
intro .............................................. introduction to file formats
a.out ............................................ assembler and link editor output
6. Games

intro ........................................ introduction to games
arithmetic .................................... provide drill in number facts
back ............................................. the game of backgammon
bj ................................................ the game of blackjack
chess ........................................... the game of chess
craps ........................................... the game of craps
hangman ....................................... guess the word
maze ............................................ generate a maze
moo ............................................. guessing game
quiz ............................................ test your knowledge
reversi ........................................ a game of dramatic reversals
sky .............................................. obtain ephemerides
ttt ............................................... tic-tac-toe
wump ........................................... the game of hunt-the-wumpus

7. Miscellaneous Facilities

intro ........................................... introduction to miscellany
ascii ............................................ map of ASCII character set
environ ........................................ user environment
eqncchar ........................................ special character definitions for eqn and neqn
fctl ............................................... file control options
greek ........................................... graphics for the extended TTY-37 type-box
man ................................................ macros for formatting entries in this manual
mm ................................................ the MM macro package for formatting documents
mv ................................................ a macro package for making view graphs
regexp .......................................... regular expression compile and match routines
stat ............................................... data returned by stat system call
term ............................................ conventional names
types ........................................... primitive system data types
8. System Maintenance Procedures

intro .................................. introduction to system maintenance procedures
70boot .................................. 11/70 bootstrap procedures
crash .................................. what to do when the system crashes
diskboot ................................ disk bootstrap programs
etp .................................... Equipment Test Package
filesave ................................ daily/weekly UNIX file system backup
getty .................................... set the modes of a terminal
hasp .................................... RJE (Remote Job Entry) to IBM
init ..................................... process control initialization
makekey ................................ generate encryption key
mk ....................................... how to remake the system and commands
rc ....................................... system initialization shell script
rje ..................................... RJE (Remote Job Entry) to IBM
romboot ................................ special ROM bootstrap loaders
rp6fmt ................................ format and/or check RP06 disk packs
sar ..................................... system activity report package
tapeboot ................................ magnetic tape bootstrap program
unixboot ................................ UNIX startup and boot procedures
uvac .................................... RJE (Remote Job Entry) to UNIVAC
vaxops .................................. VAX-11/780 console operations
PERMUTED INDEX

70boot: 11/70 bootstrap procedures. 70boot(8)

handle special functions of HP 2640 and
functions of DASI 300 and/
special functions of DASI
of DASI 300 and 300s/ 300,
functions of DASI 300 and
l3tol, ltol3: convert between
3-byte integers and long/
comparison. diff3:
 Tektronix 4014 terminal.
paginator for the Tektronix
of the DASI 450 terminal.
special functions of the DASI
procedures. 70boot: 11/70 bootstrap
commands. graphics:
access: determine
enable or disable process
accton: connect-time
acctprc: process
acctsh: shell procedures for
accounting/ acct: overview of
accounting and miscellaneous
acct: per-process
search and print process
acctmerg: merge or add total
summary from per-process
runacct: run daily
and miscellaneous accounting/
per-process accounting/
process accounting file(s).
accounting.
acctcom: search and print
accounting.
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
sin, cos, tan, asin,
sag: system
sar: system
command and generate a system
current SCCS file editing
dn: DN-11
acctcmrg: merge or add total
acctcom: search and print
alarm: set a process's
clock.

abs: integer
floor, fabs, ceil, fmod:
of a file. touch: update
utime: set file
accessibility of a file.
acctcon: connect-time
acctprc: process
acctsh: shell procedures for
accounting/ acct: overview of
accounting and miscellaneous
acct: per-process
search and print process
acctmerg: merge or add total
summary from per-process
runacct: run daily
process accounting.
acct: enable or disable process
acctcom: command summary from
acctcom: search and print
accounting.
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
command and generate a system
activity report. /time a
activity. acct: print
acctcom: search and print
acctmerg: merge or add total
acctprc: process accounting.
acctsh: shell procedures for
acos, atan, atan2/
sag: activity graph.
sar: activity report package.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs</td>
<td>big file scanner.</td>
</tr>
<tr>
<td>fread, fwrite</td>
<td>buffered binary input/output.</td>
</tr>
<tr>
<td>bsearch</td>
<td>binary search.</td>
</tr>
<tr>
<td>remove symbols and relocation</td>
<td>bits. strip.</td>
</tr>
<tr>
<td>bj: the game of</td>
<td>black jack.</td>
</tr>
<tr>
<td>bcopy: interactive</td>
<td>block copy.</td>
</tr>
<tr>
<td>sync: update the super block</td>
<td></td>
</tr>
<tr>
<td>df: report number of free disk blocks.</td>
<td></td>
</tr>
<tr>
<td>sum: sum and count blocks in a file.</td>
<td></td>
</tr>
<tr>
<td>unixboot: UNIX startup and boot procedures.</td>
<td></td>
</tr>
<tr>
<td>romboot: special ROM bootstrap loaders.</td>
<td></td>
</tr>
<tr>
<td>70boot: 11/70 bootstrap procedures.</td>
<td></td>
</tr>
<tr>
<td>tapeboot: magnetic tape bootstrap program.</td>
<td></td>
</tr>
<tr>
<td>diskboot: disk bootstrap programs.</td>
<td></td>
</tr>
<tr>
<td>space allocation.</td>
<td></td>
</tr>
<tr>
<td>modest-sized programs.</td>
<td></td>
</tr>
<tr>
<td>Bj: the game of black jack.</td>
<td></td>
</tr>
<tr>
<td>bcopy: interactive block copy.</td>
<td></td>
</tr>
<tr>
<td>sync: update the super block</td>
<td></td>
</tr>
<tr>
<td>df: report number of free disk blocks.</td>
<td></td>
</tr>
<tr>
<td>sum: sum and count blocks in a file.</td>
<td></td>
</tr>
<tr>
<td>unixboot: UNIX startup and boot procedures.</td>
<td></td>
</tr>
<tr>
<td>romboot: special ROM bootstrap loaders.</td>
<td></td>
</tr>
<tr>
<td>70boot: 11/70 bootstrap procedures.</td>
<td></td>
</tr>
<tr>
<td>tapeboot: magnetic tape bootstrap program.</td>
<td></td>
</tr>
<tr>
<td>diskboot: disk bootstrap programs.</td>
<td></td>
</tr>
<tr>
<td>space allocation.</td>
<td></td>
</tr>
<tr>
<td>modest-sized programs.</td>
<td></td>
</tr>
<tr>
<td>bj: the game of black jack.</td>
<td></td>
</tr>
<tr>
<td>bcopy: interactive block copy.</td>
<td></td>
</tr>
<tr>
<td>sync: update the super block</td>
<td></td>
</tr>
<tr>
<td>df: report number of free disk blocks.</td>
<td></td>
</tr>
<tr>
<td>sum: sum and count blocks in a file.</td>
<td></td>
</tr>
<tr>
<td>unixboot: UNIX startup and boot procedures.</td>
<td></td>
</tr>
<tr>
<td>romboot: special ROM bootstrap loaders.</td>
<td></td>
</tr>
<tr>
<td>70boot: 11/70 bootstrap procedures.</td>
<td></td>
</tr>
<tr>
<td>tapeboot: magnetic tape bootstrap program.</td>
<td></td>
</tr>
<tr>
<td>diskboot: disk bootstrap programs.</td>
<td></td>
</tr>
<tr>
<td>space allocation.</td>
<td></td>
</tr>
<tr>
<td>modest-sized programs.</td>
<td></td>
</tr>
<tr>
<td>bj: the game of black jack.</td>
<td></td>
</tr>
<tr>
<td>bcopy: interactive block copy.</td>
<td></td>
</tr>
<tr>
<td>sync: update the super block</td>
<td></td>
</tr>
<tr>
<td>df: report number of free disk blocks.</td>
<td></td>
</tr>
<tr>
<td>sum: sum and count blocks in a file.</td>
<td></td>
</tr>
<tr>
<td>unixboot: UNIX startup and boot procedures.</td>
<td></td>
</tr>
<tr>
<td>romboot: special ROM bootstrap loaders.</td>
<td></td>
</tr>
<tr>
<td>70boot: 11/70 bootstrap procedures.</td>
<td></td>
</tr>
<tr>
<td>tapeboot: magnetic tape bootstrap program.</td>
<td></td>
</tr>
<tr>
<td>diskboot: disk bootstrap programs.</td>
<td></td>
</tr>
</tbody>
</table>
Permuted Index

eqn delimiters. mmchek(l)
constant-width text for/ cw,  
text for nroff or/ eqn, neqn,
lint: a C program
grpck: password/group file
copy file systems with label
systems processed by fsck.
chess: the game of chess
check.
check usage of mm macros and
checkeq: format mathematical
eqn(ch(1)
checker.
lint(ch(1)
checkers. pwck,
checking. volcopy, labelit:
volcopy(ch(1)
checklist: list of file
chess.
checklist(ch(5)
chess: the game of chess.
chgrp: change owner or group.
chown(ch(1)
times: get process and
terminate. wait: wait for
of a file.
group.
for a command.
iscntrl, isascii: character
classification. /isgraph,
clear-i-node.
clearerr, fileno: stream
status/ ferror, feof,
clock daemon.
close: close a file descriptor.
close: close a file
close or flush a stream.
close: clear i-node.
clock.
cmp: compare two files.
cmp.
cmp: compare two files.
cmp: compare two files.
cmp: compare two files.
cmp: compare two files.
cmp: compare two files.
cmp.
mush:
close: close a file
close: close a file
close: close a file
close: close a file
close: close a file
close: close a file
status/ ferror, feof,
clock daemon.
close: close a file descriptor.
close: close a file
close or flush a stream.
close: clear i-node.
clock.
cmp: compare two files.
cmp.
cmp: compare two files.
cmp: compare two files.
cmp: compare two files.
cmp: compare two files.
cmp.
mush:
close: close a file
close: close a file
close: close a file
close: close a file
close: close a file
close: close a file
status/ ferror, feof,
clock daemon.
close: close a file descriptor.
close: close a file
close or flush a stream.
close: clear i-node.
clock.
cmp: compare two files.
cmp.
cmp: compare two files.
cmp: compare two files.
cmp: compare two files.
cmp: compare two files.
cmp.
mush:
regcmp: regular expression compiler.

cc, gcc: C compiler.
f77: Fortran 77 compiler.
programs. scc: C compiler.
portable machine. vpmc: C compiler.
yacc: yet another parser generator.
modest-sized programs. bs: a block assembler.
wait: await completion of process.
pack, pcut, unpack: compress and expand files.
cat: concatenate and print files.
test: test condition.
system: configure a UNIX system.
acctcon: connect-time accounting.

interactive/ fsck: file system consistency check and repair.
vlx: VAX-11/780 LSI console operations.
vaxops: VAX-11/780 console operations.
report and interactive status report.
checkcw: prepare constant-width text for troff.
mkfs: construct a file system.
execute command. xargs: construct argument list(s) and execute command.
nroff/troff, tbl, and eqn constructs.
ls: list contents of directories.
toc: graphical table of contents routines.
csplit: split context.
ioctl: control device.
fcnt!: file control options.
st: synchronous terminal control.
uucp status inquiry and job control. uuestat: RJE status report and interactive status console.
cw, checkcw: prepare constant-width text for troff.
cw(1)
fcntl!: file control.
cntlp: construct a file system.

inittab: control information for init.
run: process control initialization.
fcntl!: file control options.

ecvt, fcvt: output conversion.
units: conversion program.
scanf: formatted input conversion. scanf, fscanf, sscanf.

dd: convert and copy a file.
cpio: format of cpio archive.
check: format of core image file.
mem, kmem: core memory.
atan2: trigonometric/ sin, cos, tan, asin, acos, atan.
functions. sinh, cosh, tanh: hyperbolic functions.
sum: sum and count blocks in a file.
cpio: format of cpio archive.

program. sce: C compiler for stand-alone protocol machine.
vpmc: compiler for the virtual machine.
yace: yet another compiler-compiler.
bs: a compiler/interpreter for modest-sized programs.
wait: await completion of process.
pack, pcut, unpack: compress and expand files.
cat: concatenate and print files.
test: test condition.
system: configure a UNIX system.
acctcon: connect-time accounting.

interactive/ fsck: file system consistency check and repair.
vlx: VAX-11/780 LSI console operations.
vaxops: VAX-11/780 console operations.
report and interactive status report.
checkcw: prepare constant-width text for troff.
mkfs: construct a file system.
execute command. xargs: construct argument list(s) and execute command.
nroff/troff, tbl, and eqn constructs.
ls: list contents of directories.
toc: graphical table of contents routines.
csplit: split context.
ioctl: control device.
fcnt!: file control options.
st: synchronous terminal control.
uucp status inquiry and job control. uuestat: RJE status report and interactive status console.
cw, checkcw: prepare constant-width text for troff.
cw(1)
fcntl!: file control options.

ecvt, fcvt: output conversion.
units: conversion program.
scanf: formatted input conversion. scanf, fscanf, sscanf.

dd: convert and copy a file.
cpio: format of cpio archive.
check: format of core image file.
mem, kmem: core memory.
atan2: trigonometric/ sin, cos, tan, asin, acos, atan.
functions. sinh, cosh, tanh: hyperbolic functions.
sum: sum and count blocks in a file.
cpio: format of cpio archive.

program. sce: C compiler for stand-alone protocol machine.
vpmc: compiler for the virtual machine.
yace: yet another compiler-compiler.
bs: a compiler/interpreter for modest-sized programs.
wait: await completion of process.
pack, pcut, unpack: compress and expand files.
cat: concatenate and print files.
test: test condition.
system: configure a UNIX system.
acctcon: connect-time accounting.

interactive/ fsck: file system consistency check and repair.
vlx: VAX-11/780 LSI console operations.
vaxops: VAX-11/780 console operations.
report and interactive status report.
checkcw: prepare constant-width text for troff.
mkfs: construct a file system.
execute command. xargs: construct argument list(s) and execute command.
nroff/troff, tbl, and eqn constructs.
ls: list contents of directories.
toc: graphical table of contents routines.
csplit: split context.
ioctl: control device.
fcnt!: file control options.
st: synchronous terminal control.
uucp status inquiry and job control. uuestat: RJE status report and interactive status console.
cw, checkcw: prepare constant-width text for troff.
cw(1)
fcntl!: file control options.

ecvt, fcvt: output conversion.
units: conversion program.
scanf: formatted input conversion. scanf, fscanf, sscanf.

dd: convert and copy a file.
cpio: format of cpio archive.
check: format of core image file.
mem, kmem: core memory.
atan2: trigonometric/ sin, cos, tan, asin, acos, atan.
functions. sinh, cosh, tanh: hyperbolic functions.
sum: sum and count blocks in a file.
cpio: format of cpio archive.

program. sce: C compiler for stand-alone protocol machine.
vpmc: compiler for the virtual machine.
yace: yet another compiler-compiler.
bs: a compiler/interpreter for modest-sized programs.
wait: await completion of process.
pack, pcut, unpack: compress and expand files.
cat: concatenate and print files.
test: test condition.
system: configure a UNIX system.
acctcon: connect-time accounting.

interactive/ fsck: file system consistency check and repair.
vlx: VAX-11/780 LSI console operations.
vaxops: VAX-11/780 console operations.
report and interactive status report.
checkcw: prepare constant-width text for troff.
mkfs: construct a file system.
execute command. xargs: construct argument list(s) and execute command.
nroff/troff, tbl, and eqn constructs.
ls: list contents of directories.
toc: graphical table of contents routines.
csplit: split context.
ioctl: control device.
fcnt!: file control options.
st: synchronous terminal control.
uucp status inquiry and job control. uuestat: RJE status report and interactive status console.
cw, checkcw: prepare constant-width text for troff.
cw(1)
fcntl!: file control options.

ecvt, fcvt: output conversion.
units: conversion program.
scanf: formatted input conversion. scanf, fscanf, sscanf.

dd: convert and copy a file.
cpio: format of cpio archive.
check: format of core image file.
mem, kmem: core memory.
atan2: trigonometric/ sin, cos, tan, asin, acos, atan.
functions. sinh, cosh, tanh: hyperbolic functions.
sum: sum and count blocks in a file.
cpio: format of cpio archive.

program. sce: C compiler for stand-alone protocol machine.
vpmc: compiler for the virtual machine.
yace: yet another compiler-compiler.
bs: a compiler/interpreter for modest-sized programs.
wait: await completion of process.
pack, pcut, unpack: compress and expand files.
cat: concatenate and print files.
test: test condition.
system: configure a UNIX system.
acctcon: connect-time accounting.
Permuted Index
craps: the game of craps. . • • • • • • . • • •
craps: the game of craps.
crash: examine system images.
system crashes. crash: what to do when the
what to do when the system crashes. crash: • • • • • . •
rewrite an existing one. creat: create a new file or
file. tmpnam: create a name for a temporary
an existing one. creat: create a new file or rewrite
fork: create a new process. . . •
tmpfile: create a temporary file.
channel. pipe: create an interprocess . . •
files. admin: create and administer sces
umask: set and get file creation mask. • • • • .
listing. cref: make cross-reference
cron: clock daemon.
programs. xref: cross reference for C
cref: make cross-reference listing.
crypt: encode/decode.
encryption. crypt, setkey, encrypt: DES
csplit: context split. • • • •
ct: call terminal. . • • . •
for terminal. ciermid: generate file name
asctime. tzset: convert date/ ctime, localtime, gmtime, •
cu: call another UNIX system.
ttt. cubic: tic-tac-toe. • . . .
actIVIty. sact: print current sces file editing
uname: get name of current UNIX system.
uname: print name of current UNIX. . • • . .
spline: interpolate smooth curve. '" • • • . • . •
of the user. cuserid: character login name
of each line of a file. cut: cut out selected fields
each line of a file. cut: cut out selected fields of
constant-width text fori cw, checkcw: prepare ••
cron: clock daemon.
.••••••
sending daemons, line printer daemon. /odpd, Ipd: HONEYWELL
errdemon: error-logging daemon.
..•••.
terminate the error-logging daemon. errstop: . • . . • •
fget.odemon: file retrieval daemons. fget.demon,
/odpd, Ipd: HONEYWELL sending daemons, line printer daemon.
runacct: run daily accounting. • • • • • •
backup. filesave, tapesave: daily/weekly UNIX file system
/handle special functions of DASI 300 and 300s terminals.
special functions of the DASI 450 terminal. /handle •
prof: display profile. data. • • . • . • • • • • •
call. stat: data returned by stat system •
brk, sbrk: change data segment space allocation.
types: primitive system data types. • • • • • •
join: relational database operator.
/asctime, tzset: convert date and time to ASCn.
date: print and set the date. • . • . • . • .
date: print and set the date.
dc: desk calculator. • • • .
dd: convert and copy a file.
/link with built-in DDCMP protocol.
adb: debugger.
fsdb: file system debugger.
sdb: symbolic debugger.
sysdef: system definition.
eqnchar: special character definitions for eqn and neqn.
usage of mm macros and eqn delimiters. mmchek: check
names. basename, dirname: deliver portions of path
file. tail: deliver the last part of a
delta commentary of an sces delta. cdc: change the •

-6-

craps(6)
craps(6)
crash(IM)
crash(8)
crash(8)
creat(2)
tmpnam(3S)
creat(2)
fork(2)
tmpfile(3S)
pipe(2)
admin(l)
umask(2)
cref( I)
cron(lM)
xref(l )
cref(I )
crypt( I)
crypt(3C)
csplit(l )
ct(lC)
ctermid(3S)
ctime(3C)
cu(lC)
ttt(6)

sact( I)
uname(2)
uname(l)
spline(IG)
cuserid(3S)
cut(1)
cut(l)
cw(J)
cron(IM)
dpd(IC)
errdemon(lM)
errstop( I M)
fget.demon( I C)
dpd(IC)
runacct( I M)
filesave(8)
300(1)
450(1)
prof(l)
stat(7)
brk(2)
types(7)
join(l)
ctime(3C)
date(l )
date( I)
dc(l)
dd(l)
dmc(4)
adb(l)
fsdb(lM)
sdb(l)
sysdef(IM)
eqnchar(7)
mmchek(I)
basename( I )
tail(I)
cdc(l )


file. delta: make a delta (change) to an SCSS
delta. cdc: change the delta commentary of an SCSS
deltal: remove a delta from an SCSS file.
comb: combine SCSS
desc: describe
mesg: permit or tbl, and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
tbl. and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
tbl. and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
tbl. and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
tbl. and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
tbl. and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
tbl. and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
tbl. and eqn constructs.
crypt, setkey, encrypt: DES encryption.
close: close a file
dup: duplicate an open file
dc: file. access:
devnm: device name.
mesg: permit or deny messages.
Permuted Index

multiplexor. dj: DJ-11 asynchronous
kl: KL-11 or DL-11 asynchronous interface.
built-in DDCMP protocol.
dm: communications link with dmc:
dn: DN-11 ACU interface.
MM macros. mm: print out documents formatted with the
macro package for formatting
slides. mmt, mvt: typeset documents, view graphs and
who: who is
sending daemons, line printer/
two-point BSC. dqs: DQS-II interface for
reversi: a game of dramatic reversals.
graph: draw a graph.
math: provide drill in number facts.
trace: event-tracing interface.
interface. du: DU-11 synchronous line
du: summarize disk usage.
dump: incremental file system
dump: incremental dump tape
don: DN-11 ACU interface.
dp: off-line print.
dqs: DQS-II interface for two-point
reversi: dramatic reversals.
dump: incremental dump file system
dump: incremental dump tape
dump: duplicate an open file
dump: duplicate an open file
DZ-II/KMC-11, DH-II/
dz, dzk, dh: DZ-11, DZ-II/KMC-11, DH-II/
dz, dzk, dh: DZ-11, DZ-II/KMC-11, DH-II/
DH-II asynchronous/ dz, dzk, dh:
dz, dzk, dh: DZ-11, DZ-II/KMC-11, DH-II/
DH-II asynchronous/ dz, dh: DZ-11, DZ-II/KMC-11, DH-II/
der: DU-11 asynchronous line
endgrent: get group file entry.
getgrnam, setgrent, endgrent: get group file entry.
getgrnam, setgrent, endgrent: get group file entry.
getpwent, setpwent, putpwent: write password file entry.
hasp: RJE (Remote Job Entry) to IBM.

INDEX

- 8 -
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rje</td>
<td>RJE (Remote Job) to IBM</td>
</tr>
<tr>
<td>uvac</td>
<td>RJE (Remote Job) to UNIVAC</td>
</tr>
</tbody>
</table>
expression routines. regexp: regular expression compile and match
regcmp: regular expression compile.
regex, regcmp: regular expression compile/execute.
expr: evaluate arguments as an expression.
greek: graphics for the HONEYWELL 6000.
dump, errdead:
value, floor, ceiling, / floor, large primes, factor, primes:
number, generate large/
true, abort: generate an IOT
a stream.
ecvt, fopen, freopen, status inquiries. feoff, fileno: stream status/
stream, fclose: file control.
HONEYWELL 6000,
word from, getc, getchar, fgetc, getw: get character or
retrieval daemons. fget, fgets: get a string from a
pattern, grep, egrep, times, utime: set
determine accessibility of a file.
tar: tape
ecvt, fcntl: file control options.
close, flush: close or flush
fcntl: file control.
file access and modification
utime: set
file access:
tar:
cpio: copy
file archives in and out.
pwck, grpck: password/group change owner and group of a file.
chmod: change mode of
diff: differential
diff3: 3-way differential
fcntl: file control.
fcntl: file control options.
upick: public UNIX-to-UNIX
core: format of core image
umask: set and get
file creation mask.
file: cut: cut out selected
dd: convert and copy a a file.
fields each line of a
dd: convert and copy a
a delta (change) to an SCCS
close: close a
dup: duplicate an open
sact: print current SCCS
setgrent, endgrent: get group
dupw: get password
putpwent, read password
execvp: execute a
grep, egrep, fgrep: search a
acct: per-process accounting
ar: archive
errfile: error-log
punch: intro: introduction to
tar: tape
group: group
fixed-head disk split: split a
file: h: RH11/RJS03-RJS04
file into pieces.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>link to a file.</td>
</tr>
<tr>
<td>mknod</td>
<td>build special or a special or ordinary file name for terminal.</td>
</tr>
<tr>
<td>ctermid</td>
<td>generate null: the null file.</td>
</tr>
<tr>
<td>mktemp</td>
<td>make a unique file or rewrite an existing file or /lines of several files.</td>
</tr>
<tr>
<td>null</td>
<td>the null file.</td>
</tr>
<tr>
<td>create</td>
<td>create a new file or /make a directory,</td>
</tr>
<tr>
<td>passwd</td>
<td>password file.</td>
</tr>
<tr>
<td>ls</td>
<td>move read/write file pointer.</td>
</tr>
<tr>
<td>prs</td>
<td>print an SCSS read from file.</td>
</tr>
<tr>
<td>reform</td>
<td>reformat text.</td>
</tr>
<tr>
<td>fget.demon, fget.odemon</td>
<td>file retrieval daemons.</td>
</tr>
<tr>
<td>rf</td>
<td>RF11/RS11 fixed-head disk file.</td>
</tr>
<tr>
<td>rm</td>
<td>remove a delta from an SCSS file system backup.</td>
</tr>
<tr>
<td>bfs</td>
<td>big file scanner.</td>
</tr>
<tr>
<td>sccsdiff</td>
<td>compare two versions of an SCSS file.</td>
</tr>
<tr>
<td>sccsfile</td>
<td>format of SCSS file.</td>
</tr>
<tr>
<td>size</td>
<td>size of an object file status.</td>
</tr>
<tr>
<td>stat, fstat</td>
<td>get file system statistics.</td>
</tr>
<tr>
<td>sum</td>
<td>sum and count blocks in a file.</td>
</tr>
<tr>
<td>tapesave, daily/weekly UNIX</td>
<td>file system.</td>
</tr>
<tr>
<td>filesave</td>
<td>file systems with label/ account a delta from an SCSS file.</td>
</tr>
<tr>
<td>fsck</td>
<td>system consistency check file systems processed by</td>
</tr>
<tr>
<td>fget</td>
<td>retrieve files from the HONEYWELL 6000.</td>
</tr>
<tr>
<td>fsdb</td>
<td>file system debugger.</td>
</tr>
<tr>
<td>dump</td>
<td>incremental file system dump.</td>
</tr>
<tr>
<td>mkfs</td>
<td>construct a file system.</td>
</tr>
<tr>
<td>umount</td>
<td>mount and dismount file system.</td>
</tr>
<tr>
<td>mount</td>
<td>mount a file system.</td>
</tr>
<tr>
<td>restor</td>
<td>incremental file system restore.</td>
</tr>
<tr>
<td>ustat</td>
<td>get file system statistics.</td>
</tr>
<tr>
<td>mnttab</td>
<td>mounted file system table.</td>
</tr>
<tr>
<td>umount</td>
<td>unmount a file system.</td>
</tr>
<tr>
<td>fsck, checklist</td>
<td>list of file systems processed by</td>
</tr>
<tr>
<td>volcopy</td>
<td>systems with label/ file systems.</td>
</tr>
<tr>
<td>labelit</td>
<td>copy file systems with label/ file systems processed by</td>
</tr>
<tr>
<td>deliver</td>
<td>the last part of a file.</td>
</tr>
<tr>
<td>tmpfile</td>
<td>create a temporary file.</td>
</tr>
<tr>
<td>create</td>
<td>a name for a temporary file.</td>
</tr>
<tr>
<td>file</td>
<td>determine file type.</td>
</tr>
<tr>
<td>undo</td>
<td>a previous get of an SCSS file.</td>
</tr>
<tr>
<td>report</td>
<td>repeated lines in a file.</td>
</tr>
<tr>
<td>val</td>
<td>validate SCSS file.</td>
</tr>
<tr>
<td>write</td>
<td>write on a file.</td>
</tr>
<tr>
<td>umask</td>
<td>set file-creation mode mask.</td>
</tr>
<tr>
<td>ferror, feof, clearerr, and print process accounting</td>
<td>file-no: stream status/ file(s).</td>
</tr>
<tr>
<td>merge or add total accounting</td>
<td>acctcom: search acctcom(1).</td>
</tr>
<tr>
<td>send, gath: gather</td>
<td>files. acctmerg: acctmerg(1M).</td>
</tr>
<tr>
<td>cat: concatenate and print</td>
<td>files. admin: admin(1).</td>
</tr>
<tr>
<td>cmp: compare two files.</td>
<td>files and/or submit RJE jobs.</td>
</tr>
<tr>
<td>cp, ln, mv: copy, link or move</td>
<td>files. comm: select or reject comm(1).</td>
</tr>
<tr>
<td>mark differences between files.</td>
<td>files. diffmk: diffmk(1).</td>
</tr>
<tr>
<td>find: find</td>
<td>files. find(1).</td>
</tr>
<tr>
<td>arcv: convert archive</td>
<td>files. from PDP-11 to/ arcv(1).</td>
</tr>
<tr>
<td>fget: retrieve</td>
<td>files. from the HONEYWELL 6000. fget(1C).</td>
</tr>
<tr>
<td>format specification in text string, format of graphical files.</td>
<td></td>
</tr>
<tr>
<td>intro: introduction to special files.</td>
<td>files. /graphical primitive gpa(5).</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Permuted Index

rm, rmdir: remove files or directories. .......... rm(1)
PASTE: paste file blocks. .......................... paste(1)
/pr: print pack, pcat. ............................... pack(1)
files. ................................................. pr(1)
/files. ................................................. sort(1)
files to the HONEYWELL 6000. ....................... fsend(1C)
/files. ................................................. what(1)
/filesave, tapesave: ................................. filesave(8)
filter. .................................................. greek(1)
/files. ................................................. nl(1)
/files. .................................................. col(1)
/find files. ............................................ find(1)
/find files. ............................................ find(1)
/find files. ............................................ find(1)
/find files. ............................................ find(1)
/hyphen. ................................................. hyphen(1)
/find hyphenated words. ............................... hyphen(1)
/find name of a terminal. ............................. tyname(3C)
/find ordering relation for an object library. ....... lorder(1)
/find possible typographical errors. ................... typo(1)
/find spelling errors. ................................ spell(1)
/fitting .................................................. tee(1)
/fixed-head disk file. ................................ hs(4)
/fixed-head disk file. ................................ rf(4)
/fputsrap: floating point interpreter. .............. fptrap(3X)
/absolute value, floor.,/ceil, fmod: absolute value, floor. .......... floor(3M)
/close, flush: close or flush a stream. .............. fclose(3S)
/ceiling, floor, fabs, ceil, stream. ................. floor(3M)
/fopen, freopen, fopen: open a stream. ............... fopen(3S)
/fork: create a new process. ........................... fork(2)

per-process accounting file format. acct. .......... acct(5)
/packs. rpsfmt: format and/or check RP06 disk .......... rpsfmt(8)
/ar: archive file format. ................................ ar(9)
/from PDP-11 to VAX-11/780 format. /convert archive files .......... arcv(1)
/dump: incremental dump tape format. .................. dump(5)
/errfile: error-log file format. ......................... errfile(5)
/pnch: file format for card images. ..................... pnch(5)
/nroff or/ eqn, neqn, checkeq: format a mathematical text for eqn(1)
/inode: format of an inode. ................................ inode(5)
/core: format of core image file. ......................... core(5)
/cpio: format of cpio archive. .......................... cpio(5)
/dir: format of directories. ............................. dir(5)
/graphical primitive string, format of graphical files. ....... gimp(5)
/fsccfile: format of SCSS file. .......................... fsccfile(5)
/file system: format of system volume. ................. fs(5)
/files. fspec: format specification in text ............. fspec(5)
/troff. tbl: format tables for nroff or troff. .......... tbl(1)
/troff, nroff: typeset or format text. ................... troff(1)
/tp: magnetic tape format. ................................ tp(5)
/wtmp: utmp and wtmp entry format. ...................... utmp(5)
/intro: introduction to file format. ..................... intro(5)
/scanf, fscanf, sscanf: formatted input conversion. ....... scanf(3S)
/mm: print out documents formatted with the MM macros. formatted with the MM macros. .......... mm(1)
/fprintf, sprintf: output formatters. printf. ............ printf(3S)
/mm: the MM macro package for formatting documents. .......... mm(7)
/manual. man: macros for formatting entries in this .......... man(7)
/f77: Fortran 77 compiler. ................................ f77(1)
/ratfor: rational Fortran dialect. ........................ ratfor(1)
/e7: Extended Fortran Language. .......................... eff(1)
/formatters. printf, fprintf, sprintf: output .......... printf(3S)
/interpreter. fptrap: floating point fptrap(3X)
/word on a/ putc, putwchar, fputc, putw: put character or ....... putc(3S)
Permuted Index

gets, fgets: get a string from a stream.
get: get a version of an SCCS file.
ulimit: get and set user limits.
getc, getchar, fgetc, getw: get character or word from /
unlist: get entries from name list.
ulimit: get and set user limits.
gete(3S)
get: get a version of an SCCS file.
stat(2)
get file system statistics.
stat(2)
get: get a string from a stream.

/getgrnam, setgrent, endgrent: get group file entry.
getlogin: get login name.
logname: get login name.
getpw: get name from UID.
getreal: get real user.
get environment.

/getpwnam, setpwent, endpwent: get password file entry.
getgrent: get group file entry.
getgrent: get group file entry.
getgrent: get group file entry.
get pwent: get name from UID.
setgrent: get group file entry.
getpwent: get password file entry.

get character or word from /
character or word from / getc, gete, getechar, getfgete, getw:
getegid: get real user.
getenv: value for environment
getuid.
getuid.
getuid.
getuid.
getuid.
group.
space.
user.
getgrent.
getpwent.
getpw.

process group, and / gethprompt, gethprompt, gethprompt, gethprompt, gethprompt, gethprompt,

setpwent, endpwent: get/
get/ getpwent, getpuid, endpwent: get/ getpwent, a stream.
terminal.
getegid: get real user.

from/ getc, getchar, fgetc, convert/ ctime, localtime, setjmp, longjmp: non-local string, format of graphical/

graphic primitive graph: draw a graph.

sag: system activity
commands: access

/network use useful with
gerade, hardcopy, tekst, td:

ged: get character or word

gmtime, asctime, tzset:
goto.

setjmp(3C)

stat(1G)
gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)

gdev(1G)
maintenance/ intro: introduction to system
ncheck: generate names from
popen, pclose: initiate
/ islower, isdigit, isxdigit,
isalpha, isupper, islower,
isalnum, isspce, ispunct,/ isalpha, isupper, islower,
isalnum, isspce, ispunct,
isxdigit, isalnum, isspace,
isxdigit, isalnum, isspace,
isdigit, isxdigit, isalnum,
isalnum, isspace, ispunct,
isdigit, isxdigit, isalnum,
isalnum, isspace, ispunct,
isalpha, isupper, islower,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalpha, isupper, islower,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
isalnum, isspace, ispunct,
Permuted Index

Isearch:
col: filter reverse
files. comm: select or reject
uniq: report repeated
of several files or subsequent
subsequent/ paste: merge same
link, unlink: exercise
Id:
pel: parallel communications
cp, In, mv: copy,

link:
and unlink system calls.
protocol. dmc: communications
a.out: assembler and
Is:
nlist: get entries from name
nm: print name
by fsck. checklist:
cref: make cross-reference
xargs: construct argument
files. cp,
vpmstart, vpmsnap, vpmtrace:
romboot: special ROM bootstrap
tzset: convert date/ ctime,
end, etext, edata: last
gamma:
newgrp:
logarithm, power, square/ exp,
/Iog, pow, sqrt: exponential,
errpt: process a report of
getlogin: get
logname: get
cuserid: character
logname:
passwd: change
setting up an environment at
a64I, 164a: convert between
between 3-byte integers and
setjmp,
for an object library.
nice: run a command at
daemons, line/ dpd, odpd,
directories.
update.
pointer.
vlx: VAX-l 1/780
integers and long/ 13tol,
vpm: The Virtual Protocol
for the virtual protocol
documents. mm: the MM
graphs. mv: a
m4:
mmchek: check usage of mm
in this manual. man:

line: read one line.
linear search and update.
line-feeds. • . . • • • •
lines common to two sorted
lines in a file. • • . • • •
lines of one file. /same lines
lines of several files or
link and unlink system calls.
link editor.
link interface.
link: link to a file.
link or move files.
link to a file. . . •
link, unlink: exercise link
link with built-in DDCMP
link editor output.
lint: a C program checker.
list contents of directories.
list. . . . . . . . • • •
list. . • . . . . • . • •
list of file systems processed
•••••••••
listing.
Iist(s) and execute command.
In, mv: copy, link or move
load the KMCII-B: print VPM/
loaders. • • • . • • • • •
localtime, gmtime, asctime,
locations in program. . • •
log gamma function.
log in to a new group.
log. pow, sqrt: exponential,
logarithm, power, square root/
logged errors.
login name. . • . • •
login name. • • • • •
login name of the user.
login name of user.
login password.
login: sign on.
login time. profile:
logname: get login name.
logname: login name of user.
long and base-64 ASCn:
long integers. /ltoI3: convert
longjmp: non-local goto.
lorder: find ordering relation
low priority. • • • • • • •
Ip: line printer. • • . . • •
Ipd: HONEYWELL sending
Ipr: line printer spooler. • •
Is: list contents of
Isearch: linear search and
Iseek: move read/write file
lSI console 110ppy interface.
Itol3: convert between 3-byte
m4: macro processor. • • • •
Machine. • • • • • • • • •
machine. vpmc: compiler . •
macro package for formatting
macro package for making view
macro processor. • • • • • •
macros and eqn delimiters.
macros for formatting entries

- 18 -

line(1)
\search(3C)
col(1)
comm(l)
uniq(1)
paste(l)
paste(1)
link(1M)
Id(1)
pel(4)
link(2)
cp(\)

link(2)
link(1M)
dmc(4)
a.out(5)
lint( 1)
Is(1)
nlist(3C)
nm(1)
checklist(5)
cref(1)
xargs(l)
cp(l)

vpmstart(1C)
romboot(8)
ctime(3C)
end(3C)
gamma(3M)
newgrp(1)
exp(3M)
exp(3M)
errpt(IM)
getiogin(3C)
10gname(1)
cuserid(3S)
logname(3X)
passwd(l)
10gin(1 )
profile(5)
logname(l)
logname(3X)
a641(3C)
13tol(3C)
setjmp(3C)
lorder(l)
nice(l)
Ip(4)
dpd(lC)
Ipr(1)
15(1 )
Isearch(3C)
Iseek(2)
vlx(1M)
13tol(3C)
m4(1)
vpm(4)
vpmc(lC)
mm(7)
mv(7)
m4(1)
mmchek(1)
man(7)


<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tapeboot</td>
<td>magnetic tape bootstrap</td>
</tr>
<tr>
<td>tp</td>
<td>magnetic tape format.</td>
</tr>
<tr>
<td>ht</td>
<td>magnetic tape interface.</td>
</tr>
<tr>
<td>tm</td>
<td>magnetic tape interface.</td>
</tr>
<tr>
<td>mail</td>
<td>mail to users or read mail.</td>
</tr>
<tr>
<td>mail</td>
<td>mail to HIS user.</td>
</tr>
<tr>
<td>gcsmail</td>
<td>send mail to HIS user.</td>
</tr>
<tr>
<td>tmll</td>
<td>maintain, update, and maintain.</td>
</tr>
<tr>
<td>mail</td>
<td>send mail to users or read mail.</td>
</tr>
<tr>
<td>malloc</td>
<td>main memory allocator.</td>
</tr>
<tr>
<td>mknod</td>
<td>make a directory, or a special file.</td>
</tr>
<tr>
<td>mktemp</td>
<td>make a unique file name.</td>
</tr>
<tr>
<td>cref</td>
<td>make cross-reference listing.</td>
</tr>
<tr>
<td>regen</td>
<td>regenerate groups of files.</td>
</tr>
<tr>
<td>banner</td>
<td>make posters.</td>
</tr>
<tr>
<td>mail</td>
<td>send mail to users or read mail.</td>
</tr>
<tr>
<td>malloc</td>
<td>main memory allocator.</td>
</tr>
<tr>
<td>free</td>
<td>split into mantissa and exponent.</td>
</tr>
<tr>
<td>man</td>
<td>print entries in this manual.</td>
</tr>
<tr>
<td>for</td>
<td>for formatting entries in this manual.</td>
</tr>
<tr>
<td>ascii</td>
<td>map of ASCII character set.</td>
</tr>
<tr>
<td>difmk</td>
<td>mark differences between man files.</td>
</tr>
<tr>
<td>umask</td>
<td>set file-creation mode of files or subsequent/ paste.</td>
</tr>
<tr>
<td>master</td>
<td>master device information table.</td>
</tr>
<tr>
<td>regular</td>
<td>match routines. regexp: format mathematical text for nroff or/eqn, neqn, checkeq: format maze: generate a maze.</td>
</tr>
<tr>
<td>expression</td>
<td>free, realloc, calloc: main memory allocator.</td>
</tr>
<tr>
<td>mem</td>
<td>memory allocator.</td>
</tr>
<tr>
<td>sort</td>
<td>sort and/or files. acctmerg: merge or add total accounting.</td>
</tr>
<tr>
<td>files</td>
<td>merge files.</td>
</tr>
<tr>
<td>sys_nerr</td>
<td>messages.</td>
</tr>
<tr>
<td>sys_errno</td>
<td>messages.</td>
</tr>
<tr>
<td>kas</td>
<td>assembler for the KMC11 microprocessor.</td>
</tr>
<tr>
<td>kmc</td>
<td>assembler for the KMC11 microprocessor.</td>
</tr>
<tr>
<td>mk</td>
<td>how to remake the system.</td>
</tr>
<tr>
<td>special</td>
<td>special or ordinary file. name.</td>
</tr>
<tr>
<td>mm</td>
<td>formatting documents. mm: the MM macro package for documents formatted with the MM macros.</td>
</tr>
<tr>
<td>mmchek</td>
<td>check usage of documents formatted with the MM macros.</td>
</tr>
<tr>
<td>mm</td>
<td>formatting documents. mm: the MM macro package for documents formatted with the MM macros.</td>
</tr>
<tr>
<td>mmchek</td>
<td>check usage of documents formatted with the MM macros.</td>
</tr>
</tbody>
</table>
Permuted Index

view graphs, and slides. mmt, mvt: typeset documents, mmt(1)
setmnt: establish mnttab: mounted file system mnttab(5)
chmod: change mnttab table. setmnt(1M)
chmod: change mode. chmod(1)
chmod: change mode mask. chmod(1)
chmod: change mode of file. chmod(2)
chmod: change modes of a terminal. getty(8)
bs: a compiler/interpreter for exponent. frexp, ldexp.
modf: split into mantissa and touch: update access and
modification times of a file. touch(1)
utation: set file access and profile. monitor: prepare execution
getux: monitor uucp network. monitor(3C)
getux: monitor uucp network. umnttab(5)
mnttab: mounted file system setmnt(IM)
mnttab: mounted file system table. mnttab(5)
mvdir: move a directory. mvdir(1M)
cp, ln, mv: copy, link or move files. cp(1)
seek: move read/write file pointer. lseek(2)
hp: RP04/RP05/RP06 moving-head disk. hp(4)
rp: RP-11/RP03 moving-head disk. rp(4)
DH-11 asynchronous multiplexers. /DZ-11/KMC-11, dz(4)
dj: DJ-11 asynchronous multiplexor. dj(4)
view graphs. mv: a macro package for making
cp, ln, mv: copy, link or move files. cp(1)
mvdir: move a directory. mvdir(1M)
graphs, and slides. mmt, i-numbers.
ncheck: generate names from ncheck(IM)
frexp(3C)
ncheck: generate names from ncheck(IM)
nl: line numbering filter. nl(1)
list.
stat: statistical network useful with graphical
network. uusub: monitor uucp
newgrp: log in to a new group. newgrp(1)
news: print news items. news(1)
process. nice: change priority of a
priority. nice: change priority of a
nice: run a command at low
priority. nice: run a command at low
nl: line numbering filter. nl(1)
list.
list.
nl: line numbering filter. nl(1)
nlist: get entries from name
list.
nlist: get entries from name
nm: print name list. nm(1)
nice: run a command at low
nice: run a command at low
non-local goto. setjmp(3C)
format mathematical text for/troff.
/nroff or troff. /checkeq: /checkeq:
com_opts: format tables for troff
com_opts: format tables for troff
nroff/troff, tbl, and eqn deroff(1)
nroff/troff, tbl, and eqn deroff(1)
nul: the null file. null(4)
nul: the null file. null(4)
factor, primes: factor a number, generate large primes.
factor(1)
factor(1)
nl: line
graphics: access graphical and graphics(1G)
numerical commands.
find ordering relation for an object file. size(1)
find ordering relation for an object library. lorder(1)
find ordering relation for an object library. lorder(1)
sky: obtain ephemeredes. sky(6)
oct: octal dump. od(1)
oct: octal dump. od(1)
dp: duplicate an open file descriptor. dup(2)
open a stream. foppen(3S)
dp: duplicate an open file descriptor. dup(2)
dp: duplicate an open file descriptor. dup(2)
dp: duplicate an open file descriptor. dup(2)
dp: duplicate an open file descriptor. dup(2)
open a stream. foppen(3S)
<table>
<thead>
<tr>
<th>subroutines</th>
<th>plot: graphics interface</th>
<th>plot(3X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>images</td>
<td>pch: file format for card</td>
<td>pch(5)</td>
</tr>
<tr>
<td>lseek: move read/write file</td>
<td>pointer</td>
<td>lseek(2)</td>
</tr>
<tr>
<td>to/from a process</td>
<td></td>
<td></td>
</tr>
<tr>
<td>basename, dirname: deliver</td>
<td>portions of path names</td>
<td>basename(1)</td>
</tr>
<tr>
<td>banner: make</td>
<td>posters</td>
<td>banner(1)</td>
</tr>
<tr>
<td>logarithm, power, exp, log,</td>
<td>pow, sqrt: exponential,</td>
<td>exp(3M)</td>
</tr>
<tr>
<td>sqrt: exponential, logarithm,</td>
<td>power, square root functions.</td>
<td>exp(3M)</td>
</tr>
<tr>
<td>for troff. cw, checkcw:</td>
<td>prepare constant-width text</td>
<td>cw(1)</td>
</tr>
<tr>
<td>monitor:</td>
<td>prepare execution profile</td>
<td>monitor(3C)</td>
</tr>
<tr>
<td>unget: undo a</td>
<td>previous get of an SCCS file.</td>
<td>unget(1)</td>
</tr>
<tr>
<td>profiler.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>operating/ pfld, pfstat,</td>
<td>prf: operating system</td>
<td>prf(4)</td>
</tr>
<tr>
<td>prfsnap, prfpr: operating/</td>
<td>prfdc, prfsnap, prfpr:</td>
<td>prf(1M)</td>
</tr>
<tr>
<td>/prfsnap, prfpr: operating/</td>
<td>prfdc, prfsnap, prfdc,</td>
<td>prf(1M)</td>
</tr>
<tr>
<td>system/ pfld, pfstat, prfdc,</td>
<td>prfsnap, prfpr: operating/</td>
<td>prf(1M)</td>
</tr>
<tr>
<td>prfpr: operating/ pfld,</td>
<td>prfsnap, prfpr: operating/</td>
<td>prf(1M)</td>
</tr>
<tr>
<td>generate large/ factor,</td>
<td>prfsnap, prfpr: operating/</td>
<td>prf(1M)</td>
</tr>
<tr>
<td>a number, generate large</td>
<td>primes: factor a number,</td>
<td>factor(1)</td>
</tr>
<tr>
<td>graphical/ gps: graphical</td>
<td>primes. /primes: factor</td>
<td>factor(1)</td>
</tr>
<tr>
<td>types:</td>
<td>primitive string, format of</td>
<td>gps(5)</td>
</tr>
<tr>
<td>prs:</td>
<td>primitive system data types.</td>
<td>types(7)</td>
</tr>
<tr>
<td>date:</td>
<td>print an SCCS file.</td>
<td>prs(1)</td>
</tr>
<tr>
<td>cal:</td>
<td>print and set the date.</td>
<td>date(1)</td>
</tr>
<tr>
<td>editing activity. sact:</td>
<td>print calendar.</td>
<td>cal(1)</td>
</tr>
<tr>
<td>dpr: off-line</td>
<td>print.</td>
<td>dpr(1C)</td>
</tr>
<tr>
<td>man:</td>
<td>print entries in this manual.</td>
<td>man(1)</td>
</tr>
<tr>
<td>cat: concatenate and</td>
<td>print files.</td>
<td>cat(1)</td>
</tr>
<tr>
<td>pr:</td>
<td>print files.</td>
<td>pr(1)</td>
</tr>
<tr>
<td>mm:</td>
<td>print name list.</td>
<td>nm(1)</td>
</tr>
<tr>
<td>uname:</td>
<td>print name of current UNIX.</td>
<td>uname(1)</td>
</tr>
<tr>
<td>news:</td>
<td>print news items.</td>
<td>news(1)</td>
</tr>
<tr>
<td>with the MM macros.</td>
<td>print out documents formatted</td>
<td>mm(1)</td>
</tr>
<tr>
<td>file(s), acctcom: search and</td>
<td>print process accounting</td>
<td>acctcom(1)</td>
</tr>
<tr>
<td>names. id</td>
<td>print user and group IDs and</td>
<td>id(1)</td>
</tr>
<tr>
<td>vpmtrace: load the KMC11-B;</td>
<td>print VPM traces.</td>
<td>vpmstart(1C)</td>
</tr>
<tr>
<td>sending daemons, line</td>
<td>/HONEYWELL</td>
<td></td>
</tr>
<tr>
<td>lp: line</td>
<td>printer daemon.</td>
<td></td>
</tr>
<tr>
<td>lpr: line</td>
<td>/HONEYWELL</td>
<td></td>
</tr>
<tr>
<td>vpm: Versatec</td>
<td>printer.</td>
<td>lp(4)</td>
</tr>
<tr>
<td>vpr: Versatec</td>
<td>printer spooler.</td>
<td>lpr(1)</td>
</tr>
<tr>
<td>output formatters.</td>
<td>printer spooler.</td>
<td>vpr(1)</td>
</tr>
<tr>
<td>nice: run a command at low</td>
<td>printer.</td>
<td>vpr(4)</td>
</tr>
<tr>
<td>nice: change priority</td>
<td>priority.</td>
<td></td>
</tr>
<tr>
<td>errors. erpt:</td>
<td>priority of a process.</td>
<td></td>
</tr>
<tr>
<td>acct: enable or disable</td>
<td>process accounting.</td>
<td>acct(2)</td>
</tr>
<tr>
<td>acctprc:</td>
<td>process accounting.</td>
<td>acctprc(1M)</td>
</tr>
<tr>
<td>acctcom: search and print</td>
<td>process accounting file(s).</td>
<td>acctcom(1)</td>
</tr>
<tr>
<td>times. init:</td>
<td>process and child process.</td>
<td>times(2)</td>
</tr>
<tr>
<td>exit: terminate</td>
<td>process control.</td>
<td>init(8)</td>
</tr>
<tr>
<td>fork: create a new process</td>
<td>process.</td>
<td>exit(2)</td>
</tr>
<tr>
<td>/getpgrp, getppid: get process,</td>
<td>process group, and parent/</td>
<td>getppid(2)</td>
</tr>
<tr>
<td>setpgrp: set process group ID.</td>
<td>get process,</td>
<td>getppid(2)</td>
</tr>
<tr>
<td>process group, and parent</td>
<td>get process group ID.</td>
<td>getppid(2)</td>
</tr>
<tr>
<td>kill: terminate a</td>
<td>process.</td>
<td>kill(1)</td>
</tr>
<tr>
<td>nice: change priority of a</td>
<td>process.</td>
<td>kill(2)</td>
</tr>
<tr>
<td>kill: send a signal to a</td>
<td>process or a group of/</td>
<td>kill(2)</td>
</tr>
<tr>
<td>pclose: initiate 1/O to/from a</td>
<td>process.</td>
<td>nice(2)</td>
</tr>
<tr>
<td>getppid: get process group, and/</td>
<td>process.</td>
<td>nice(2)</td>
</tr>
<tr>
<td>ps: report</td>
<td>process status.</td>
<td>ps(1)</td>
</tr>
<tr>
<td>times: get process and child</td>
<td>process times.</td>
<td>times(2)</td>
</tr>
</tbody>
</table>
wait: wait for child
ptrace: process trace.
pause: suspend
wait: await completion of
list of file systems
to a process or a group of
awk: pattern scanning and
shutdown: terminate all
m4: macro
alarm: set a
profile.
prof: display profile data.
monitor: prepare execution
profil: execution time
environment at login time.
prf: operating system
profpr: operating system
shell: the standard command
link with built-in DDCMP
vpm: The Virtual
vpmc: compiler for the virtual
arithmetic: provide drill in number facts.
true, false: provide truth values.
rand, srand: random number generator.
ratfor: rational Fortran
dialect.
ratfor: rational Fortran dialect.
shell script.
getpass: read a password.
read: read from file.
rmail: send mail to users or
line: read one line.
open: open for
iseek: move
allocator: malloc, free, specify what to do upon
from per-process accounting
errdead: extract error
wtmpfix: manipulate wtmp
xref: cross
compile/execute, regex.
make: maintain, update, and
compile: regular expression
regcmp: regular expression
regcmp: regular expression
regex: regular expression
regcmp: regular expression
reform: reform text file.
reform: reform text file.
reform: regular expression
reject: regular expression
reform: groups of programs.
exec: execute.
wait: wait for child
ptrace: process trace.
pause: suspend
wait: await completion of
list of file systems
to a process or a group of
awk: pattern scanning and
shutdown: terminate all
m4: macro
alarm: set a
profile.
prof: display profile data.
monitor: prepare execution
profil: execution time
environment at login time.
prf: operating system
profpr: operating system
shell: the standard command
link with built-in DDCMP
vpm: The Virtual
vpmc: compiler for the virtual
arithmetic: provide drill in number facts.
true, false: provide truth values.
rand, srand: random number generator.
ratfor: rational Fortran
dialect.
ratfor: rational Fortran dialect.
shell script.
getpass: read a password.
read: read from file.
rmail: send mail to users or
line: read one line.
open: open for
iseek: move
allocator: malloc, free, specify what to do upon
from per-process accounting
errdead: extract error
wtmpfix: manipulate wtmp
xref: cross
compile/execute, regex.
make: maintain, update, and
expression compile/execute. regex, regcmp: regular expression compile and
regcmp: regular expression compile. ... regcmp(1)
compile and match routines. regexp: regular expression compile... regexp(7)
match routines. regexp: regular expression compile and... regexp(7)
regcmp: regular expression compile. ... regcmp(1)
regex, regcmp: regular expression compile and... regcmp(1)
sorted files. comm: select or reject lines common to two... comm(1)
join: relational database operator. ... join(1)
strip: remove symbols and relocation bits. ... strip(1)
value, floor, ceiling, remainder functions. /absolute floor(3M)
commands. mk: how to remake the system and... mk(8)
calendar: reminder service. ... calendar(1)
hasp: RJE (Remote Job Entry) to IBM. ... hasp(8)
rl: RJE (Remote Job Entry) to IBM. ... rje(8)
unlink: remove directory entry. ... unlink(2)
rm, rmdir: remove files or directories. ... rm(1)
eqn constructs. deroff: remove nroff/troff, tbl, and... deroff(1)
bits. strip: remove symbols and relocation... strip(1)
check and interactive repair. /system consistency... fsck(1M)
uniq: report repeated lines in a file. ... uniq(1)
console. rjestat: RJE status and enquiries. ... rjestat(1C)
blocks. df: report number of free disk... df(1)
erpt: process a report of logged errors. ... errpt(1M)
sar: system activity report package. ... sar(8)
ps: report process status. ... ps(1)
file. uniq: report repeated lines in a file... uniq(1)
and generate a system activity report. timex: time a command... timex(1)
fseek, ftell, rewind: reposition a stream. ... fseek(3S)
system restore. restor: incremental file... restor(1M)
incremental file system restore. restor:... restor(1M)
interpreter). rsh: restricted shell (command... rsh(1)
fsck (system consistency check and interactive repair. /system consistency... fsck(1M)
uniq: report repeated lines in a file. ... uniq(1)
fget.demon, fget.demon: file retrieval daemons. ... fget.demon(1C)
HONEYWELL 6000. fget: retrieve files from the file retrieval daemons. ... fget(1C)
stat: data returned by stat system call. ... stat(7)
reversi: a game of dramatic reversals. ... reversi(6)
reversals. ... reversi(6)
col: filter reverse line-feeds... col(1)
col: filter reverse line-feeds... col(1)
reversals. ... reversi(6)
create a new file or... create(2)
file. file:... file:... file:... file:... file:
rf: RF11/RS11 fixed-head disk... rf(4)
RF11/RS11 fixed-head disk... rf(4)
RF11/RS11 fixed-head disk... rf(4)
RF11/RS303-RJS04 fixed-head disk... hs(4)
RF11/RS303-RJS04 fixed-head disk... hs(4)
JRE jobs. send, gather:... send(1C)
JRE jobs. send, gather:... send(1C)
hasp: JRE (Remote Job Entry) to IBM. ... hasp(8)
rl: RL-11/RL01 disk. ... rl(4)
rl: RL-11/RL01 disk. ... rl(4)
read mail. mail: send mail to users or... mail(1)
rm, rm: remove files or... rm(1)
rm: remove files or... rm(1)
SCCS files. rm: remove files or... rm(1)
SCCS files. rm: remove files or... rm(1)
directories. rm, rmdir: remove files or... rm(1)
directories. rm, rmdir: remove files or... rm(1)
romboot: special ROM bootstrap loaders. ... romboot(8)
loaders.

chroot: change

logarithm, power, square

 económet, lg: graphical device

expression compile and match

 graphic table of contents

 disk.

 moving-head disk. hp:

 rp6fmt: format and/or check

 rp: RP-11/RP03 moving-head disk.

 interpreter).

 nice:

 chroot: change root directory.

 log: change root directory for a command.

 nice: run a command at low priority.

 hangups and quits. nohup:

 runacct:

 editing activity.

 package.

 space allocation. brk:

 formatted input conversion.

 bft: big file

 language. awk: pattern

 stand-alone programs.

 the delta commentary of an

 comb: combine

 make a delta (change) to an

 sact: print current

 get: get a version of an

 prs: print an

 rmful: remove a delta from an

 compare two versions of an

 sccsfie: format of

 undo a previous get of an

 val: validate

 admin: create and administer

 what: identify

 of an SCCS file.

 system initialization shell

 program.

 grep, egrep, fgrep:

 accounting file(s). acctcom:

 isearch: linear

 bsearch: binary

 brk, sbbrk: change data

to two sorted files. comm:

greek:

 of a file. cut: cut out

 a group of processes. kill:

 6000. fsend:

 and/or submit RJE jobs.

gosmail:

 mail. mail, rmail:

 the HONEYWELL 6000. gcat:

dpd, odpd, lpd: HONEYWELL

 stream.

 IDs. setuid, getgrent, getgrnam,

 gos

 romboot: special ROM bootstrap

 root directory.

 root directory for a command.

 root functions. /exponential,

 routines and filters.

 routines. regexp: regular

 routines. toc:

 rp: RP-11/RP03 moving-head disk.

 rp6fmt: format and/or check

 rsh: restricted shell (command

 nice: run a command at low priority.

 nohup: run a command immune to

 runacct: run daily accounting.

 runacct: run daily accounting.

 runacct: run daily accounting.

 get: get a version of an

 prs: print an

 rmdel: remove a delta from an

 compare two versions of an

 sccsfie: format of

 undo a previous get of an

 val: validate

 admin: create and administer

 what: identify

 of an SCCS file.

 system initialization shell

 program.

 grep, egrep, fgrep:

 accounting file(s). acctcom:

 isearch: linear

 bsearch: binary

 brk, sbbrk: change data

to two sorted files. comm:

greek:

 of a file. cut: cut out

 a group of processes. kill:

 6000. fsend:

 and/or submit RJE jobs.

gosmail:

 mail. mail, rmail:

 the HONEYWELL 6000. gcat:

dpd, odpd, lpd: HONEYWELL

 stream.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>getpwent, getpwpuid, getpwnam</td>
<td>Return information about a user.</td>
<td>[3C]</td>
</tr>
<tr>
<td>login time, profile</td>
<td>Establish a new environment for a user.</td>
<td>[5]</td>
</tr>
<tr>
<td>command programming language</td>
<td>Shell command interpreter.</td>
<td>[3C]</td>
</tr>
<tr>
<td>rsh: restricted</td>
<td>Set process group ID.</td>
<td>[2]</td>
</tr>
<tr>
<td>system: issue a system</td>
<td>Send a signal to a system.</td>
<td>[3C]</td>
</tr>
<tr>
<td>accounting. acctsh:</td>
<td>Shell procedures for accounting.</td>
<td>[3S]</td>
</tr>
<tr>
<td>sh: shell, the standard</td>
<td>Set user and group IDs.</td>
<td>[1M]</td>
</tr>
<tr>
<td>setuid, setgid: setuser</td>
<td>Set process group ID.</td>
<td>[1M]</td>
</tr>
<tr>
<td>rsh: restricted</td>
<td>Restricted shell command.</td>
<td>[1M]</td>
</tr>
<tr>
<td>login: sign on</td>
<td>Shell script.</td>
<td>[8]</td>
</tr>
<tr>
<td>pause: suspend process</td>
<td>Suspend execution for a process until a signal.</td>
<td>[2]</td>
</tr>
<tr>
<td>what to do upon receipt of a signal</td>
<td>Specify what to do upon receipt of a signal.</td>
<td>[2]</td>
</tr>
<tr>
<td>of processes. kill: send a signal</td>
<td>Send a signal to a process or a group.</td>
<td>[2]</td>
</tr>
<tr>
<td>lex: generate programs for programming language</td>
<td>Lexical analysis and generation of code.</td>
<td>[1]</td>
</tr>
<tr>
<td>'&amp;': compound operator</td>
<td>Logical operator.</td>
<td>[2]</td>
</tr>
<tr>
<td>qsort: quicker</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[1]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>fspec: format</td>
<td>Size of an object file.</td>
<td>[1M]</td>
</tr>
<tr>
<td>slept: sleep</td>
<td>Sleep: suspend execution for a process until a signal.</td>
<td>[3C]</td>
</tr>
<tr>
<td>diameter</td>
<td>Size of a file.</td>
<td>[1M]</td>
</tr>
<tr>
<td>slides: mmt, mvt: typeset</td>
<td>Spline: interpolate smooth curve.</td>
<td>[1G]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>qsort: quicker</td>
<td>Sort and/or merge files.</td>
<td>[1]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
<tr>
<td>brk, sbkr: change data segment</td>
<td>Change data segment.</td>
<td>[2]</td>
</tr>
<tr>
<td>flex: format</td>
<td>Specification in text files.</td>
<td>[5]</td>
</tr>
<tr>
<td>ssignal, gsignal: software</td>
<td>Software signals for communication.</td>
<td>[3C]</td>
</tr>
<tr>
<td>sno: SNOBOL interpreter</td>
<td>SNOBOL interpreter.</td>
<td>[1]</td>
</tr>
<tr>
<td>sort: sort and/or merge files</td>
<td>Sort and/or merge files.</td>
<td>[3C]</td>
</tr>
<tr>
<td>tsort: topological sort</td>
<td>Sort files.</td>
<td>[1]</td>
</tr>
</tbody>
</table>
package. stdio: standard buffered input/output
language. sh: shell, the
unixboot: UNIX system call.
useful with graphical/: stat: data returned by with graphical/
stat: data returned by with graphical/
ustat: get file system
rjestat: RJE status report and interactive
status report and interactive
ferror, clearerr, fileno: stream control. ustat: uucp
ps: report process
status console. rjestat: RJE status inquiry and job
status console. rjestat: RJE status inquiry and job
stat, fsstat: get file
input/output package.
stdio: standard buffered
ftime: set time.
wait for child process to
name: stop or terminate. wait:
strncmp, strcpy, strncpy, strncpy,
strncpy, strncpy, strlncpy, strlncpy,
strncpy, strlncpy, strlncpy,
strncpy, strlncpy, strlncpy,
strncpy, strlncpy, strlncpy,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
strlen, strchr, strrchr, strstr,
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>pause</td>
<td>Suspend process until signal.</td>
<td>pause(2)</td>
</tr>
<tr>
<td>swab</td>
<td>Swap bytes.</td>
<td>swab(3C)</td>
</tr>
<tr>
<td>sdb</td>
<td>Symbolic debugger.</td>
<td>sdb(1)</td>
</tr>
<tr>
<td>strip: remove</td>
<td>Symbols and relocation bits.</td>
<td>strip(1)</td>
</tr>
<tr>
<td>sync</td>
<td>Update super-block.</td>
<td>sync(2)</td>
</tr>
<tr>
<td>sync</td>
<td>Update the super block.</td>
<td>sync(1M)</td>
</tr>
<tr>
<td>du: DU-11</td>
<td>Synchronous line interface.</td>
<td>du(4)</td>
</tr>
<tr>
<td>st:</td>
<td>Synchronous terminal control.</td>
<td>st(1M)</td>
</tr>
<tr>
<td>interface: st</td>
<td>Synchronous terminal</td>
<td>st(4)</td>
</tr>
<tr>
<td>sysdef:</td>
<td>System definition.</td>
<td>sysdef(1M)</td>
</tr>
<tr>
<td>perror, sys_errlist, sys_nerr, errno:</td>
<td>System error messages</td>
<td>perror(3C)</td>
</tr>
<tr>
<td>master</td>
<td>Master device information table.</td>
<td>master(5)</td>
</tr>
<tr>
<td>mnttab:</td>
<td>Mounted file system table.</td>
<td>mnttab(5)</td>
</tr>
<tr>
<td>toc: graphical</td>
<td>Table of contents routines.</td>
<td>toc(1G)</td>
</tr>
<tr>
<td>tbl: format</td>
<td>Tables for nroff or troff.</td>
<td>tbl(1)</td>
</tr>
<tr>
<td>tabs: set</td>
<td>Tabs on a terminal.</td>
<td>tabs(1)</td>
</tr>
<tr>
<td>tail:</td>
<td>Deliver the last part of a file.</td>
<td>tail(1)</td>
</tr>
<tr>
<td>trigonometric/ sin, cos, sinh, cosh, tan, asin, acos, atan, atan2:</td>
<td>Hyperbolic functions</td>
<td>sinh(3M)</td>
</tr>
<tr>
<td>tape archive.</td>
<td>Tape archive.</td>
<td>tapearchive(1)</td>
</tr>
<tr>
<td>tape bootstrap program.</td>
<td>Tape bootstrap program.</td>
<td>tapeboot(8)</td>
</tr>
<tr>
<td>tar:</td>
<td>Tape file archiver.</td>
<td>tar(1)</td>
</tr>
<tr>
<td>dump:</td>
<td>Incremental dump.</td>
<td>dump(5)</td>
</tr>
<tr>
<td>tp:</td>
<td>Tape file system backup utility.</td>
<td>tp(1)</td>
</tr>
<tr>
<td>ht: TU16 magnetic</td>
<td>Magnetic tape archive.</td>
<td>ht(4)</td>
</tr>
<tr>
<td>tm:</td>
<td>TM11/TU10 magnetic bootstrap program.</td>
<td>tm(4)</td>
</tr>
<tr>
<td>file system backup, filesave,</td>
<td>Tapesave: daily/weekly UNIX</td>
<td>filesave(8)</td>
</tr>
<tr>
<td>tbl:</td>
<td>Format tables for nroff or troff.</td>
<td>tbl(1)</td>
</tr>
<tr>
<td>tables for nroff or troff.</td>
<td>Tbl, and eqn constructs.</td>
<td>deroff(1)</td>
</tr>
<tr>
<td>tee: pipe fitting.</td>
<td>Tees: terminal name.</td>
<td>tee(1)</td>
</tr>
<tr>
<td>td:</td>
<td>Graphical device routines/tee:</td>
<td>gdev(1G)</td>
</tr>
<tr>
<td>tekset</td>
<td>Tektronix 4014 terminal.</td>
<td>4014(1)</td>
</tr>
<tr>
<td>tmpfile:</td>
<td>Temporary file.</td>
<td>tmpfile(3S)</td>
</tr>
<tr>
<td>tmplam:</td>
<td>Create a name for a temporary file.</td>
<td>tmplam(3S)</td>
</tr>
<tr>
<td>term:</td>
<td>Terminal's name.</td>
<td>term(7)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal control.</td>
<td>st(1M)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal control.</td>
<td>ct(1C)</td>
</tr>
<tr>
<td>generate file name for greek: select</td>
<td>Create a file name for greek: select</td>
<td>ctermid(3S)</td>
</tr>
<tr>
<td>terminal filter.</td>
<td>Terminal filter.</td>
<td>greek(1)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal.</td>
<td>st(4)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal interface.</td>
<td>tty(4)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal interface.</td>
<td>stty(1)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal.</td>
<td>tabs(1)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal.</td>
<td>tty(1)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal.</td>
<td>kill(1)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal all processing.</td>
<td>shutdown(1M)</td>
</tr>
<tr>
<td>terminal:</td>
<td>Terminal process.</td>
<td>exit(2)</td>
</tr>
<tr>
<td>daemon:</td>
<td>Terminate the error logging</td>
<td>errstop(1M)</td>
</tr>
</tbody>
</table>

**Note:** The commands and descriptions are extracted from the permuted index, which may not follow the standard order and format of a typical index.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>for child process to stop</td>
<td>terminate.</td>
</tr>
<tr>
<td>command</td>
<td>wait. wait</td>
</tr>
<tr>
<td>etp: Equipment</td>
<td>Test Package.</td>
</tr>
<tr>
<td>quiz</td>
<td>test your knowledge.</td>
</tr>
<tr>
<td>ed:</td>
<td>text editor.</td>
</tr>
<tr>
<td>reform: reform</td>
<td>text file.</td>
</tr>
<tr>
<td>fspec: format specification</td>
<td>text files.</td>
</tr>
<tr>
<td>/checkeq: format mathematical</td>
<td>text for nroff or troff.</td>
</tr>
<tr>
<td>prepare constant-width</td>
<td>text for troff. cw, checkcwn:</td>
</tr>
<tr>
<td>nroff: typeset or format</td>
<td>troff(1)</td>
</tr>
<tr>
<td>ttt, cubic:</td>
<td>tic-tac-toe.</td>
</tr>
<tr>
<td>system activity/ timex:</td>
<td>time a command and generate a time.</td>
</tr>
<tr>
<td>time:</td>
<td>time a command.</td>
</tr>
<tr>
<td>profil: execution</td>
<td>time: get time.</td>
</tr>
<tr>
<td>up an environment at login</td>
<td>time. time.</td>
</tr>
<tr>
<td>stime: set</td>
<td>time: time a command.</td>
</tr>
<tr>
<td>time: get</td>
<td>time.</td>
</tr>
<tr>
<td>tzset: convert date and</td>
<td>time to ASCII. /asctime.</td>
</tr>
<tr>
<td>process times.</td>
<td>times: get process and child times.</td>
</tr>
<tr>
<td>update access and modification</td>
<td>times of a file. touch.</td>
</tr>
<tr>
<td>get process and child process</td>
<td>times: times.</td>
</tr>
<tr>
<td>file access and modification</td>
<td>times: utime: set.</td>
</tr>
<tr>
<td>generate a system activity/</td>
<td>timex: time a command and</td>
</tr>
<tr>
<td>interface.</td>
<td>interface. tm: TM11/TU10 magnetic tape</td>
</tr>
<tr>
<td>interface. file.</td>
<td>tmpfile: create a temporary file.</td>
</tr>
<tr>
<td>temporary file.</td>
<td>tmppname: create a name for a toascii: character/</td>
</tr>
<tr>
<td>toupper, tolower, contents</td>
<td>toascii: character/ conv(3C)</td>
</tr>
<tr>
<td>routines.</td>
<td>toascii: character/ conv(3C)</td>
</tr>
<tr>
<td>popen, pclose: initiate I/O</td>
<td>translation. toupper, toascii: character/ conv(3C)</td>
</tr>
<tr>
<td>translation. toupper,</td>
<td>tp: magnetic tape format.</td>
</tr>
<tr>
<td>toascii: character/</td>
<td>tp: manipulate tape archive.</td>
</tr>
<tr>
<td>contents routines.</td>
<td>tpplot: graphics filters.</td>
</tr>
<tr>
<td>toupper, tolower,</td>
<td>tr: translate characters.</td>
</tr>
<tr>
<td>contents routines.</td>
<td>trace: event-tracing driver.</td>
</tr>
<tr>
<td>toupper, tolower,</td>
<td>ttyname(3C)</td>
</tr>
<tr>
<td>character translation.</td>
<td>tt: general terminal</td>
</tr>
<tr>
<td>load the KMC11-B; print VM</td>
<td>tty: get the terminal's name.</td>
</tr>
<tr>
<td>ptrace: process</td>
<td>trajectories. vpmsnap, vpmtrace:</td>
</tr>
<tr>
<td>tr:</td>
<td>translate characters.</td>
</tr>
<tr>
<td>tolower, toascii: character</td>
<td>translation. toupper,</td>
</tr>
<tr>
<td>tan, asin, acos, atan, atan2</td>
<td>trig(3M)</td>
</tr>
<tr>
<td>constant-width text for</td>
<td>troff. cw, checkcwn: prepare</td>
</tr>
<tr>
<td>mathematical text for nroff</td>
<td>troff. /neq, checkeq: format.</td>
</tr>
<tr>
<td>or format text.</td>
<td>troff, nroff: typeset or</td>
</tr>
<tr>
<td>format tables for nroff or</td>
<td>troff. tbl:</td>
</tr>
<tr>
<td>values.</td>
<td>tty: get the terminal's name.</td>
</tr>
<tr>
<td>true, false: provide</td>
<td>tt: general terminal</td>
</tr>
<tr>
<td>interface.</td>
<td>tty: get the terminal's name.</td>
</tr>
<tr>
<td>graphics for the extended</td>
<td>tty: get the terminal's name.</td>
</tr>
<tr>
<td>a terminal.</td>
<td>tt: general terminal</td>
</tr>
<tr>
<td>ht:</td>
<td>tty: get the terminal's name.</td>
</tr>
<tr>
<td>dqa: DQS-11 interface for</td>
<td>tt: general terminal</td>
</tr>
<tr>
<td>file: determine file</td>
<td>tty: get the terminal's name.</td>
</tr>
<tr>
<td>for the extended TTY-37</td>
<td>tty: get the terminal's name.</td>
</tr>
<tr>
<td>- 29 -</td>
<td>- 29 -</td>
</tr>
</tbody>
</table>
types: primitive system data

graphs, and slides. mmt, mvt: typset documents, view

troff, nroff: typeset or format text.
typo: find possible typographical errors.
typo: find possible

//localtime, gmtime, asctime,
getpw: get name from UID.

ulimit: get and set user limits.
creation mask. umask: set and get file mask.
file system. mount, umount: mount and dismount

UNIX system.
UNIX.
KMC11/DMC11/ kun: un-assembler for the

file. unget: undo a previous get of an SCSS unget(1)

an SCSS file.

into input stream. unget: push character back ungetc(3S)
a file.

mktemp: make a unique file name.

units: conversion program.

RJE (Remote Job Entry) to UNIVAC. uvac: 

boot procedures. unixboot: UNIX startup and unixboot(8)

uto, uupick: public UNIX-to-UNIX file copy. uuto(1C)

unlink system calls. link, entry. unlink: exercise link and unlink(2)

unlink: exercise link and unlink system calls. link, unlink(1M)

umount: unmount a file system.

files. pack. pcat, times of a file. touch: update access and modification

touch(1)

of programs. make: maintain, update, and regenerate groups

make(1)

lsearch: linear search and update.

lssearch(3C)

sync: update super-block.

sync(2)

sync: update the super block.

sync(1M)

du: summarize disk usage.

du(1)

delimiters. mmchek: check usage of mm macros and eqn

mmchek(1)

stat: statistical network useful with graphical/

stat(1G)

id: print user and group IDs and names.
id(1)

setuid, setgid: set user and group IDs.

setuid(2)

character login name of the user. cuserid: 
cuserid(3S)

/getgid, getegid: get real user and effective user, real/

environ: user environment.
environ(7)

gcosmail: send mail to HIS user.
gcosmail(1C)

ulimit: get and set user limits.

ulimit(2)

logname: login name of user.

logname(3X)

/get real user, effective user, real group, and/

getuid(2)

become super-user or another user. su:
su(1)

write: write to another user.

write(1)

mail, rmail: send mail to users or read mail.

mail(1)

wall: write to all users.

wall(1M)

stat: statistical network.

stat(1G)

gutil: graphical utilities.
gutil(1G)

modification times.

utime: set file access and

utime(2)

utmp, wttmp: entry format.

utmp and wttmp entry format.

utmp(5)

wc: word count utilities.

wc(1G)

utmp: time stamp.

utmp(1M)

uucp: monitor uucp spool directory clean-up.
uucp(1M)

uucp: uucp network.
uucp(1M)

uucp: uucp spool directory clean-up.
uucp(1M)

uucp status inquiry and job

uucp(1C)

unix copy. uucp, ulog, uuname: unix to uucp(1C)

copy. uucp, ulog, uuname: unix to uucp(1C)

uuname: unix to unix copy. uucp(1C)
<table>
<thead>
<tr>
<th>File Copy</th>
<th>uuto: public UNIX-to-UNIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>and Job Control</td>
<td>uupick: public UNIX-to-UNIX</td>
</tr>
<tr>
<td></td>
<td>uustat: uucp status inquiry</td>
</tr>
<tr>
<td></td>
<td>uusub: monitor uucp network</td>
</tr>
<tr>
<td>UNIX-to-UNIX File Copy</td>
<td>uuto: public UNIX-to-UNIX</td>
</tr>
<tr>
<td>Execution</td>
<td>uux: unix to unix command</td>
</tr>
<tr>
<td>To UNIVAC</td>
<td>uvac: RJE (Remote Job Entry)</td>
</tr>
<tr>
<td></td>
<td>val: validate SCCS file</td>
</tr>
<tr>
<td></td>
<td>val: validate SCCS file</td>
</tr>
<tr>
<td></td>
<td>abs: integer absolute value</td>
</tr>
<tr>
<td></td>
<td>fabs, ceil, fmod: absolute</td>
</tr>
<tr>
<td></td>
<td>getenv: value for environment name</td>
</tr>
<tr>
<td></td>
<td>true, false: provide truth values</td>
</tr>
<tr>
<td></td>
<td>as: assembler for VAX-11/780</td>
</tr>
<tr>
<td></td>
<td>vaxops: VAX-11/780 console operations</td>
</tr>
<tr>
<td></td>
<td>archive files from PDP-11 to VAX-11/780</td>
</tr>
<tr>
<td>Interface</td>
<td>vxl: VAX-11/780 LSI console floppy</td>
</tr>
<tr>
<td></td>
<td>systems with label checking</td>
</tr>
<tr>
<td></td>
<td>file system: format of system volume</td>
</tr>
<tr>
<td>Assert Program</td>
<td>assert: program verification</td>
</tr>
<tr>
<td></td>
<td>vpr: Versatec printer spooler</td>
</tr>
<tr>
<td></td>
<td>vp: Versatec printer</td>
</tr>
<tr>
<td></td>
<td>vc: version control</td>
</tr>
<tr>
<td>Get</td>
<td>get: get a version of an SCCS file</td>
</tr>
<tr>
<td></td>
<td>sccsdiff: compare two versions of an SCCS file</td>
</tr>
<tr>
<td></td>
<td>mmt, mvt: typeset documents</td>
</tr>
<tr>
<td></td>
<td>mv: a macro package for making</td>
</tr>
<tr>
<td></td>
<td>vpm: The Virtual Protocol Machine</td>
</tr>
<tr>
<td></td>
<td>vpmc: compiler for the virtual machine</td>
</tr>
<tr>
<td>Floppy Interface</td>
<td>vxl: VAX-11/780 LSI console floppy</td>
</tr>
<tr>
<td></td>
<td>file system: format of system volume</td>
</tr>
<tr>
<td></td>
<td>volcopy: label: copy file</td>
</tr>
<tr>
<td></td>
<td>fs: word count</td>
</tr>
<tr>
<td></td>
<td>signal: specify</td>
</tr>
<tr>
<td></td>
<td>crashes: specify</td>
</tr>
<tr>
<td></td>
<td>signal: what to do upon receipt of a</td>
</tr>
<tr>
<td></td>
<td>crash: what to do when the system</td>
</tr>
<tr>
<td></td>
<td>whodo: who is doing what</td>
</tr>
<tr>
<td></td>
<td>who: who is on the system</td>
</tr>
<tr>
<td></td>
<td>whodo: who is doing what</td>
</tr>
<tr>
<td></td>
<td>cd: change working directory</td>
</tr>
<tr>
<td></td>
<td>chdir: change working directory</td>
</tr>
<tr>
<td></td>
<td>pwd: working directory name</td>
</tr>
<tr>
<td></td>
<td>write: write on a file</td>
</tr>
<tr>
<td></td>
<td>putpwent: write password file entry</td>
</tr>
<tr>
<td></td>
<td>wall: write to all users</td>
</tr>
<tr>
<td></td>
<td>write: write to another user</td>
</tr>
<tr>
<td></td>
<td>open: open for reading or writing</td>
</tr>
<tr>
<td></td>
<td>utmp, wtmp: utmp and wtmp entry format</td>
</tr>
</tbody>
</table>
fwtmp, wtmpfix: manipulate wtmp records.
format. utmp, wtmp: utmp and wtmp entry
records. fwtmp, wtmpfix: manipulate wtmp
hunt-the-wumpus. wump: the game of
list(s) and execute command. xargs: construct argument
programs. xref: cross reference for C
j0, j1, jn, y0, yl: bessel functions.
j0, j1, jn, y0, yl: bessel functions.
compiler-compiler. yacc: yet another
j0, j1, jn, y0, yl: bessel functions.

NAME
intro — introduction to commands and application programs

DESCRIPTION
This section describes, in alphabetical order, publicly-accessible commands. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility.
(1C) Commands for communication with other systems.
(1G) Commands used primarily for graphics and computer-aided design.
(1M) Commands used primarily for system maintenance.

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

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

where:

name 
- The name of an executable file.

option
- noargletter(s) or,
- argletter <> optarg

where <> is optional white space.

noargletter
- A single letter representing an option without an argument.

argletter
- A single letter representing an option requiring an argument.

optarg
- Argument (character string) satisfying preceding argletter.

cmdarg
- Path name (or other command argument) not beginning with — or, — by itself indicating the standard input.

SEE ALSO
getopt(1), getopt(3C).
Section 6 of this volume for computer games.
How to Get Started, at the front of this volume.

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.

BUGS
Regretfully, many commands do not adhere to the aforementioned syntax.
NAME
300, 300s — handle special functions of DASI 300 and 300s terminals

SYNOPSIS
300 [ +12 ] [ −n ] [ −dt,l,c ]
300s [ +12 ] [ −n ] [ −dt,l,c ]

DESCRIPTION
300 supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; 300s performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%.

300 can be used to print equations neatly, in the sequence:
neq file ... | nroff 300

WARNING: if your terminal has a PLOT switch, make sure it is turned on before 300 is used.

The behavior of 300 can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

+12 permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the +12 option.

−n controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of n overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, nroff(1) half-lines could be made to act as quarter-lines by using −2. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option −3 alone, having set the PITCH switch to 12-pitch.

−dt,l,c controls delay factors. The default setting is −d3,90,30. DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of t tabs, and for every contiguous string of c non-blank, non-tab characters. If a line is longer than l bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for t (c) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like /etc/passwd. Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output. The −d option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file /etc/passwd may be printed using −d3,30,5. The value −d0,1 is a good one to use for C programs that have many levels of indentation.
Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The *stty* modes **mlo cr2** or **mlo cr3** are recommended for most uses.

**300** can be used with the *nroff* `-s` flag or `.rd` requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

```
nroff -T300 files ... and nroff files ... | 300
nroff -T300-12 files ... and nroff files ... | 300 +12
```

The use of **300** can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of **300** may produce better-aligned output.

The *neqn*(1) names of, and resulting output for, the Greek and special characters supported by **300** are shown in *greek*(7).

**SEE ALSO**

450(1), eqn(1), graph(1G), mesg(1), stty(1), tabs(1), tbl(1), tplot(1G), troff(1), greek(7).

**BUGS**

Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.
NAME
4014 — paginator for the Tektronix 4014 terminal

SYNOPSIS
4014 [ -t ] [ -n ] [ -cN ] [ -pL ] [ file ]

DESCRIPTION
The output of 4014 is intended for a Tektronix 4014 terminal; 4014
arranges for 66 lines to fit on the screen, divides the screen into N
columns, and contributes an eight-space page offset in the (default) single-
column case. Tabs, spaces, and backspaces are collected and plotted when
necessary. TELETYPE® Model 37 half- and reverse-line sequences are inter-
preted and plotted. At the end of each page, 4014 waits for a new-line
(empty line) from the keyboard before continuing on to the next page. In
this wait state, the command lcmd will send the cmd to the shell.

The command line options are:
- - t Don't wait between pages (useful for directing output into a file).
- - n Start printing at the current cursor position and never erase the
  screen.
- - cN Divide the screen into N columns and wait after the last column.
- - pL Set page length to L; L accepts the scale factors i (inches) and l
  (lines); default is lines.

SEE ALSO
pr(1), tc(1), troff(1).
NAME
450 — handle special functions of the DASI 450 terminal

SYNOPSIS
450

DESCRIPTION
450 supports special functions of, and optimizes the use of, the DASI 450 terminal, or any terminal that is functionally identical, such as the DIABLO 1620 or XEROX 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as 300(1). 450 can be used to print equations neatly, in the sequence:

```
neqn file ... | nroff | 450
```

WARNING: make sure that the PLOT switch on your terminal is ON before 450 is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

450 can be used with the `nroff(1) -s` flag or `.rd` requests, when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the use of 450 can be eliminated in favor of one of the following:

```
nroff -T450 files ...
```

or

```
nroff -T450-12 files ...
```

The use of 450 can thus often be avoided unless special delays or options are required; in a few cases, however, the additional movement optimization of 450 may produce better-aligned output.

The `neqn(1)` names of, and resulting output for, the Greek and special characters supported by 450 are shown in `greek(7)`.

SEE ALSO
300(1), eqn(1), graph(1G), msg(1), stty(1), tabs(1), tbl(1), tplot(1G), troff(1), greek(7).

BUGS
Some special characters cannot be correctly printed in column 1 because the print head cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor; although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.
ACCT(1M)

NAME
acct — overview of accounting and miscellaneous accounting commands

SYNOPSIS
acctdisk
acctdisk [ -u file ] [ -p file ] > dtmp-file
accton [file]
acctwtmp [name [line]] >> /usr/adm/wtmp

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

Connect time accounting is handled by various programs that write records into /usr/adm/utmp, as described in utmp(5). The programs described in acctcon(1M) convert this file into session and charging records, which are then summarized by acctmerg(1M).

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

Process accounting and connect time accounting (or any accounting records in the format described in acct(5)) can be merged and summarized into total accounting records by acctmerg (see tacct format in acct(5)). Ptrace(see acctsh(1M)) is used to format any or all accounting records.

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

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

accton alone turns process accounting off. If file is given, it must be the name of an existing file, to which the kernel appends process accounting records (see acct(2) and acct(5)).

acctwtmp writes a wtmp(5) record to its standard output. The record contains the current time, name, and line. If line is omitted, a value is emitted that is interpreted by other programs as a reboot. For more precise accounting, the following are recommended for use in reboot and shutdown procedures, respectively:

acctwtmp `uname` >> /usr/adm/wtmp
acctwtmp reason >> /usr/adm/wtmp

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

- 1 -
SEE ALSO
acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctpre(1M),
acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

The UNIX Accounting System by H. S. McCreary.
NAME
acctcms — command summary from per-process accounting records

SYNOPSIS
acctcms [options] files

DESCRIPTION
acctcms reads one or more files, normally in the form described in acct(5). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The options are:

- a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, and "hog factor", as in acctcom(1). Output is normally sorted by total kcore-minutes.
- c Sort by total CPU time, rather than total kcore-minutes.
- j Combine all commands invoked only once under "---other".
- n Sort by number of command invocations.
- s Any file names encountered hereafter are already in internal summary format.

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

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

SEE ALSO
acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).
ACCTCOM(1)

NAME
acctcom — search and print process accounting file(s)

SYNOPSIS
acctcom [[options][file]] ...

DESCRIPTION
Acctcom reads file, the standard input, or /usr/adm/pacct, in the form
described by acct(5) and writes selected records to the standard output.
Each record represents the execution of one process. The output shows the
COMMAND NAME, USER, TTYYNAME, START TIME, END TIME, REAL
(SEC), CPU (SEC), MEAN SIZE(K), and optionally, F (the fork/exec flag: 1
for fork without exec) and STAT (the system exit status).

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 TTYYNAME field.

If no files are specified, and if the standard input is associated with a ter-
minal 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 in their respective order.
Each file is normally read forward, i.e., 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 files, in which case all but the
current will be found in /usr/adm/pacct?. The options are:

- b Read backwards, showing latest commands first.
- 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 exec-
  ution. This "hog factor" is computed as:
   (total CPU time)/(elapsed time).
- i Print columns containing the I/O counts in the output.
- k Instead of memory size, show total kcore-minutes.
- m Show mean core size (the default).
- r Show CPU factor (user time/(system-time + user-time).
- t Show separate system and user CPU times.
- v Exclude column headings from the output.
- l line Show only processes belonging to terminal /dev/line.
- u user Show only processes belonging to user that may be specified
  by: a user ID, a login name that is then converted to a user ID,
  a $ which designates only those processes executed with
  super-user privileges, or ? which designates only those pro-
  cesses associated with unknown user IDs.

- g group Show only processes belonging to group. The group may be
  designated by either the group ID or group name.
- d mm/dd Any time arguments following this flag are assumed to occur
  on the given month and day, rather than during the last 24
  hours. This is needed for looking at old files.
- s time Show only those processes that existed on or after time, given
  in the form hr:min:sec. The :sec or :min:sec may be omitted.
- e time Show only those processes that existed on 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.


ACCTCOM(1) ACCTCOM(1)

-H factor Show only processes that exceed factor, where factor is the “hog factor” as explained in option -h above.
-O time Show only those processes with operating system CPU time that exceeds time.
-C time Show only those processes that exceed time that indicates the total CPU time.

Listing options together has the effect of a logical and.

FILES
/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO
acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), ps(1), runacct(1M), su(1), acct(2), acct(5), utmp(5).

BUGS
acctcom only reports on processes that have terminated; use ps(1) for active processes.
NAME
acctcon — connect-time accounting

SYNOPSIS
acctcon1 [options]
acctcon2

DESCRIPTION
acctcon1 converts a sequence of login/logoff records read from its standard
input to a sequence of records, one per login session. Its input should nor-
ma lly be redirected from /usr/adm/wtmp. Its output is ASCII, giving dev-
ice, user ID, login name, prime connect time (seconds), non-prime connect
time (seconds), session starting time (numeric), and starting date and time.
The options are:
-p Print input only, showing line name, login name, and time (in
both numeric and date/time formats).
-t Acctcon1 maintains a list of lines on which users are logged in.
When it reaches the end of its input, it emits a session record for
each line that still appears to be active. It normally assumes that
its input is a current file, so that it uses the current time as the
ending time for each session still in progress. The -t flag causes
it to use, instead, the last time found in its input, thus assuring
reasonable and repeatable numbers for non-current files.
-l file File is created to contain a summary of line usage showing line
name, number of minutes used, percentage of total elapsed time
used, number of sessions charged, number of logins, and number
of logoffs. This file helps track line usage, identify bad lines, and
find software and hardware oddities. Both hang-up and termina-
tion of the login shell generate a logoff record, so that the number
of logoffs is often twice the number of sessions.
-o file File is filled with an overall record for the accounting period,
giving starting time, ending time, number of reboots, and number
of date changes.

acctcon2 expects as input a sequence of login session records and converts
them into total accounting records (see tacet format in acct(5)).

EXAMPLES
These commands are typically used as shown below. The file ctmp is
created only for the use of acctprc(1M) commands:
acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct

FILES
/usr/adm/wtmp

SEE ALSO
acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M),
acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

BUGS
The line usage report is confused by date changes. Use wtmpfix (see
fwtmp(1M)) to correct this situation.
NAME
acctmerg — merge or add total accounting files

SYNOPSIS
acctmerg [options] [file] ...

DESCRIPTION
Acctmerg reads its standard input and up to nine additional files, all in the tacct format (see acct(5)), or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys. Options are:

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

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

```
acctmerg -v <file1 >file2
edit file2 as desired ...
acctmerg -a <file2 >file1
```

SEE ALSO
acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).
NAME
acctprc — process accounting

SYNOPSIS
acctprc1 [ctmp]
acctprc2

DESCRIPTION
acctprc1 reads input in the form described by acct(5), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in 64-byte units). If ctmp is given, it is expected to contain a list of login sessions, in the form described in acctcon(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in ctmp helps it distinguish among different login names that share the same user ID.

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

These commands are typically used as shown below:

acctprc1 ctmp < /usr/adm/pacct | acctprc2 > ptacct

FILES
/etc/passwd

SEE ALSO
acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(5), utmp(5).

BUGS
Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from cron(1M), for example. More precise conversion can be done by faking login sessions on the console via the acctwtmp program in acct(1M).
NAME
acctsh — shell procedures for accounting

SYNOPSIS
chargefee login-name number
ckpacct [blocks]
dodisk
lastlogin
monacct number
nulladm file
prctmp
prdaily
prtacct file [ "heading" ]
nrunacct [mmdd] [mmdd state]
shutacct [ "reason" ]
startup
turnacct [ on | off | switch ]

DESCRIPTION
Chargefee is invoked to charge number dollars to login-name. A record is written to /usr/adm/fee, to be merged with other accounting records during the night.

Ckpacct is initiated via cron. It periodically checks the size of /usr/adm/pac:c:t. If the size exceeds blocks, 1000 by default, turnacct will be invoked with argument switch.

Dodisk is invoked by cron to perform the disk accounting functions.

Lastlogin is invoked by runacct to update /usr/adm/ac:t/sum/loginlog, which shows the last date on which each person logged in.

Monacct should be invoked once each month or each accounting period. Number indicates which month or period it is. It creates summary files in /usr/adm/acct/fiscal and restarts summary file in /usr/adm/acct/sum. Nulladm creates file with mode 644 and insures owner is adm. It is called by lastlogin, runacct, and turnacct.

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

Prdaily is invoked by runacct to print a report of the previous day’s accounting. The report resides in /usr/adm/acct/sum/rprtxxxx where xxxx is the month and day of the report. The daily accounting reports may be printed (by the command “cat /usr/adm/acct/sum/rprt*”) as often as desired and they must be explicitly deleted when no longer needed.

Prtacct can be used to format and print any total accounting file.

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

Shutacct should be invoked during a system shutdown to turn process accounting off and append a "reason" record to /usr/adm/wtmp. Startup should be called by rc(8) to turn the accounting on whenever the system is brought up.
**Turnacct** is an interface to *accton* (see *acct*(1M)) to turn process accounting on or off. The switch argument moves the current `/usr/adm/pacct` to the next free name in `/usr/adm/pacct[1-9]`, turns accounting off, then turns it back on again. This procedure is called by *ckpacct* via the *cron* to keep the *pacct* file size smaller.

**FILES**

- `/usr/adm/fee` accumulator for fees
- `/usr/adm/pacct` current file for per-process accounting
- `/usr/adm/pacct[1-9]` used if pacct gets large and during execution of daily accounting procedure
- `/usr/adm/wtmp` login/logoff summary
- `/usr/adm/wtmp[1-9]` used during daily accounting procedure
- `/usr/lib/acct` working directory
- `/usr/lib/acct/nite` holds all accounting commands listed in sub-class 1M of this manual
- `/usr/adm/acct/sum` summary directory, should be saved

**SEE ALSO**

NAME
adb — debugger

SYNOPSIS
adb [-w] [ objfil [ corfil ] ]

DESCRIPTION
adb is a general purpose debugging program. It may be used to examine
files and to provide a controlled environment for the execution of UNIX
programs.

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

Requests to adb are read from the standard input and responses are to the
standard output. If the -w flag is present then both objfil and corfil are
created if necessary and opened for reading and writing so that files can be
modified using adb. adb ignores QUIT; INTERRUPT causes return to the
next adb command.

In general requests to adb are of the form

[ address ] [ , count ] [ command ] ;

If address is present then dot is set to address. Initially dot is set to 0. For
most commands count specifies how many times the command will be exe­
cuted. The default count is 1. Address and count are expressions.

The interpretation of an address depends on the context it is used in. If a
subprocess is being debugged then addresses are interpreted in the usual
way in the address space of the subprocess. For further details of address
mapping see ADDRESSES.

EXPRESSIONS
.
The value of dot.
+
The value of dot incremented by the current increment.
-
The value of dot decremented by the current increment.
*
The last address typed.

integer
An octal number if integer begins with a 0; a hexadecimal number
if preceded by ; otherwise a decimal number.

integer . fraction
A 32 bit floating point number.

'cccc'
The ASCII value of up to 4 characters. \ may be used to escape a ‘.

< name
The value of name, which is either a variable name or a register
name. adb maintains a number of variables (see VARIABLES)
named by single letters or digits. If name is a register name then
the value of the register is obtained from the system header in
corfil. The register names are r0 ... r15 sp pc ps.

symbol
A symbol is a sequence of upper or lower case letters, underscores
or digits, not starting with a digit. The value of the symbol is taken
from the symbol table in objfil. An initial _ or ~ will be prepended
to symbol if needed.

_ symbol
In C, the “true name” of an external symbol begins with _. It may
be necessary to utter this name to distinguish it from internal or
hidden variables of a program.

`routine.name`

The address of the variable `name` in the specified C routine. Both
`routine` and `name` are symbols. If `name` is omitted the value is the
address of the most recently activated C stack frame corresponding
to `routine`.

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

Monadic operators:

- `*exp` The contents of the location addressed by `exp` in `corfil`.
- `@exp` The contents of the location addressed by `exp` in `objfil`.
- `−exp` Integer negation.
- `^exp` Bitwise complement.

Dyadic operators are left associative and are less binding than monadic
operators.

- `e1 + e2` Integer addition.
- `e1 − e2` Integer subtraction.
- `e1 * e2` Integer multiplication.
- `e1 % e2` Integer division.
- `e1 & e2` Bitwise conjunction.
- `e1 | e2` Bitwise disjunction.
- `e1 # e2` `e1` rounded up to the next multiple of `e2`.

**COMMANDS**

Most commands consist of a verb followed by a modifier or list of
modifiers. The following verbs are available. (The commands `?` and `/` may
be followed by `*`; see `ADDRESSES` for further details.)

- `?f` Locations starting at `address` in `objfil` are printed according to the
  format `f`. `dot` is incremented by the sum of the increments for
each format letter (q.v.).
- `/f` Locations starting at `address` in `corfil` are printed according to the
  format `f` and `dot` is incremented as for `?`.
- `−f` The value of `address` itself is printed in the styles indicated by the
  format `f`. (For `i` format `?` is printed for the parts of the instruction
  that reference subsequent words.)

A format consists of one or more characters that specify a style of printing.
Each format character may be preceded by a decimal integer that is a repeat
count for the format character. While stepping through a format `dot` is
incremented by the amount given for each format letter. If no format is
given then the last format is used. The format letters available are as fol­

- `o 2` Print 2 bytes in octal. All octal numbers output by `adb` are
  preceded by `0`.
- `O 4` Print 4 bytes in octal.
- `q 2` Print in signed octal.
- `Q 4` Print long signed octal.
- `d 2` Print in decimal.
- `D 4` Print long decimal.
x 2  Print 2 bytes in hexadecimal.
X 4  Print 4 bytes in hexadecimal.
u 2  Print as an unsigned decimal number.
U 4  Print long unsigned decimal.
f 4  Print the 32 bit value as a floating point number.
F 8  Print double floating point.
b 1  Print the addressed byte in octal.
c 1  Print the addressed character.
C 1  Print the addressed character using the following escape
     convention. Character values 000 to 040 are printed as @
     followed by the corresponding character in the range 0100
     to 0140. The character @ is printed as @@.
s n  Print the addressed characters until a zero character is
     reached.
S n  Print a string using the @ escape convention. n is the
     length of the string including its zero terminator.
Y 4  Print 4 bytes in date format (see ctime(3C)).
i n  Print as PDP-11 instructions. n is the number of bytes
     occupied by the instruction. This style of printing causes
     variables I and 2 to be set to the offset parts of the source
     and destination respectively.
a 0  Print the value of dot in symbolic form. Symbols are
     checked to ensure that they have an appropriate type as
     indicated below.

/   local or global data symbol
?   local or global text symbol
=   local or global absolute symbol

p 2  Print the addressed value in symbolic form using the same
     rules for symbol lookup as a.
t 0  When preceded by an integer tabs to the next appropriate
     tab stop. For example, 8t moves to the next 8-space tab
     stop.
r 0  Print a space.
n 0  Print a new-line.
"..." 0 Print the enclosed string.

  Dot is decremented by the current increment. Nothing is
  printed.
  +  Dot is incremented by 1. Nothing is printed.
  -  Dot is decremented by 1. Nothing is printed.

new-line
Repeat the previous command with a count of 1.

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

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

[?/m] bl el fl[/?]  
New values for (bl, el, fl) are recorded. If less than three expres-
sions are given then the remaining map parameters are left
unchanged. If the ? or / is followed by * then the second segment \((b_2, e_2, f_2)\) of the mapping is changed. If the list is terminated by ? or / then the file (objfil or corfil respectively) is used for subsequent requests. (So that, for example, /m? will cause / to refer to objfil.)

>name

Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following !.

$modifier

Miscellaneous commands. The available modifiers are:

- `<f` Read commands from the file \(f\) and return.
- `>f` Send output to the file \(f\), which is created if it does not exist.
- `r` Print the general registers and the instruction addressed by \(pc\). Dot is set to \(pc\).
- `f` Print the floating registers in single or double length. If the floating point status of \(ps\) is set to double (0200 bit) then double length is used anyway.
- `b` Print all breakpoints and their associated counts and commands.
- `a` ALGOL 68 stack backtrace. If \(address\) is given then it is taken to be the address of the current frame (instead of \(r4\)). If \(count\) is given then only the first \(count\) frames are printed.
- `c` C stack backtrace. If \(address\) is given then it is taken as the address of the current frame (instead of \(r5\)). If \(C\) is used then the names and (16 bit) values of all automatic and static variables are printed for each active function. If \(count\) is given then only the first \(count\) frames are printed.
- `e` The names and values of external variables are printed.
- `w` Set the page width for output to \(address\) (default 80).
- `s` Set the limit for symbol matches to \(address\) (default 255).
- `o` All integers input are regarded as octal.
- `d` Reset integer input as described in \(EXPRESSIONS\).
- `q` Exit from \(adb\).
- `v` Print all non zero variables in octal.
- `m` Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

- `bc` Set breakpoint at \(address\). The breakpoint is executed \(count - 1\) times before causing a stop. Each time the breakpoint is encountered the command \(c\) is executed. If this command sets \(dot\) to zero then the breakpoint causes a stop.
- `d` Delete breakpoint at \(address\).
- `r` Run objfil as a subprocess. If \(address\) is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. \(count\) specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on
entry to the subprocess.

es The subprocess is continued with signal \textit{s} (see \textit{signal}(2)). If \textit{address} is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for \textit{r}.

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

\textbf{k} The current subprocess, if any, is terminated.

\textbf{VARIABLES}

\textit{Adb} provides a number of variables. Named variables are set initially by \textit{adb} but are not used subsequently. Numbered variables are reserved for communication as follows.

- \textit{0} The last value printed.
- \textit{1} The last offset part of an instruction source.
- \textit{2} The previous value of variable \textit{1}.

On entry the following are set from the system header in the \textit{corfil}. If \textit{corfil} does not appear to be a \textit{core} file then these values are set from \textit{objfil}.

- \textit{b} The base address of the data segment.
- \textit{d} The data segment size.
- \textit{e} The entry point.
- \textit{m} The "magic" number (0405, 0407, 0410 or 0411).
- \textit{s} The stack segment size.
- \textit{t} The text segment size.

\textbf{ADDRESSES}

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples \((b_1, e_1, f_1)\) and \((b_2, e_2, f_2)\) and the \textit{file address} corresponding to a written \textit{address} is calculated as follows:

\[
\text{if } b_1 \leq \text{address} < e_1 \Rightarrow \text{file address} = \text{address} + f_1 - b_1
\]

otherwise

\[
\text{if } b_2 \leq \text{address} < e_2 \Rightarrow \text{file address} = \text{address} + f_2 - b_2.
\]

otherwise, the requested \textit{address} is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a \texttt{?} or \texttt{/} is followed by an \texttt{*} then only the second triple is used.

The initial setting of both mappings is suitable for normal \textit{a.out} and \textit{core} files. If either file is not of the kind expected then, for that file, \textit{b1} is set to 0, \textit{e1} is set to the maximum file size and \textit{f1} is set to 0; in this way the whole file can be examined with no address translation.

In order for \textit{adb} to be used on large files all appropriate values are kept as signed 32 bit integers.

\textbf{FILES}

- \texttt{/dev/mem}
- \texttt{/dev/swap}
- \texttt{a.out}
- \texttt{core}
SEE ALSO
ptrace(2), a.out(5), core(5).

DIAGNOSTICS
"Adb" when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc.
Exit status is 0, unless last command failed or returned nonzero status.

BUGS
A breakpoint set at the entry point is not effective on initial entry to the program.
When single stepping, system calls do not count as an executed instruction.
Local variables whose names are the same as an external variable may foul up the accessing of the external.
NAME
admin — create and administer SCCS files

SYNOPSIS
admin [-n] [-i(name)] [-rrel] [-t(name)] [-f(flag[flag-val])]
[-d(flag[flag-val])] [-a(login)] [-e(login)] [-m(mrlist)] [-y(comment)]
[-h] [-z] files

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

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

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

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

-i(name)
The name of a file from which the text for a new
SCCS file is to be taken. The text constitutes the first
delta of the file (see -r keyletter for delta numbering
scheme). If the i keyletter is used, but the file name
is omitted, the text is obtained by reading the stan-
dard input until an end-of-file is encountered. If this
keyletter is omitted, then the SCCS file is created
empty. Only one SCCS file may be created by an
admin command on which the i keyletter is supplied.

Using a single admin to create two or more SCCS files
require that they be created empty (no -i keyletter).
Note that the -i keyletter implies the -n keyletter.

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

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

This keyletter specifies a flag, and, possibly, a value for the flag, to be placed in the SCSS file. Several flags may be supplied on a single admin command line. The allowable flags and their values are:

b Allows use of the -b keyletter on a get(1) command to create branch deltas.
cceil The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a get(1) command for editing. The default value for an unspecified c flag is 9999.
ffloor The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a get(1) command for editing. The default value for an unspecified f flag is 1.
dSID The default delta number (SID) to be used by a get(1) command.
i Causes the "No id keywords (ge6)" message issued by get(1) or delta(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCSS identification keywords (see get(1)) are found in the text retrieved or stored in the SCSS file.
j Allows concurrent get(1) commands for editing on the same SID of an SCSS file. This allows multiple concurrent updates to the same version of the SCSS file.
list A list of releases to which deltas can no longer be made (get -e against one of these "locked" releases fails). The list has the following syntax:

\(<list> ::= <range> | <list> , <range> \\
<range> ::= RELEASE NUMBER | a

The character a in the list is equivalent to specifying all releases for the named SCSS file.

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

qtext User definable text substituted for all occurrences of the %Q% keyword in SCSS file text retrieved by get(1).
mmod Module name of the SCSS file substituted for all occurrences of the %M% keyword in SCSS file text retrieved by get(1). If the m flag is not specified, the value assigned is the name of the SCSS file with the
leading s. removed.

type

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

v[pgm]

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

-dflag

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

list

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

-alogin

A login name, or numerical UNIX group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all login names common to that group ID. Several a keyletters may be used on a single admin command line. As many logins, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.

-e-login

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

-y[comment]

The comment text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of delta(1). Omission of the -y keyletter results in a default comment line being inserted in the form: date and time created YY/MM/DD HH:MM:SS by login The -y keyletter is valid only if the -i and/or -m keyletters are specified (i.e., a new SCCS file is being created).

-m[mrlist]

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

-h

Causes admin to check the structure of the SCCS file (see sccsfile(5)), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum
that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

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

```
-z
```

The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see \textit{--h}, above).

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

\textbf{FILES}

The last component of all SCCS file names must be of the form \texttt{x.file-name}. New SCCS files are given mode 444 (see \textit{chmod(1)}). Write permission in the pertinent directory is, of course, required to create a file. All writing done by \texttt{admin} is to a temporary \texttt{x-file}, called \texttt{x.file-name}, (see \textit{get(1)}), created with mode 444 if the \texttt{admin} command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of \texttt{admin}, the SCCS file is removed (if it exists), and the \texttt{x-file} is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

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

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

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

\textbf{SEE ALSO}

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


\textbf{DIAGNOSTICS}

Use \textit{help(1)} for explanations.
NAME
ar — archive and library maintainer

SYNOPSIS
ar key [ posname ] afile name ...

DESCRIPTION
Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose.

Ar can read archive files produced in either PDP-11 or VAX-11/780 format (see ar(5)). However, when ar creates an archive, it always creates the header in the format of the local system. A conversion program exists to convert PDP-11 archives to VAX-11/780 archive format (see arcv(1)). This feature is useful only for source archive files. Individual files are inserted without conversion into the archive file.

Key is one character from the set drqtpmx, optionally concatenated with one or more of vuaibcl. Afile is the archive file. The names are constituent files in the archive file. The meanings of the key characters are:

d  Delete the named files from the archive file.

r  Replace the named files in the archive file. If the optional character u is used with r, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set abi is used, then the posname argument must be present and specifies that new files are to be placed after (a) or before (b or i) posname. Otherwise new files are placed at the end.

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

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

p  Print the named files in the archive.

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

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

v  Verbose. Under the verbose option, ar gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with t, it gives a long listing of all information about the files. When used with x, it precedes each file with a name.

c  Create. Normally ar will create afile when it needs to. The create option suppresses the normal message that is produced when afile is created.

l  Local. Normally ar places its temporary files in the directory /tmp. This option causes them to be placed in the local directory.

FILES
/tmp/v*  temporaries
SEE ALSO
   arcv(1), ld(1), lorder(1), ar(5).

BUGS
   If the same file is mentioned twice in an argument list, it may be put in the archive twice.
NAME
  arcv — convert archive files from PDP-11 to VAX-11/780 format

SYNOPSIS
  arcv files

DESCRIPTION
  Arcv converts source archive files from the PDP-11 format to the VAX-
  11/780 format. Because each converted file is copied over the original file,
  arcv runs with all interrupts turned off.

FILES
  /tmp/arC*

SEE ALSO
  ar(1), ar(5).
NAME
as — assembler for PDP-II

SYNOPSIS
as [ - ] [ -o objfile ] file ...

DESCRIPTION
As assembles the concatenation of the named files. If the optional first argument is used, all undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file objfile; if that is omitted, a.out is used. It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

FILES
/lib/as2 pass 2 of the assembler
/tmp/atm[1-3]? temporary
a.out object

SEE ALSO
adb(1), ld(1), nm(1), a.out(5).
Unix Assembler Manual by D. M. Ritchie

DIAGNOSTICS
If the name chosen for the output file is of the form *?[es], the assembler issues an appropriate complaint and quits. When an input file cannot be read, its name followed by a question mark is typed and assembly ceases. When syntactic or semantic errors occur, a single-character diagnostic is typed out together with the line number and the file name in which it occurred. Errors in pass 1 cause cancellation of pass 2. The possible errors are:

) Parentheses error
] Parentheses error
< String not terminated properly
* Indirection used illegally
. Illegal assignment to .
# Error in address
b Branch instruction is odd or too remote
e Error in expression
f Error in local (f or b) type symbol
g Garbage (unknown) character
i End of file inside an .if
m Multiply-defined symbol as label
o Word quantity assembled at odd address
p . different in pass 1 and 2
r Relocation error
u Undefined symbol
x Syntax error

BUGS
Syntax errors can cause incorrect line numbers in subsequent diagnostics.
NAME
as — assembler for VAX-11/780

SYNOPSIS
as [ -d124 ] [ -o objfile ] [ name ]

DESCRIPTION
As assembles the named file, or the standard input if no file name is specified. The optional argument -d may be used to specify the number of bytes to be assembled for offsets which involve forward or external references, which have sizes unspecified in the assembly language. The default is four bytes, i.e., -d4. All undefined symbols in the assembly are treated as global.

The output of the assembly is left on the file objfile; if that is omitted, a.out is used. It is executable if no errors occurred during the assembly, and if there were no unresolved external references.

FILES
/tmp/as* temporary
/tmp/a[ab][a−h]t* temporary
a.out object

SEE ALSO
adb(1), ld(1), nm(1), sdb(1), a.out(5).
NAME
awk — pattern scanning and processing language

SYNOPSIS
awk [ -Fc ] [ prog ] [ files ]

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

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

An input line is made up of fields separated by white space. (This default can be changed by using FS, see below). The fields are denoted $1, $2, ...; $0 refers to the entire line.

A pattern-action statement has the form:

pattern { action }

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; 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 quoted (").

The print statement prints its arguments on the standard output (or on a file if >expr is present), separated by the current output field separator, and terminated by the output record separator. The printf statement formats its expression list according to the format (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
```

where a relop is any of the six relational operators in C, and a matchop is either `~` (for `contains`) or `!` (for `does not contain`). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character `c` may be used to separate the fields by starting the program with:

```
BEGIN { FS = c }
```

or by using the `-Fc` option.

Other variable names with special meanings include `NF`, the number of fields in the current record; `NR`, the ordinal number of the current record; `FILENAME`, the name of the current input file; `OFS`, the output field separator (default blank); `ORS`, the output record separator (default new-line); and `OFMT`, the output format for numbers (default `%.6g`).

**EXAMPLES**

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
{ s += $1 }
END { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

**SEE ALSO**

grep(1), lex(1), sed(1).

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
  banner — make posters

SYNOPSIS
  banner strings

DESCRIPTION
  *Banner* prints its arguments (each up to 10 characters long) in large letters
  on the standard output.
NAME
basename, dirname — deliver portions of path names

SYNOPSIS
basename string [ suffix ]
dirname string

DESCRIPTION
Base name 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.

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

EXAMPLES
The following example, invoked with the argument /usr/src/cmd/cat.c, compiles the named file and moves the output to a file named cat in the current directory:

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

The following example will set the shell variable NAME to /usr/src/cmd:

NAME=`dirname /usr/src/cmd/cat.c`

SEE ALSO
sh(1).
NAME
bc — arbitrary-precision arithmetic language

SYNOPSIS
bc [ -e ] [ -l ] [ file ... ]

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

Comments
are enclosed in /* and */.

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

Other operands
arbitrarily long numbers with optional sign and decimal point.
( E )
sqrt ( E )
length ( E ) number of significant decimal digits
scale ( E ) number of digits right of decimal point
L ( E , ... , E )

Operators

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

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

Function definitions
define L ( L , ... , L ) {
    auto L , ... , L
    S; ... S
    return ( E )
}

Functions in -l math library
s(x) sine
C(x) cosine
e(x) exponential
l(x) log
a(x) arctangent
j(n,x) Bessel function

All function arguments are passed by value.
The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to `scale` influences the number of digits to be retained on arithmetic operations in the manner of `dc(1)`. Assignments to `ibase` or `obase` set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables empty square brackets must follow the array name.

`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.

**EXAMPLE**

```plaintext
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1; 1==1; i++){
        a = a*x;
        b = b*i;
        c = a/b;
        if(c == 0) return(s);
        s = s+c;
    }
}
```

defines a function to compute an approximate value of the exponential function and

```plaintext
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

**FILES**

- `/usr/lib/lib.b`  mathematical library
- `/usr/bin/dc`  desk calculator proper

**SEE ALSO**

`dc(1)`.

**BC — An Arbitrary Precision Desk-Calculator Language**

by L. L. Cherry and R. Morris.

**BUGS**

No `&`, `||` yet.

*For* statement must have all three E's.

*Quit* is interpreted when read, not when executed.
NAME
bcopy — interactive block copy

SYNOPSIS
/etc/bcopy

DESCRIPTION
Bcopy dates from a time when neither the UNIX file system nor the DEC
disk drives were as reliable as they are now. Bcopy copies from and to files
starting at arbitrary block (512-byte) boundaries.

The following questions are asked:

to: (you name the file or device to be copied to).
offset: (you provide the starting "to" block number).
from: (you name the file or device to be copied from).
offset: (you provide the starting "from" block number).
count: (you reply with the number of blocks to be copied).

After count is exhausted, the from question is repeated (giving you a
chance to concatenate blocks at the to+offset+count location). If you
answer from with a carriage return, everything starts over.

Two consecutive carriage returns terminate bcopy.

SEE ALSO
cpio(1), dd(1).
NAME
bdiff — big diff

SYNOPSIS
bdiff file1 file2 [n] [−s]

DESCRIPTION
Bdiff is used in a manner analogous to diff(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for diff. Bdiff ignores lines common to the beginning of both files, splits the remainder of each file into n-line segments, and invokes diff upon corresponding segments. The value of n is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for n. This is useful in those cases in which 3500-line segments are too large for diff, causing it to fail. If file1 (file2) is −, the standard input is read. The optional −s (silent) argument specifies that no diagnostics are to be printed by bdiff (note, however, that this does not suppress possible exclamations by diff). If both optional arguments are specified, they must appear in the order indicated above.

The output of bdiff is exactly that of diff, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, bdiff does not necessarily find a smallest sufficient set of file differences.

FILES
/tmp/bd?????

SEE ALSO
diff(1).

DIAGNOSTICS
Use help(1) for explanations.
NAME
bfs — big file scanner

SYNOPSIS
bfs [ - ] name

DESCRIPTION
Bfs is (almost) like ed(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 255 characters per line. Bfs is usually more efficient than ed for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where csplit(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the w command. The optional - suppresses printing of sizes. Input is prompted with * if P and a carriage return are typed as in ed. Prompting can be turned off again by inputting another P and carriage return. Note that messages are given in response to errors if prompting is turned on.

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. Since bfs uses a different regular expression-matching routine from ed, the regular expressions accepted are slightly wider in scope (see regex(3X)). There is a slight difference in mark names: only the letters a through z may be used, and all 26 marks are remembered.

The e, g, v, k, n, p, q, w, =, ! and null commands operate as described under ed. Commands such as -- --, ++++, +++=, -12, and +4p are accepted. Note that 1,10p and 1,10 will both print the first ten lines. The f command only prints the name of the file being scanned; there is no remembered file name. The w command is independent of output diversion, truncation, or crunching (see the xo, xt and xc commands, below). The following additional commands are available:

xf file
Further commands are taken from the named file. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the xf. Xf commands may be nested to a depth of 10.

xo [file]
Further output from the p and null commands is diverted to the named file, which, if necessary, is created mode 666. If file is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: label
This positions a label in a command file. The label is terminated by new-line, and blanks between the : and the start of the label are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

( , , )xb/regular expression/label
A jump (either upward or downward) is made to label if the command succeeds. It fails under any of the following conditions:
1. Either address is not between 1 and $.
2. The second address is less than the first.
3. The regular expression doesn't match at least one line in the specified range, including the first and last lines.

On success, . is set to the line matched and a jump is made to label. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

```
xb/' label
```

is an unconditional jump.

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

```
xv
```

Output from the p and null commands is truncated to at most number characters. The initial number is 255.

```
xv [digit] [spaces] [value]
```

The variable name is the specified digit following the xv. xv5100 or xv5 100 both assign the value 100 to the variable 5. xv61,100p assigns the value 1,100p to the variable 6. To reference a variable, put a % in front of the variable name. For example, using the above assignments for variables 5 and 6:

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

will all print the first 100 lines.

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

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

```
g/.*%\[cdsl]/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 UNIX command can be stored into a variable. The only requirement is that the first character of value be an !. For example:

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

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

```
xv7!date
```

- 2 -
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 UNIX command (!command) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string size.

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

**xc [switch]**

If switch is 1, output from the p and null commands is crunched; if switch is 0 it isn’t. Without an argument, xc reverses switch. Initially switch is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

SEE ALSO

csplit(1), ed(1), regex(3X).

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.
NAME
bs — a compiler/interpreter for modest-sized programs

SYNOPSIS
bs [ file [ args ] ]

DESCRIPTION
Bs is a remote descendant of Basic and Snobol4 with a little C language
thrown in. Bs is designed for programming tasks where program develop­
ment time is as important as the resulting speed of execution. Formalities
of data declaration and file/process manipulation are minimized. Line-at-
a-time debugging, the trace and dump statements, and useful run-time error
messages all simplify program testing. Furthermore, incomplete programs
can be debugged; inner functions can be tested before outer functions have
been written and vice versa.

If the command line file argument is provided, the file is used for input
before the console is read. By default, statements read from the file
argument are compiled for later execution. Likewise, statements entered
from the console are normally executed immediately (see compile and exe­
cute below). Unless the final operation is assignment, the result of an
immediate expression statement is printed.

Bs programs are made up of input lines. If the last character on a line is a
\, the line is continued. Bs accepts lines of the following form:

statement
label statement

A label is a name (see below) followed by a colon. A label and a variable
can have the same name.

A bs statement is either an expression or a keyword followed by zero or
more expressions. Some keywords (clear, compile, !, execute, include, ibase, obase, and run) are always executed as they are compiled.

Statement Syntax:

expression

The expression is executed for its side effects (value, assignment or
function call). The details of expressions follow the description of sta­
tement types below.

break
Break exits from the inner-most for/while loop.

clear
Clears the symbol table and compiled statements. Clear is executed
immediately.

compile [ expression ]
Succeeding statements are compiled (overrides the immediate execution
default). The optional expression is evaluated and used as a file name
for further input. A clear is associated with this latter case. Compile is
executed immediately.

continue
Continue transfers to the loop-continuation of the current for/while loop.

dump
The name and current value of every non-local variable is printed.
After an error or interrupt, the number of the last statement and (possi­
bly) the user-function trace are displayed.
exit [ expression ]
Return to system level. The expression is returned as process status.

execute
Change to immediate execution mode (an interrupt has a similar effect).
This statement does not cause stored statements to execute (see run below).

for name = expression expression statement
for name = expression expression
....
next
for expression , expression , expression statement
for expression , expression , expression
....
next
The for statement repetitively executes a statement (first form) or a
group of statements (second form) under control of a named variable.
The variable takes on the value of the first expression, then is
incremented by one on each loop, not to exceed the value of the second
expression. The third and fourth forms require three expressions
separated by commas. The first of these is the initialization, the second
is the test (true to continue), and the third is the loop-continuation
action (normally an increment).

fun f([a, ...]) [v, ...]

nuf
Fun defines the function name, arguments, and local variables for a
user-written function. Up to ten arguments and local variables are
allowed. Such names cannot be arrays, nor can they be I/O associated.
Function definitions may not be nested.

freturn
A way to signal the failure of a user-written function. See the interroga-
tion operator (?) below. If interrogation is not present, freturn merely
returns zero. When interrogation is active, freturn transfers to that
expression (possibly by-passing intermediate function returns).

ibase N
Ibase sets the input base (radix) to N. The only supported values for N
are 8, 10 (the default), and 16. Hexadecimal values 10—15 are entered
as a—f. A leading digit is required (i.e., f0a must be entered as 0f0a).
Ibase (and obase, below) are executed immediately.

goto name
Control is passed to the internally stored statement with the matching
label.

if expression statement
if expression
....
[ else
.... ]
fi
The statement (first form) or group of statements (second form) is exe-
cuted if the expression evaluates to non-zero. The strings 0 and "
(null) evaluate as zero. In the second form, an optional else allows for
a group of statements to be executed when the first group is not. The
only statement permitted on the same line with an else is an if; only
other fi's can be on the same line with a fi. The elision of else and if into an endif is supported. Only a single fi is required to close an if . . . elif . . . [ else . . . ] sequence.

include expression
The expression must evaluate to a file name. The file must contain bs Such statements become part of the program being compiled. source statements. Include statements may not be nested.

obase $N$
Obase sets the input base to $N$ (see ibase above).

onintr label
onintr
The onintr command provides program control of interrupts. In the first form, control will pass to the label given, just as if a goto had been executed at the time onintr was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause bs to terminate.

return [expression]
The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run
The random number generator is reset. Control is passed to the first internal statement. If the run statement is contained in a file, it should be the last statement.

stop
Execution of internal statements is stopped. Bs reverts to immediate mode.

trace [ expression ]
The trace statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each return decrements the trace expression value.

while expression statement
while expression

next
While is similar to for except that only the conditional expression for loop-continuation is given.

! shell command
An immediate escape to the Shell.

# . . .
This statement is ignored. It is used to interject commentary in a program.

Expression Syntax:
name
A name is used to specify a variable. Names are composed of a letter (upper or lower case) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in fun statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function open() below).
name ( [expression [, expression] ... ] )

Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a fun statement. Arguments to functions are passed by value.

name [ expression [, expression ] ... ]

This syntax is used reference either arrays or tables (see built-in table functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; a[1,2] is the same as a[1][2]. The truncated expressions are restricted to values between 0 and 32767.

number

A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an e followed by a possibly signed exponent.

string

Character strings are delimited by * characters. The \ escape character allows the double quote ("), new-line (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. Otherwise, \ stands for itself.

( expression )

Parentheses are used to alter the normal order of evaluation.

( expression, expression [, expression ... ] ) [ expression ]

The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

( False, True )[ a == b ]

has the value True if the comparison is true.

? expression

The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-file (see examples in the Programming Tips section below), the result of the eval built-in function, and for checking the return from user-written functions (see freturn). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

~ expression

The result is the negation of the expression.

++ name

Increments the value of the variable (or array reference). The result is the new value.

-- name

Decrements the value of the variable. The result is the new value.

! expression

The logical negation of the expression. Watch out for the shell escape command.

expression operator expression

Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands
are converted to numeric form before the function is applied.

**Binary Operators** (in increasing precedence):

- 
  - is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

- _ (underscore) is the concatenation operator.

& | & (logical and) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.

< <= > >= == !=

The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: a>b>c is the same as a>b & b>c. A string comparison is made if both operands are strings.

+ -
  - Add and subtract.

* / %
  - Multiply, divide, and remainder.

- Exponentiation.

**Built-in Functions:**

*Dealing with arguments*

**arg(i)**

is the value of the i-th actual parameter on the current level of function call. At level zero, arg returns the i-th command-line argument (arg(0) returns bs).

**narg()**

returns the number of arguments passed. At level zero, the command argument count is returned.

*Mathematical*

**abs(x)**

is the absolute value of x.

**atan(x)**

is the arctangent of x. Its value is between $-\pi/2$ and $\pi/2$.

**ceil(x)**

returns the smallest integer not less than x.

**cos(x)**

is the cosine of x (radians).

**exp(x)**

is the exponential function of x.

**floor(x)**

returns the largest integer not greater than x.
\[ \log(x) \]
is the natural logarithm of \( x \).

\[ \text{rand}() \]
is a uniformly distributed random number between zero and one.

\[ \sin(x) \]
is the sine of \( x \) (radians).

\[ \sqrt{x} \]
is the square root of \( x \).

**String operations**

\[ \text{size}(s) \]
the size (length in bytes) of \( s \) is returned.

\[ \text{format}(f, a) \]
returns the formatted value of \( a \). \( F \) is assumed to be a format specification in the style of `printf(3S)`. Only the \%f, \%e, and \%s types are safe.

\[ \text{index}(x, y) \]
returns the number of the first position in \( x \) that any of the characters from \( y \) matches. No match yields zero.

\[ \text{trans}(s, f, t) \]
Translates characters of the source \( s \) from matching characters in \( f \) to a character in the same position in \( t \). Source characters that do not appear in \( f \) are copied to the result. If the string \( f \) is longer than \( t \), source characters that match in the excess portion of \( f \) do not appear in the result.

\[ \text{substr}(s, \text{start}, \text{width}) \]
returns the sub-string of \( s \) defined by the \text{start}ing position and \text{width}.

\[ \text{match}(\text{string}, \text{pattern}) \]
\[ \text{mstring}(n) \]
The \text{pattern} is similar to the regular expression syntax of the `ed(1)` command. The characters ., [ ], ~ (inside brackets), * and $ are special. The \text{mstring} function returns the \text{n}-th \((1 <= n <= 10)\) sub-string of the subject that occurred between pairs of the pattern symbols \( \backslash ( \) and \( \backslash ) \) for the most recent call to \text{match}. To succeed, patterns must match the beginning of the string (as if all patterns began with '*'). The function returns the number of characters matched. For example:

\[ \text{match}("a123ab123", ".*([a-z]\}) = = 6 \]
\[ \text{mstring}(1) = = "b" \]

**File handling**

\[ \text{open}(\text{name}, \text{file}, \text{function}) \]
\[ \text{close}(\text{name}) \]
The \text{name} argument must be a \text{bs} variable name (passed as a string). For the \text{open}, the \text{file} argument may be 1) a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively, 2) a string representing a file name, or 3) a string beginning with an `!` representing a command to be executed (via `sh -c`). The \text{function} argument must be either \text{r} (read), \text{w} (write), \text{W} (write without new-line), or \text{a} (append). After a \text{close}, the \text{name} reverts to being an ordinary variable. The initial associations are:

\[ \text{open}("get", 0, "r") \]
\[ \text{open}("put", 1, "w") \]
\[ \text{open}("puterr", 2, "w") \]
Examples are given in the following section.

**access(s, m)**
executes \texttt{access(2)}.

**ftype(s)**
returns a single character file type indication: \texttt{f} for regular file, \texttt{d} for directory, \texttt{b} for block special, or \texttt{c} for character special.

### Tables

**table(name, size)**
A table in \texttt{bs} is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The \texttt{name} argument must be a \texttt{bs} variable name (passed as a string). The \texttt{size} argument sets the minimum number of elements to be allocated. \texttt{Bs} prints an error message and stops on table overflow.

**item(name, i)**

**key()**
The \texttt{item} function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the \texttt{item} function accesses values, the \texttt{key} function accesses the "subscript" of the previous \texttt{item} call. The \texttt{name} argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table, for example:

```plaintext
table("t", 100)
... # If \texttt{word} contains "party", the following expression adds one to the count
# of that word:
++t[word]
... # To print out the the key/value pairs:
for i = 0, (?s = item(t, i)), ++i if key() put = key()"::*":s
```

**iskey(name, word)**
The \texttt{iskey} function tests whether the key \texttt{word} exists in the table \texttt{name} and returns one for true, zero for false.

### Odds and ends

**eval(s)**
The string argument is evaluated as a \texttt{bs} expression. The function is handy for converting numeric strings to numeric internal form. \texttt{Eval} can also be used as a crude form of indirection, as in:

```plaintext
name = "xyz"
Eval("+"_name)
```

which increments the variable \texttt{xyz}. In addition, \texttt{eval} preceded by the interrogation operator permits the user to control \texttt{bs} error conditions. For example:

```plaintext
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a \texttt{goto} to the label \texttt{L} (if it exists):

```plaintext
label="L"
if !(?eval("goto "_label)) puterr = "no label"
```
plot(request, args)

The plot function produces output on devices recognized by tplot(1G).
The requests are as follows:

<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot(0, term)</td>
<td>causes further plot output to be</td>
</tr>
<tr>
<td></td>
<td>piped into tplot(1G) with an</td>
</tr>
<tr>
<td></td>
<td>argument of -Tterm.</td>
</tr>
<tr>
<td>plot(1)</td>
<td>&quot;erases&quot; the plotter.</td>
</tr>
<tr>
<td>plot(2, string)</td>
<td>labels the current point with string.</td>
</tr>
<tr>
<td>plot(3, x1, y1, x2, y2)</td>
<td>draws the line between (x1,y1) and</td>
</tr>
<tr>
<td></td>
<td>(x2,y2).</td>
</tr>
<tr>
<td>plot(4, x, y, r)</td>
<td>draws a circle with center (x,y) and</td>
</tr>
<tr>
<td></td>
<td>radius r.</td>
</tr>
<tr>
<td>plot(5, x1, y1, x2, y2, x3, y3)</td>
<td>draws an arc (counterclockwise) with</td>
</tr>
<tr>
<td></td>
<td>center (x1,y1) and endpoints (x2,y2)</td>
</tr>
<tr>
<td></td>
<td>and (x3,y3).</td>
</tr>
<tr>
<td>plot(6)</td>
<td>is not implemented.</td>
</tr>
<tr>
<td>plot(7, x, y)</td>
<td>makes the current point (x,y).</td>
</tr>
<tr>
<td>plot(8, x, y)</td>
<td>draws a line from the current point</td>
</tr>
<tr>
<td></td>
<td>to (x,y).</td>
</tr>
<tr>
<td>plot(9, x, y)</td>
<td>draws a point at (x,y).</td>
</tr>
<tr>
<td>plot(10, string)</td>
<td>sets the line mode to string.</td>
</tr>
<tr>
<td>plot(11, x1, y1, x2, y2)</td>
<td>makes (x1,y1) the lower left corner</td>
</tr>
<tr>
<td></td>
<td>of the plotting area and (x2,y2) the</td>
</tr>
<tr>
<td></td>
<td>upper right corner of the plotting</td>
</tr>
<tr>
<td></td>
<td>area.</td>
</tr>
<tr>
<td>plot(12, x1, y1, x2, y2)</td>
<td>causes subsequent x (y) coordinates</td>
</tr>
<tr>
<td></td>
<td>to be multiplied by x1 (y1) and then</td>
</tr>
<tr>
<td></td>
<td>added to x2 (y2) before they are</td>
</tr>
<tr>
<td></td>
<td>plotted. The initial scaling is</td>
</tr>
<tr>
<td></td>
<td>plot(12, 1.0, 1.0, 0.0, 0.0).</td>
</tr>
</tbody>
</table>

Some requests do not apply to all plotters. All requests except zero and
twelve are implemented by piping characters to tplot(1G). See plot(5)
for more details.

last( )
in immediate mode, last returns the most recently computed value.

PROGRAMMING TIPS
Using bs as a calculator:

```
$ bs
#     Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

#     Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5=4 bal = bal + bal*int
bal = 1000
```
The outline of a typical bs program:

```bash
# initialize things:
var1 = 1
open("read", "infile", "r")
...
# compute:
while ?(str = read)
  ...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```bash
# Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

# Pipe between commands.
open("ls", "ls *", "r")
open("pr", "!pr --2 -h 'List'", "w")
while ?(pr = ls) ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

SEE ALSO
ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), Section 3 of this volume for further description of the mathematical functions (pow(3M) is used for exponentiation), plot(5). Bs uses the Standard Input/Output package.
NAME
cal — print calendar

SYNOPSIS
cal [ month ] year

DESCRIPTION
Cal prints a calendar for the specified year. If a month is also specified, a
calendar just for that month is printed. Year can be between 1 and 9999.
The month is a number between 1 and 12. The calendar produced is that
for England and her colonies.
Try September 1752.

BUGS
The year is always considered to start in January even though this is historically naive.
Beware that "cal 78" refers to the early Christian era, not the 20th century.
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 “Dec. 7,” “December 7,” “12/7,” etc., are recognized, but not “7 December” or “7/12”. On weekends “tomorrow” extends through Monday.

When an argument is present, calendar does its job for every user who has a file calendar in his login directory and sends him any positive results by mail(1). Normally this is done daily in the wee hours under control of cron(1M).

FILES
calendar
/usr/lib/calprog to figure out today’s and tomorrow’s dates
/etc/passwd
/tmp/cal
/usr/lib/crontab

SEE ALSO
cron(1M), mail(1).

BUGS
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 print files

SYNOPSIS
cat [ -u ] [ -s ] file ...

DESCRIPTION
Cat reads each file in sequence and writes it on the standard output. Thus:
cat file
prints the file, and:
cat file1 file2 >file3
concatenates the first two files and places the result on the third.
If no input file is given, or if the argument - is encountered, cat reads
from the standard input file. Output is buffered in 512-byte blocks unless
the -u option is specified. The -s option makes cat silent about non-
existent files. No input file may be the same as the output file unless it is a
special file.

SEE ALSO
cp(1), pr(1).
NAME
   cb — C program beautifier

SYNOPSIS
   cb [file]

DESCRIPTION
   Cb places a copy of the C program in file (standard input if file is not
given) on the standard output with spacing and indentation that displays
the structure of the program.
NAME
cc, pcc — C compiler

SYNOPSIS
cc [ option ] ... file ...
pcc [ option ] ... file ...

DESCRIPTION
Cc is the UNIX C compiler. Pcc is the portable version for a PDP-11 machine. They accept several types of arguments:

Arguments whose names end with .c are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with .o substituted for .c. The .o file is normally deleted, however, if a single C program is compiled and loaded all at once.

In the same way, arguments whose names end with .s are taken to be assembly source programs and are assembled, producing a .o file.

The following options are interpreted by cc and pcc. See ld(1) for link editor options.

-c Suppress the link edit phase of the compilation, and force an object file to be produced even if only one program is compiled.
-p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one which automatically calls monitor(3C) at the start and arranges to write out a mon.out file at normal termination of execution of the object program. An execution profile can then be generated by use of prof(1).
-f Link the object program with the floating-point interpreter for systems without hardware floating-point.
-g Cause the compiler to generate additional information needed for the use of sdb(1). (VAX-11/780 only.)
-dn This option is passed through to as(1). (VAX only.)
-O Invoke an object-code optimizer.
-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed .s.
-E Run only the macro preprocessor on the named C programs, and send the result to the standard output.
-P Run only the macro preprocessor on the named C programs, and leave the result on corresponding files suffixed .i.
-C Comments are not stripped by the macro preprocessor.
-Dname=def
-Dname
Define the name to the preprocessor, as if by #define. If no definition is given, the name is defined as 1.
-Uname
Remove any initial definition of name.
-Idir Change the algorithm for searching for #include files whose names do not begin with / to look in dir before looking in the directories on the standard list. Thus, #include files whose names are enclosed in " will be searched for first in the directory of the file argument, then in directories named in -I options, and last in
directories on a standard list. For \texttt{#include} files whose names are enclosed in <>, the directory of the \texttt{file} argument is not searched.

\-\texttt{B}<\texttt{string}>

Find substitute compiler passes in the files named \texttt{string} with the suffixes \texttt{cpp}, \texttt{c0}, \texttt{c1} and \texttt{c2}. If \texttt{string} is empty, use a standard backup version.

\-t[\texttt{p012}]

Find only the designated compiler passes in the files whose names are constructed by a \-B option. In the absence of a \-B option, the \texttt{string} is taken to be /lib/m.

Other arguments are taken to be either link editor option arguments, or C-compatible object programs, typically produced by an earlier \textit{cc} or \textit{pcc} run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the name \texttt{a.out}.

FILES

\begin{align*}
\texttt{file.c} & \quad \text{input file} \\
\texttt{file.o} & \quad \text{object file} \\
\texttt{a.out} & \quad \text{linked output} \\
\texttt{/tmp/ctm*} & \quad \text{temporary} \\
\texttt{/lib/cpp} & \quad \text{preprocessor} \\
\texttt{/lib/c[01]} & \quad \text{PDP-11 compiler, cc} \\
\texttt{/usr/lib/comp} & \quad \text{compiler, pcc} \\
\texttt{/lib/ccom} & \quad \text{VAX compiler, cc} \\
\texttt{/lib/c2} & \quad \text{optional optimizer} \\
\texttt{/lib/occ*} & \quad \text{backup compiler, occ} \\
\texttt{/lib/ncc*} & \quad \text{test compiler, ncc} \\
\texttt{/lib/fcl} & \quad \text{PDP-11 floating-point compiler, cc} \\
\texttt{/lib/crt0.o} & \quad \text{runtime startoff} \\
\texttt{/lib/mcrt0.o} & \quad \text{startoff for profiling} \\
\texttt{/lib/fcrt0.o} & \quad \text{startoff for floating-point interpretation} \\
\texttt{/lib/libc.a} & \quad \text{standard library, see (3)} \\
\texttt{/usr/include} & \quad \text{standard directory for \#include files} \\
\end{align*}

SEE ALSO


B. W. Kernighan, \textit{Programming in C — A Tutorial}.


adb(1), as(1), ld(1), prof(1), monitor(3C).

DIAGNOSTICS

The diagnostics produced by \textit{C} itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor. Of these, the most mystifying are from the PDP-11 assembler, in particular m, which means a multiply-defined external symbol (function or data).
NAME
cd — change working directory

SYNOPSIS
   cd [ directory ]

DESCRIPTION
If specified, directory becomes the new working directory; otherwise, the value of the shell parameter $HOME is used. The process must have execute (search) permission in directory.

Because a new process is created to execute each command, cd would be ineffective if it were written as a normal command; therefore, it is recognized and executed by the shell.

SEE ALSO
   pwd(1), sh(1), chdir(2).
NAME
cdc — change the delta commentary of an SCCS delta

SYNOPSIS
ccdc -rSID [-m[mrlist]] [-y[comment]] files

DESCRIPTION
Cdc changes the delta commentary, for the SID specified by the -r keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the delta(1) command (-m and -y keyletters).

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

Arguments to cdc, which may appear in any order, consist of keyletter arguments, and file names.

All the described keyletter arguments apply independently to each named file:

-rSID Used to specify the SCCS IDentification (SID) string of a delta for which the delta commentary is to be changed.

-m[mrlist] If the SCCS file has the v flag set (see admin(1)) then a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the -r keyletter may be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of delta(1). In order to delete an MR, precede the MR number with the character ! (see EXAMPLES). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If -m is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the comments? prompt (see -y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value (see admin(1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, cdc terminates.
and the delta commentary remains unchanged.

\(-y[comment] \) Arbitrary text used to replace the comment(s) already existing for the delta specified by the \(-r\) keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null comment has no effect.

If \(-y\) is not specified and the standard input is a terminal, the prompt comments? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

The exact permissions necessary to modify the SCCS file are documented in the Source Code Control System User's Guide. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

**EXAMPLES**

```
cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file
```

adds bl78-12345 and bl79-00001 to the MR list, removes bl77-54321 from the MR list, and adds the comment trouble to delta 1.6 of s.file.

```
cdc -r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

**WARNINGS**

If SCCS file names are supplied to the \(cdc\) command via the standard input ( - on the command line), then the \(-m\) and \(-y\) keyletters must also be used.

**FILES**

x-file (see delta(1))
z-file (see delta(1))

**SEE ALSO**

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).


**DIAGNOSTICS**

Use help(1) for explanations.
NAME
chmod — change mode

SYNOPSIS
chmod mode file ...

DESCRIPTION
The permissions of each named file are changed according to mode, which may be absolute or symbolic. An absolute mode is an octal number constructed from the OR of the following modes:

<table>
<thead>
<tr>
<th>Octal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>set user ID on execution</td>
</tr>
<tr>
<td>2000</td>
<td>set group ID on execution</td>
</tr>
<tr>
<td>1000</td>
<td>sticky bit, see chmod(2)</td>
</tr>
<tr>
<td>0400</td>
<td>read by owner</td>
</tr>
<tr>
<td>0200</td>
<td>write by owner</td>
</tr>
<tr>
<td>0100</td>
<td>execute (search in directory) by owner</td>
</tr>
<tr>
<td>0070</td>
<td>read, write, execute (search) by group</td>
</tr>
<tr>
<td>0007</td>
<td>read, write, execute (search) by others</td>
</tr>
</tbody>
</table>

A symbolic mode has the form:

[who] op permission [ op permission ]

The who part is a combination of the letters u (for user's permissions), g (group) and o (other). The letter a stands for ugo, the default if who is omitted.

Op can be + to add permission to the file's mode, − to take away permission, or = to assign permission absolutely (all other bits will be reset).

Permission is any combination of the letters r (read), w (write), x (execute), s (set owner or group ID) and t (save text — sticky); u, g or o indicate that permission is to be taken from the current mode. Omitting permission is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter s is only useful with u or g and t only works with u.

Only the owner of a file (or the super-user) may change its mode.

EXAMPLES
The first example denies write permission to others, the second makes a file executable:

chmod o−w file

chmod +x file

SEE ALSO
ls(1), chmod(2).
NAME
chown, chgrp — change owner or group

SYNOPSIS
chown owner file ...
chgrp group file ...

DESCRIPTION
Chown changes the owner of the files to owner. The owner may be either a
decimal user ID or a login name found in the password file.

Chgrp changes the group ID of the files to group. The group may be either
a decimal group ID or a group name found in the group file.

FILES
/etc/passwd
/etc/group

SEE ALSO
chown(2), group(5), passwd(5).
NAME
chroot — change root directory for a command

SYNOPSIS
chroot newroot command

DESCRIPTION
The given command is executed relative to the new root. The meaning of
any initial slashes (/) in path names is changed for a command and any of
its children to newroot. Furthermore, the initial working directory is
newroot.

Notice that:
    chroot newroot command >x
will create the file x relative to the original root, not the new one.

This command is restricted to the super-user.

The new root path name is always relative to the current root: even if a
chroot is currently in effect, the newroot argument is relative to the current
root of the running process.

SEE ALSO
chdir(2).

BUGS
One should exercise extreme caution when referencing special files in the
new root file system.
NAME
crl - clear i-node

SYNOPSIS
crl file-system i-number ...

DESCRIPTION

Crl writes zeros on the 64 bytes occupied by the i-node numbered i-number. File-system must be a special file name referring to a device containing a file system. After crl is executed, any blocks in the affected file will show up as "missing" in an fsck(1M) of the file-system. This command should only be used in emergencies and extreme care should be exercised.

Read and write permission is required on the specified file-system device. The i-node becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an i-node which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry will destroy the new file. The new entry will again point to an unallocated i-node, so the whole cycle is likely to be repeated again and again.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M), fs(5).

BUGS

If the file is open, crl is likely to be ineffective.
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.

SEE ALSO

comm(1), diff(1).

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.
NAME
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(1) and output resulting from use of the tbl(1) preproces­
sor.

If the -b option is given, col assumes that the output device in use is not
capable of backspacing. In this case, if two or more characters are to
appear in the same place, only the last one read will be output.

Although col accepts half-line motions in its input, it normally does not
emit them on output. Instead, text that would appear between lines is
moved to the next lower full-line boundary. This treatment can be
suppressed by the -f (fine) option; in this case, the output from col may
contain forward half-line-feeds (ESC-9), but will still never contain either
kind of reverse line motion.

Unless the -x option is given, col will convert white space to tabs on out­
put wherever possible to shorten printing time.

The ASCII control characters SO (\017) and SI (\016) are assumed by col to
start and end text in an alternate character set. The character set to which
each input character belongs is remembered, and on output SI and SO
characters are generated as appropriate to ensure that each character is
printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab,
return, new-line, SI, SO, VT (\013), and ESC followed by 7, 8, or 9. The
VT character is an alternate form of full reverse line-feed, included for
compatibility with some earlier programs of this type. All other non­
printing characters are ignored.

Normally, col will ignore any unknown to it escape sequences found in its
input; the -p option may be used to cause col to output these sequences as
regular characters, subject to overprinting from reverse line motions. The
use of this option is highly discouraged unless the user is fully aware of the
textual position of the escape sequences.

SEE ALSO

nroff(1), tbl(1).

NOTES

The input format accepted by col matches the output produced by nroff(1)
with either the -T37 or -Tlp options. Use -T37 (and the -f option of
col) if the ultimate disposition of the output of col will be a device that can
interpret half-line motions, and -Tlp otherwise.

BUGS

Cannot back up more than 128 lines.
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
comb — combine SCCS deltas

SYNOPSIS
comb [-o] [-s] [-psid] [-clist] files

DESCRIPTION
Comb generates a shell procedure (see sh(1)) which, when run, will recon-
struct the given SCCS files. The reconstructed files will, hopefully, be small-
er than the original files. The arguments may be specified in any order,
but all keyletter arguments apply to all named SCCS files. If a directory is
named, comb behaves as though each file in the directory were specified as
a named file, except that non-SCCS files (last component of the path name
does not begin with s.) and unreadable files are silently ignored. If a name
of — is given, the standard input is read; each line of the standard input is
taken to be the name of an SCCS file to be processed; non-SCCS files and
unreadable files are silently ignored.

The generated shell procedure is written on the standard output.
The keyletter arguments are as follows. Each is explained as though only
one named file is to be processed, but the effects of any keyletter argument
apply independently to each named file.

-pSid The SCCS Identifier string (SID) of the oldest delta to be
preserved. All older deltas are discarded in the reconstructed file.

-clist A list (see get(1) for the syntax of a list) of deltas to be preserved.
All other deltas are discarded.

-o For each get -e generated, this argument causes the reconstructed
file to be accessed at the release of the delta to be created, other-
wise the reconstructed file would be accessed at the most recent
ancestor. Use of the -o keyletter may decrease the size of the
reconstructed SCCS file. It may also alter the shape of the delta
tree of the original file.

-s This argument causes comb to generate a shell procedure which,
when run, will produce a report giving, for each file: the file name,
size (in blocks) after combining, original size (also in blocks), and
percentage change computed by:

100 • (original - combined) / original

It is recommended that before any SCCS files are actually com-
bined, one should use this option to determine exactly how much
space is saved by the combining process.

If no keyletter arguments are specified, comb will preserve only leaf deltas
and the minimal number of ancestors needed to preserve the tree.

FILES
s.COMB The name of the reconstructed SCCS file.
comb?????? Temporary.

SEE ALSO
admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).

DIAGNOSTICS
Use help(1) for explanations.

BUGS
Comb may rearrange the shape of the tree of deltas. It may not save any
space; in fact, it is possible for the reconstructed file to actually be larger
than the original.

- 1 -
NAME
comm — select or reject lines common to two sorted files

SYNOPSIS
comm [ - [ 123 ] ] file1 file2

DESCRIPTION
Comm reads file1 and file2, which should be ordered in ASCII collating sequence (see sort(1)), and produces a three-column output: lines only in file1; lines only in file2; and lines in both files. The file name — means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus comm -12 prints only the lines common to the two files; comm -23 prints only lines in the first file but not in the second; comm -123 is a no-op.

SEE ALSO
cmp(1), diff(1), sort(1), uniq(1).
NAME
config — configure a UNIX system

SYNOPSIS
/etc/config [ -t ] [ -l file ] [ -c file ] [ -m file ] dfile

DESCRIPTION
Config is a program that takes a description of a UNIX system and generates two files. One file provides information regarding the interface between the hardware and device handlers. The other file is a C program defining the configuration tables for the various devices on the system.

The -l option specifies the name of the hardware interface file; low.s is the default name on the PDP-11; univect.c is the default name on the VAX-11/780.

The -c option specifies the name of the configuration table file; conf.c is the default name.

The -m option specifies the name of the file that contains all the information regarding supported devices; /etc/master is the default name. This file is supplied with the UNIX system and should not be modified unless the user fully understands its construction.

The -t option requests a short table of major device numbers for character and block type devices. This can facilitate the creation of special files.

The user must supply dfile; it must contain device information for the user's system. This file is divided into two parts. The first part contains physical device specifications. The second part contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment.

All configurations are assumed to have the following devices:

    one DL11 (for the system console)
    one KW11-L line clock or KW11-P programmable clock

with standard interrupt vectors and addresses. These two devices must not be specified in dfile. Note that UNIX needs only one clock, but can handle both types.

First Part of dfile
Each line contains four or five fields, delimited by blanks and/or tabs in the following format:

    devname vector address bus number

where devname is the name of the device (as it appears in the /etc/master device table), vector is the interrupt vector location (octal), address is the device address (octal), bus is the bus request level (4 through 7), and number is the number (decimal) of devices associated with the corresponding controller; number is optional, and if omitted, a default value which is the maximum value for that controller is used.

There are certain drivers that may be provided with the system, that are actually pseudo-device drivers; that is, there is no real hardware associated with the driver. Drivers of this type are identified on their respective manual entries. When these devices are specified in the description file, the interrupt vector, device address, and bus request level must all be zero.
Second Part of dfile

The second part contains three different types of lines. Note that all specifications of this part are required, although their order is arbitrary.

1. Root/pipe/dump device specification

Three lines of three fields each:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>devname</td>
<td>minor</td>
</tr>
<tr>
<td>pipe</td>
<td>devname</td>
<td>minor</td>
</tr>
<tr>
<td>dump</td>
<td>devname</td>
<td>minor</td>
</tr>
</tbody>
</table>

where minor is the minor device number (in octal).

2. Swap device specification

One line that contains five fields as follows:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>swap</td>
<td>devname</td>
<td>minor</td>
<td>swplo</td>
</tr>
</tbody>
</table>

where swplo is the lowest disk block (decimal) in the swap area and nswap is the number of disk blocks (decimal) in the swap area.

3. Parameter specification

Thirteen lines of two fields each as follows (number is decimal):

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>buffers</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>sbufs</td>
<td>number</td>
<td>(not on the VAX-11/780)</td>
</tr>
<tr>
<td>inodes</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>files</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>mounts</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>coremap</td>
<td>number</td>
<td>(not on the VAX-11/780)</td>
</tr>
<tr>
<td>swapmap</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>calls</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>procs</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>maxproc</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>texts</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>clists</td>
<td>number</td>
<td></td>
</tr>
<tr>
<td>power</td>
<td>0 or 1</td>
<td></td>
</tr>
</tbody>
</table>

EXAMPLE

Suppose we wish to configure a PDP-11/70 system with the following devices:

- one RP04 disk drive controller with 6 drives
- one DH11 asynchronous multiplexer with 16 lines (default number)
- one DM11 modem control with 16 lines (for the DH11)
- one DH11 asynchronous multiplexer with 8 lines
- one DM11 modem control with 8 lines (for the DH11)
- one LP11 line printer
- one TU16 tape drive controller with 2 drives
- one DL11 asynchronous interface

Note that UNIX only supports DH11 units that require corresponding DM11 units. It is wise to specify them in DH-DM pairs to facilitate understanding the configuration. Note also that, in the preceding case, the DL11 that is specified is in addition to the DL11 that was part of the initial system. We must also specify the following parameter information:

- root device is an RP04 (drive 0, section 0)
- pipe device is an RP04 (drive 0, section 0)
- swap device is an RP04 (drive 1, section 4),
  with a swplo of 6000 and an nswap of 2000
- dump device is a TU16 (drive 0)
- number of buffers is 35
number of *system addressable* buffers is 12
number of processes is 150
maximum number of processes per user ID is 25
number of mounts is 8
number of inodes is 120
number of files is 120
number of calls is 30
number of texts is 35
number of character buffers is 150
number of coremap entries is 50
number of swapmap entries is 50
power fail recovery is to be included

The actual system configuration would be specified as follows:

```
rp04 254 776700 5 6
dh11 320 760020 5
dm11 300 770500 4
dh11 330 760040 5 8
dm11 304 770510 4 8
lp11 200 775514 5
tu16 224 772440 5 2
dl11 350 775610 5
root rp04 0
pipe rp04 0
swap rp04 14 6000 2000
dump tu16 0
```

*Comments may be inserted in this manner

buffers 35
sabufs 12
procs 150
maxproc 25
mounts 8
inodes 120
files 120
calls 30
texts 35
clists 150
coremap 50
swapmap 50
power 1

FILES

/etc/master default input master device table
low.s default output hardware interface file for PDP-11
univec.c default output hardware interface file for the VAX-11/780
cnf.c default output configuration table file

SEE ALSO

master(5).
Setting Up UNIX.

DIAGNOSTICS

Diagnostics are routed to the standard output and are self-explanatory.

BUGS

The -t option does not know about devices that have aliases. For example, an RP06 (an alias for an RP04) will show up as an RP04; however, the major device numbers are always correct.
NAME
   cp, ln, mv — copy, link or move files

SYNOPSIS
   cp  file1  [ file2  ...] target
   ln  file1  [ file2  ...] target
   mv  file1  [ file2  ...] target

DESCRIPTION
   File1 is copied (linked, moved) to target. Under no circumstance can file1
   and target be the same. If target is a directory, then one or more files are
   copied (linked, moved) to that directory.

   If mv determines that the mode of target forbids writing, it will print the
   mode (see chmod(2)) and read the standard input for one line (if the stan‐
   dard input is a terminal); if the line begins with y, the move takes place; if
   not, mv exits.

   Only mv will allow file1 to be a directory, in which case the directory
   rename will occur only if the two directories have the same parent.

SEE ALSO
   cpio(1), link(1M), rm(1), chmod(2).

BUGS
   If file1 and target lie on different file systems, mv must copy the file and
   delete the original. In this case the owner name becomes that of the copy‐
   ing process and any linking relationship with other files is lost.

   Ln will not link across file systems.
NAME
cpio — copy file archives in and out

SYNOPSIS
  cpio -o [ acBv ]
  cpio -i [ Bcdmrtuv6 ] [ patterns ]
  cpio -p [ adlmruv ] directory

DESCRIPTION
  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.

  Cpio -i (copy in) extracts from the standard input (which is assumed to be
  the product of a previous cpio -o) the names of files selected by zero or
  more patterns given in the name-generating notation of sh(1). In patterns,
  meta-characters ?, .*, and [ ... ] match the slash / character. The default for
  patterns is * (i.e., select all files).

  Cpio -p (pass) copies out and in in a single operation. Destination path
  names are interpreted relative to the named directory.

  The meanings of the available options are:
  a   Reset access times of input files after they have been copied.
  B   Input/output is to be blocked 5,120 bytes to the record (does not
      apply to the pass option; meaningful only with data directed to or
      from /dev/rmt?).
  d   Directories are to be created as needed.
  c   Write header information in ASCII character form for portability.
  r   Interactively rename files. If the user types a null line, the file is
      skipped.
  t   Print a table of contents of the input. No files are created.
  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)).
  l   Whenever possible, link files rather than copying them. Usable
      only with the -p option.
  m   Retain previous file modification time. This option is ineffective on
      directories that are being copied.
  6   Process an old (i.e., UNIX Sixth Edition format) file. Only useful
      with -l (copy in).

EXAMPLES
  The first example below copies the contents of a directory into an archive;
  the second duplicates a directory hierarchy:
  
  ls | cpio -o >/dev/mt0
  cd olddir
  find . -print | cpio -pdl newdir

  The trivial case "find . -print | cpio -oB >/dev/rmt0" can be handled
  more efficiently by:
  
  find . -cpio /dev/rmt0

SEE ALSO
  ar(1), find(1), cpio(5).
BUGS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files.
NAME
  crash — examine system images

SYNOPSIS
  /etc/crash [ system ] [ namelist ] [ ka6 ]

DESCRIPTION
  Crash is an interactive utility for examining an operating system core image. It has facilities for interpreting and formatting the various control structures in the system and certain miscellaneous functions that are useful when perusing a dump.

The arguments to crash are the file name where the system image can be found, a namelist file to be used for symbol values, and the segment address of the initial process to be examined. The current process can be changed via subsequent commands. The default values are /dev/mem, /unix, and the location of the swapper, process 0; hence, crash with no arguments can be used to examine an active system. If a system image file is given, it is assumed to be a system core dump and the initial process is set to be that of the process running at the time of the crash. This is determined by the value of ka6 stored in a fixed location by the system dump mechanism.

COMMANDS
  Input to crash is typically of the form:
    command [ options ] [ structures to be printed ].
  When allowed, options will modify the format of the print out. If no specific structure elements are specified, all valid entries will be used. As an example, proc - 12 15 3 would print process table slots 12, 15 and 3 in a long format, while proc would print the entire process table in the standard format. The current repertory consists of:

  ka6 [ segment address ]
    Print the location of the current process if no argument is given, or set the location to that of the supplied value.

  u
    Print the user structure of the current process as determined by the value of ka6.

  trace[-r]
    Generate a kernel stack trace of the current process. If the -r option is used, the trace begins at the saved stack frame pointer in r5. Otherwise the trace starts at the bottom of the stack and attempts to find valid stack frames deeper in the stack.

  r5 [ stack frame ]
    Print the program's idea of the start of the current stack frame (set initially from a fixed location in the dump) if no argument is given, or set the frame pointer to the supplied value.

  stack
    Format an octal dump of the kernel stack of the current process. The addresses shown are virtual system data addresses rather than true physical locations.

  proc [ -[r] ] [ list of process table entries ]
    Format the process table. The -r option causes only runnable processes to be printed. The - alone generates a longer listing.

  inode [ - ] [ list of inode table entries ]
    Format the inode table. The - option will also print the inode data block addresses.
file [ list of file table entries ]
   Format the file table.

mount [ list of mount table entries ]
   Format the mount table.

text [ list of text table entries ]
   Format the text table.

tty [ type ] [ - ] [ list of tty entries ]
   Print the tty structures. The type argument determines which structure will be used (such as kl or dh; the last type is remembered). The - option prints the stty parameters for the given line.

stat
   Print certain statistics found in the dump. These include the panic string, time of crash, system name, and the registers saved in low memory by the dump mechanism.

var
   Print the tunable system parameters.

buf [ list of buffer headers ]
   Format the system buffer headers.

buffer [ format ] [ list of buffers ]
   Print the data in a system buffer according to format. Valid formats include decimal, octal, character, byte, directory, inode, and write. The last creates a file containing the buffer data.

callout
   Print all entries in the callout table.

map [ list of map names ]
   Format the named system map structures.

nm [ list of symbols ]
   Print symbol value and type as found in the namelist file.

ts [ list of text addresses ]
   Find the closest text symbols to the given addresses.

ds [ list of data addresses ]
   Find the closest data symbols to the given addresses.

od [ symbol or data address ] [ count ] [ format ]
   Dump count data values starting at the symbol value or address given according to format. Allowable formats are octal, decimal, character, or byte.

! Escape to shell.

q Exit from crash.

? Print synopsis of commands.

ALIASES
There are built in aliases for many of the commands and formats. In general, the first letter of a name is satisfactory, thus, k is a shorthand notation for kernel. Exceptions are x for text and e for decimal.

FILES
/dev/mem         default system image file
/unix            default namelist file
buf.#            files created containing buffer data

SEE ALSO
   crash(8).
NAME
cref — make cross-reference listing

SYNOPSIS
cref [ -acilnostux123 ] files

DESCRIPTION
Cref makes a cross-reference listing of assembler or C programs; files are searched for symbols in the appropriate syntax.

The output report is in four columns:
1. symbol;
2. file name;
3. see below;
4. text as it appears in the file.

Cref uses either an ignore file or an only file. If the -i option is given, the next argument is taken to be an ignore file; if the -o option is given, the next argument is taken to be an only file. Ignore and only files are lists of symbols separated by new-lines. All symbols in an ignore file are ignored in columns 1 and 3 of the output. If an only file is given, only symbols in that file will appear in column 1. Only one of these options may be given; the default setting is -i using the default ignore file (see FILES below).

Assembler pre-defined symbols or C keywords are ignored. The -s option causes current symbols to be put in column 3. In the assembler, the current symbol is the most recent name symbol; in C, the current function name. The -l option causes the line number within the file to be put in column 3.

The -t option causes the next available argument to be used as the name of the intermediate file (instead of the temporary file /tmp/crt??). This file is created and is not removed at the end of the process.

The cref options are:

- a assembler format (default)
- c C format input
- i use an ignore file (see above)
- l put line number in column 3 (instead of current symbol)
- n omit column 4 (no context)
- o use an only file (see above)
- s current symbol in column 3 (default)
- t user-supplied temporary file
- u print only symbols that occur exactly once
- x print only C external symbols
- 1 sort output on column 1 (default)
- 2 sort output on column 2
- 3 sort output on column 3.

FILES
/tmp/crt??          temporaries
/usr/lib/cref/aign  default assembler ignore file
/usr/lib/cref/atab  grammar table for assembler files
/usr/lib/cref/cign  default C ignore file
/usr/lib/cref/ctab  grammar table for C files
/usr/lib/cref/crpost post-processor
/usr/lib/cref/upost post-processor for -u option

SEE ALSO
as(1), cc(1), sort(1), xref(1).
BUGS

Cref inserts an ASCII DEL character into the intermediate file after the eighth character of each name that is eight or more characters long in the source file.
NAME
cron — clock daemon

SYNOPSIS
/etc/cron

DESCRIPTION
Cron executes commands at specified dates and times according to the
instructions in the file /usr/lib/crontab. Because cron never exits, it
should be executed only once. This is best done by running cron from the
initialization process through the file /etc/rc (see init(8)).

The file crontab consists of lines of six fields each. The fields are separated
by spaces or tabs. The first five are integer patterns that specify the minute
(0-59), hour (0-23), day of the month (1-31), month of the year (1-12),
and day of the week (0-6, with 0=Sunday). Each of these patterns may
contain:

   a number in the (respective) range indicated above;
   two numbers separated by a minus (indicating an inclusive range);
   a list of numbers separated by commas (meaning all of these num-
      bers); or
   an asterisk (meaning all legal values).

The sixth field is a string that is executed by the shell at the specified
time(s). A % in this field is translated into a new-line character. Only the
first line (up to a % or the end of line) of the command field is executed by
the shell. The other lines are made available to the command as standard
input.

Cron examines crontab once a minute to see if it has changed; if it has,
cron reads it. Thus it takes only a minute for entries to become effective.

FILES
/usr/lib/crontab
/usr/lib/cronlog

SEE ALSO
sh(1), init(8).

DIAGNOSTICS
A history of all actions by cron are recorded in /usr/lib/cronlog.

BUGS
Cron reads crontab only when it has changed, but it reads the in-core ver-

tion of that table once a minute. A more efficient algorithm could be used.
The overhead in running cron is about one percent of the CPU, exclusive of
any commands executed by cron.
CRYPT(1)

NAME
crypt - encode/decode

SYNOPSIS
 crypt [ password ]

DESCRIPTION
Crypt reads from the standard input and writes on the standard output.
The password is a key that selects a particular transformation. If no password is given, crypt demands a key from the terminal and turns off printing while the key is being typed in. Crypt encrypts and decrypts with the same key:

crypt key <clear >cypher
crypt key <cypher | pr

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; "sneak paths" by which keys or clear text can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

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

Since the key is an argument to the crypt command, it is potentially visible to users executing ps(1) or a derivative. To minimize this possibility, crypt takes care to destroy any record of the key immediately upon entry. The choice of keys and key security are the most vulnerable aspect of crypt.

FILES
/dev/tty for typed key

SEE ALSO
ed(1), makekey(8).

BUGS
If output is piped to nroff(1) and the encryption key is not given on the command line, crypt can leave terminal modes in a strange state (see stty(1)).
NAME
csplit – context split

SYNOPSIS
csplit [-s] [-k] [-f prefix] file arg1 [... argn]

DESCRIPTION
Csplit reads file and separates it into n+1 sections, defined by the arguments arg1 ... argn. By default the sections are placed in xx00 ... xxn (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.

The options to csplit are:

-s Csplit normally prints the character counts for each file created. If the -s option is present, csplit suppresses the printing of all character counts.

-k Csplit normally removes created files if an error occurs. If the -k option is present, csplit leaves previously created files intact.

-f prefix If the -f option is used, the created files are named prefix00 ... prefixn. The default is xx00 ... xxn.

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

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

%rexp% This argument is the same as /rexp/, except that no file is created for the section.

Inno A file is to be created from the current line up to (but not including) Inno. The current line becomes Inno.

{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 Inno, the file will be split every Inno lines (num times) from that point.

Enclose all rexp type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. Csplit does not affect the original file; it is the users responsibility to remove it.

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

This example creates four files, cobol00 ... cobol03. After editing the "split" files, they can be recombined as follows:
cat cobol0[0−3] > file

Note that this example overwrites the original file.

csplit −k file 100 {99}

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

csplit −k prog.c '%main(%/}/+1' {20}

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

SEE ALSO
ed(1), sh(1), regexp(7).

DIAGNOSTICS
Self explanatory except for:
arg − out of range
which means that the given argument did not reference a line between the current position and the end of the file.
CT(1C)

NAME
cr — call terminal

SYNOPSIS
cr [ -h ] [ -v ] [ -wn ] [ -sspeed ] telno

DESCRIPTION
Ct dials the phone number of a modem that is attached to a terminal, and
spawns a login process to that terminal. Telno is the telephone number,
with minus signs at appropriate places for delays.

Ct determines which dialers are associated with lines that are set to the
appropriate speed by examining the file /usr/lib/uucp/L-devices. If all
such available dialers are busy, ct will ask if it should wait for a line, and if
so, for how many minutes it should wait before it gives up. Cct will con­
tinue to try to open the dialers at one-minute intervals until the specified
limit is exceeded. The dialogue may be overridden by specifying the -wn
option, where n is the maximum number of minutes that ct is to wait for a
line.

Normally, ct will hang up the current line, so that that line can answer the
incoming call. The -h option will prevent this action. If the -v option is
used, ct will send a running narrative to standard error.

The data rate may be set with the -s option, where speed is expressed in
baud. The default rate is 300.

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

FILES
/usr/lib/uucp/L-devices

SEE ALSO
cr(1C), login(1), uucp(1C), dn(4), getty(8).
NAME
cu — call another UNIX system

SYNOPSIS

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. *speed* gives the transmission speed (110, 150, 300, 1200, 4800, 9600); 300 is the default value. Most of our modems restrict us to choose between 300 and 1200. Directly connected lines may be set to other speeds.

The `-a` and `-l` values may be used to specify device names for the ACU and communications line devices. They can be used to override searching for the first available ACU with the right speed. The `-h` option emulates local echo, supporting calls to other computer systems which expect terminals to be in half-duplex mode. The `-e` (or `-o`) option designates that even (or odd) parity is to be generated for data sent to the remote. *Telno* is the telephone number, with equal signs for secondary dial tone or minus signs for delays, at appropriate places. The string *dir* for *telno* must be used for directly connected lines, and implies a null ACU.

Cu will try each line listed in the file `/usr/lib/uucp/L-devices` until it finds an available line with appropriate attributes or runs out of entries. After making the connection, cu runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with `~`, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with `~`, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with `~` have special meanings.

The *transmit* process interprets the following:

- `~` terminate the conversation.
- `~!` escape to an interactive shell on the local system.
- `~!cmd` run *cmd* locally and send its output to the remote system.
- `~$cmd` run *cmd* on the local system (via `sh -c`).
- `~%take from [ to ]` copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
- `~%put from [ to ]` copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.
- `~...` send the line `~...` to the remote system.
- `~nostop` turn off the DC3/DC1 input control protocol for the remainder of the session. 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. A line from the remote that begins with `~>` initiates an output diversion to a file. The complete sequence is:
```
\text{->[>]}: \text{file}
\text{zero or more lines to be written to file}
```

Data from the remote is diverted (or appended, if `>>` is used) to file. The trailing `>` terminates the diversion.

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

The use of `\text{~%take}` requires the existence of `echo(1)` and `cat(1)` on the remote system. Also, `stty tabs` mode should be set on the remote system if tabs are to be copied without expansion.

**FILES**
- `/usr/lib/uucp/L-devices`
- `/usr/spool/uucp/LCK..(tty-device)`
- `/dev/null`

**SEE ALSO**
- `cat(1)`, `echo(1)`, `stty(1)`, `uucp(1C)`, `dh(4)`, `dn(4)`, `tty(4)`.

**DIAGNOSTICS**
Exit code is zero for normal exit, non-zero (various values) otherwise.

**BUGS**
There is an artificial slowing of transmission by `cu` during the `\text{~%put}` operation so that loss of data is unlikely.
NAME

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

SYNOPSIS

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

DESCRIPTION

Use `cut` to cut out columns from a table or fields from each line of a file; in
data base parlance, it implements the projection of a relation. The fields as
specified by `list` can be fixed length, i.e., character positions as on a pun­
ched card (\(-e\) option), or the length can vary from line to line and be
marked with a field delimiter character like `tab` (\(-f\) option). `Cut` can be
used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

\textbf{list} A comma-separated list of integer field numbers (in increasing
order), with optional \(-\) to indicate ranges as in the \(-o\) option of
\texttt{nroff/troff} for page ranges; e.g., \(1,4,7; 1\ -\ 3,8; -5,10\) (short for
\(1\ -\ 5,10\)); or \(3-\) (short for third through last field).

\textbf{-clist} The `list` following \(-e\) (no space) specifies character positions
(e.g., \(-c-1\ -72\) would pass the first 72 characters of each line).

\textbf{-flist} 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.

\textbf{-dchar} 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.

\textbf{-s} Suppresses lines with no delimiter characters in case of \(-f\)
option. Unless specified, lines with no delimiters will be passed
through untouched.

Either the \(-e\) or \(-f\) option must be specified.

HINTS

Use `grep(1)` to make horizontal "cuts" (by context) through a file, or
`paste(1)` to put files together column-wise (i.e., horizontally). To reorder
columns in a table, use `cut` and `paste`.

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

\textbf{line too long} A line can have no more than 511 characters or
fields.

\textbf{bad list for c/f option} Missing \(-e\) or \(-f\) option or incorrectly specified `list'.
No error occurs if a line has fewer fields than the `list'
calls for.

\textbf{no fields} The `list` is empty.

SEE ALSO

grep(1), paste(1).
NAME
cw, checkcw — prepare constant-width text for troff

SYNOPSIS
cw [-lxx] [-rxx] [-fn] [-t] [+t] [-d] [files]
checkcw [-lxx] [-rxx] files

DESCRIPTION
Cw is a preprocessor for troff(1) input files that contain text to be typeset in the constant-width (CW) font.

Text typeset with the CW font resembles the output of terminals and of line printers. This font is used to typeset examples of programs and of computer output in user manuals, programming texts, etc. (An earlier version of this font was used in typesetting The C Programming Language by B. W. Kernighan and D. M. Ritchie). It has been designed to be quite distinctive (but not overly obtrusive) when used together with the Times Roman font.

Because the CW font contains a "non-standard" set of characters and because text typeset with it requires different character and inter-word spacing than is used for "standard" fonts, documents that use the CW font must be preprocessed by cw.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$%&'( )++.,-/:;=?[\]^_`abcdefghijklmnoprstuvwxyz
```

plus eight non-ASCII characters represented by four-character troff(1) names (in some cases attaching these names to "non-standard" graphics), as follows:

<table>
<thead>
<tr>
<th>Character</th>
<th>Symbol</th>
<th>Traff Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Cents&quot; sign</td>
<td>$</td>
<td>(ct</td>
</tr>
<tr>
<td>EBCDIC &quot;not&quot; sign</td>
<td></td>
<td>(no</td>
</tr>
<tr>
<td>Left arrow</td>
<td>&lt;</td>
<td>(&lt;-</td>
</tr>
<tr>
<td>Right arrow</td>
<td>&gt;</td>
<td>(-&gt;</td>
</tr>
<tr>
<td>Down arrow</td>
<td>^</td>
<td>(da</td>
</tr>
<tr>
<td>Vertical single quote</td>
<td>`</td>
<td>(fm</td>
</tr>
<tr>
<td>Control-shift indicator</td>
<td>\</td>
<td>(dg</td>
</tr>
<tr>
<td>Visible space indicator</td>
<td>=</td>
<td>(sq</td>
</tr>
<tr>
<td>Hyphen</td>
<td>-</td>
<td>(hy</td>
</tr>
</tbody>
</table>

The hyphen is a synonym for the unadorned minus sign (-). Certain versions of cw recognize two additional names: \(ua for an up arrow and \(1h for a diagonal left-up (home) arrow.

Cw recognizes five request lines, as well as user-defined delimiters. The request lines look like troff(1) macro requests, and are copied in their entirety by cw onto its output; thus, they can be defined by the user as troff(1) macros; in fact, the .CW and .CN macros should be so defined (see HINTS below).

The five requests are:

.CW Start of text to be set in the CW font; .CW causes a break; it can take precisely the same options, in precisely the same format, as are available on the cw command line.

.CN End of text to be set in the CW font; .CN causes a break; it can take the same options as are available on the cw command line.
.CD  Change delimiters and/or settings of other options; takes the same
options as are available on the cw command line.

.CP arg1 arg2 arg3 ... argn
All the arguments (which are delimited like troff(1) macro
arguments) are concatenated, with the odd-numbered arguments
set in the CW font and the even-numbered ones in the prevailing
font.

.FC arg1 arg2 arg3 ... argn
Same as .CP, except that the even-numbered (rather than odd-
numbered) arguments are set in the CW font.

The .CW and .CN requests are meant to bracket text (e.g., a program
fragment) that is to be typeset in the CW font "as is." Normally, cw operates
in the transparent mode. In that mode, except for the .CD request and the
nine special four-character names listed in the table above, every character
between .CW and .CN request lines stands for itself. In particular, cw
arranges for periods (.) and apostrophes (') at the beginning of lines, and
backslashes (\) and ligatures (fi, ff, etc.) everywhere to be "hidden"
from troff(1). The transparent mode can be turned off (see below), in
which case normal troff(1) rules apply. In any case, cw hides from the user
the effect of the font changes generated by the .CW and .CN requests.

The only purpose of the .CD request is to allow the changing of various
options other than just at the beginning of a document.

The user can also define delimiters. The left and right delimiters perform
the same function as the .CW/.CN requests; they are meant, however, to
enclose CW "words" or "phrases" in running text (see the example under
BUGS below). Cw treats text enclosed by delimiters in precisely the same
manner as text bracketed by .CW/.CN pairs, except that, for aesthetic
reasons, spaces in text bracketed by .CW/.CN pairs have the same width
as any other CW character, while spaces between delimiters are half as
wide, so that they have the same width as spaces in the prevailing text (but
are not adjustable).

Delimiters have no special meaning inside .CW/.CN pairs.

The options are:

- `lx` The one- or two-character string `xx` becomes the left delimiter; if
  `xx` is omitted, the left delimiter becomes undefined, which it is initially.

- `xx` Same for the right delimiter. The left and right delimiters may (but
  need not) be different.

- `fn` The CW font is mounted in font position `n`; acceptable values for `n`
  are 1, 2, and 3 (default is 3, replacing the bold font). This option
  is only useful at the beginning of a document.

- `t` Turn transparent mode off.

+ `t` Turn transparent mode on (this is the initial default).

- `d` Print current option settings on file descriptor 2 in the form of
  troff(1) comment lines. This option is meant for debugging.

Cw reads the standard input when no files are specified, so it can be used
as a filter. Typical usage is:

```
cw files | troff ...
```

Checkcw checks that left and right delimiters, as well as the .CW/.CN
pairs, are properly balanced. It prints out all offending lines.
Typical definitions of the .CW and .CN macros meant to be used with the \texttt{mm(7)} macro package:

\begin{verbatim}
.de CW
.DS I
.ps 9
.vs 10.5p
.ta 16m/3u 32m/3u 48m/3u 64m/3u 80m/3u 96m/3u ... 
 ..
de CN
.ta 0.5i 1i 1.5i 2i 2.5i 3i 3.5i 4i 4.5i 5i 5.5i 6i
.vs
.ps 
.DE
.. 
\end{verbatim}

At the very least, the .CW macro should invoke the \texttt{troff(1)} \texttt{no-fill (.nf)} mode.

When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point \textit{smaller} than the prevailing point size (the displayed definitions of .CW and .CN above are one point smaller than the running text on this page). The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.

Documents that contain CW text may also contain tables and/or equations. If this is the case, the order of preprocessing should be: \texttt{cw}, \texttt{tbl}, and \texttt{eqn}. Usually, the tables contained in such documents will not contain any CW text, although it is entirely possible to have elements of the table set in the CW font; of course, care must be taken that \texttt{tbl(1)} format information not be modified by \texttt{cw}. Attempts to set equations in the CW font are not likely to be either pleasing or successful.

In the CW font, overstriking is most easily accomplished with backspaces: letting \texttt{-} represent a backspace, \texttt{- - (dg)} yields \texttt{\textvisiblespace}. Because spaces (and, therefore backspaces) are half as wide between delimiters as inside .CW/.CN pairs (see above), two backspaces are required for each over-strike between delimiters.

FILES

\texttt{/usr/lib/font/ftCW} \hspace{1em} CW font-width table

SEE ALSO

\texttt{eqn(1), mmt(1), tbl(1), troff(1), mm(7), mv(7)}.

WARNINGS

If text preprocessed by \texttt{cw} is to make any sense, it must be set on a typesetter equipped with the CW font or on the MHCC STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

BUGS

Only a masochist would use periods (.) or backslashes (\textbackslash) as delimiters. Certain CW characters don’t concatenate gracefully with certain Times Roman characters, e.g., a CW ampersand (\&) followed by a Times Roman comma (,), in such cases, judicious use of \texttt{troff(1)} half- and quarter-spaces (\textbackslash{} and \textbackslash{}	extbackslash{}) is most salutary, e.g., one should use \texttt{\_\_\_\_\_<}, (rather than just plain \texttt{\_\_<}), to obtain \&, (assuming that \_ is used for both delimiters).

Using \texttt{cw} with \texttt{troff} is silly.

The output of \texttt{cw} is hard to read.

See also \texttt{BUGS} under \texttt{troff(1)}.
NAME

date — print and set the date

SYNOPSIS

date [ mmddhhmm[yy] ] [ +format ]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date
and time are printed. Otherwise, the current date is set. The first mm is
the month number; dd is the day number in the month; hh is the hour
number (24 hour system); the second mm is the minute number; yy is the
last 2 digits of the year number and is optional. For example:

    date 10080045

sets the date to Oct 8, 12:45 AM. The current year is the default if no year
is mentioned. The system operates in GMT. Date takes care of the conver­
sion to and from local standard and daylight time.

If the argument begins with +, the output of date is under the control of
the user. The format for the output is similar to that of the first argument
to printf(3S). All output fields are of fixed size (zero padded if necessary).
Each field descriptor is preceded by % and will be 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.

Field Descriptors:

- a insert a new-line character
- t insert a tab character
- m month of year — 01 to 12
- d day of month — 01 to 31
- y last 2 digits of year — 00 to 99
- D date as mm/dd/yy
- H hour — 00 to 23
- M minute — 00 to 59
- S second — 00 to 59
- T time as HH:MM:SS
- j Julian date — 001 to 366
- w day of week — Sunday = 0
- a abbreviated weekday — Sun to Sat
- h abbreviated month — Jan to Dec
- r time in AM/PM notation

EXAMPLE

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

would generate as output:

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

DIAGNOSTICS

No permission if you aren’t the super-user and you try to change the
date;

bad conversion if the date set is syntactically incorrect;

bad format character if the field descriptor is not recognizable.

FILES

/dev/kmem

- 1 -
NAME
dc — desk calculator

SYNOPSIS
dc [ file ]

DESCRIPTION
Dc is an arbitrary precision arithmetic package. Ordinarily it operates on
decimal integers, but one may specify an input base, output base, and a
number of fractional digits to be maintained. 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 fol­
lowing constructions are recognized:

number
The value of the number is pushed on the stack. A number is an
unbroken string of the digits 0—9. It may be preceded by an under­
score (_) to input a negative number. Numbers may contain decimal
points.

+ - / * %
The top two values on the stack are added (+), subtracted (−),
multiplied (*), divided (/), remaindered (%), or exponentiated (\(^\)).
The two entries are popped off the stack; the result is pushed on the
stack in their place. Any fractional part of an exponent is ignored.

sx
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.

Ix
The value in register x is pushed on the stack. The register x is not
altered. All registers start with zero value. If the I 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 pop­
ped 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.

√
replaces the top element on the stack by its square root. Any exist­
ting fractional part of the argument is taken into account, but oth­
erwise the scale factor is ignored.

!
interprets the rest of the line as a UNIX command.

c
All values on the stack are popped.
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.

The top value on the stack is popped and used as the number radix for further output.

O pushes the output base on the top of the stack.

k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

Z replaces the number on the top of the stack with its length.

? A line of input is taken from the input source (usually the terminal) and executed.

; : are used by bc for array operations.

EXAMPLE
This example prints the first ten values of n!:

[lal +dsa*pla10>y]sy
Osal
lyx.

SEE ALSO
bc(1), which is a preprocessor for dc providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS
x is unimplemented
where x is an octal number.

stack empty
for not enough elements on the stack to do what was asked.

Out of space
when the free list is exhausted (too many digits).

Out of headers
for too many numbers being kept around.

Out of pushdown
for too many items on the stack.

Nesting Depth
for too many levels of nested execution.
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 records before starting copy
seek=n seek n records from beginning of output file before copying

count=n copy only n input records
conv=ascii convert EBCDIC to ASCII
ebcdic convert ASCII to EBCDIC
ibm slightly different map of ASCII to EBCDIC
lcase map alphabets to lower case
ucase map alphabets to upper case
swab swap every pair of bytes
noerror do not stop processing on an error
sync pad every input record to ibs
.....,..... several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may
end with k, b, or w to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by x to indicate 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, and 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
record of size cbs.

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

EXAMPLE
This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card
images per record into the ASCII file x :

dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase

Note the use of raw magtape. Dd is especially suited to I/O on the raw
physical devices because it allows reading and writing in arbitrary record
sizes.

SEE ALSO
 cp(1).
DIAGNOSTICS

\( f + p \) records \( in(out) \) numbers of full and partial records read(written)

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The \textit{ibm} conversion, while less blessed as a standard, corresponds better 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
delta — make a delta (change) to an SCCS file

SYNOPSIS
delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

DESCRIPTION
Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by get(1) (called the g-file, or generated file).

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

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see admin(1)) that may be present in the SCCS file (see -m and -y keyletters below).

Keyletter arguments apply independently to each named file.

- rSID
Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding gets for editing (get -e) on the same SCCS file were done by the same person (login name). The SID value specified with the -r keyletter can be either the SID specified on the get command line or the SID to be made as reported by the get command (see get(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

- s
Suppresses the issue, on the standard output, of the created delta’s SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

- n
Specifies retention of the edited g-file (normally removed at completion of delta processing).

- glist
Specifies a list (see get(1) for the definition of list) of deltas which are to be ignored when the file is accessed at the change level (SID) created by this delta.

- m[mrlist]
If the SCCS file has the v flag set (see admin(1)) then a Modification Request (MR) number must be supplied as the reason for creating the new delta.

If -m is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the comments? prompt (see -y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.
Note that if the v flag has a value (see admin(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, delta terminates (it is assumed that the MR numbers were not all valid).

-y[comment] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid comment. If -y is not specified and the standard input is a terminal, the prompt comments? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p Causes delta to print (on the standard output) the SCCS file differences before and after the delta is applied in a diff(1) format.

FILES

All files of the form ?-file are explained in the Source Code Control System User's Guide. The naming convention for these files is also described there.

g-file Existed before the execution of delta; removed after completion of delta.
p-file Existed before the execution of delta; may exist after completion of delta.
q-file Created during the execution of delta; removed after completion of delta.
x-file Created during the execution of delta; renamed to SCCS file after completion of delta.
z-file Created during the execution of delta; removed during the execution of delta.
d-file Created during the execution of delta; removed after completion of delta.

/usr/bin/bdiff Program to compute differences between the "gotten" file and the g-file.

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the sees file unless the SOH is escaped. This character has special meaning to SCCS (see sccsfile(5)) and will cause an error.

A get of many SCCS files, followed by a delta of those files, should be avoided when the get generates a large amount of data. Instead, multiple get/delta sequences should be used.

If the standard input (-) is specified on the delta command line, the -m (if necessary) and -y keyletters must also be present. Omission of these keyletters causes an error to occur.

SEE ALSO

admin(1), bdiff(1), get(1), help(1), prs(1), sccsfile(5).


DIAGNOSTICS

Use help(1) for explanations.
NAME
deroff — remove nroff/troff, tbl, and eqn constructs

SYNOPSIS
deroff [ -w ] [ -mx ] [ 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, and between delimiters), and tbl(1) descriptions, and writes the remainder of the file on the standard output. Deroff follows chains of included files (.SO and .MX troff commands); if a file has already been included, a .SO naming that file is ignored and a .MX naming that file terminates execution. If no input file is given, deroff reads the standard input:

The -m option may be followed by an m, s, or l. The resulting -mm or -ms option causes the mm or ms 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. In text, a “word” is any string that contains at least two letters and is composed of letters, digits, ampersands (&), and apostrophes (’); in a macro call, however, a “word” is a string that begins with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from “words.”

SEE ALSO
eqn(1), tbl(1), troff(1).

BUGS
Deroff is not a complete troff interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output. The -ml option does not handle nested lists correctly.
NAME
devnm — device name

SYNOPSIS
/etc/devnm [names]

DESCRIPTION
Devnm identifies the special file associated with the mounted file system
where the argument name resides.
This command is most commonly used by /etc/rc (see rc(8)) to construct a
mount table entry for the root device.

EXAMPLE
The command:
/etc/devnm /usr
produces
rpl /usr
if /usr is mounted on /dev/rpl.

FILES
/dev/rpl
/etc/mnttab

SEE ALSO
setmnt(1M).
NAME
df — report number of free disk blocks

SYNOPSIS
df [ -t ] [ -f ] [ file-systems ]

DESCRIPTION
Df prints out the number of free blocks and free i-nodes available for
online file systems by examining the counts kept in the super-blocks; file-
systems may be specified either by device name (e.g., /dev/rpl) or by
mounted directory name (e.g., /usr). If the file-systems argument is
unspecified, the free space on all of the mounted file systems is printed.

The -t flag causes the total allocated block figures to be reported as well.

If the -f flag is given, only an actual count of the blocks in the free list is
made (free i-nodes are not reported). With this option, df will report on
raw devices.

FILES
/dev/rf*
/dev/rk*
/dev/rp*
/etc/mnttab

SEE ALSO
fsck(1M), fs(5), mnttab(5).
NAME
diff — differential file comparator

SYNOPSIS
diff [ -efbh ] file1 file2

DESCRIPTION
Diff tells what lines must be changed in two files to bring them into agreement. If file1 (file2) is —, the standard input is used. If file1 (file2) is a directory, then a file in that directory with the name file2 (file1) is used. The normal output contains lines of these forms:

nl a n3,n4
nl,n2 d n3
nl,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 nl = 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 >.

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

The -e option produces a script of a, c and d commands for the editor ed, which will recreate file2 from file1. The -f option produces a similar script, not useful with ed, in the opposite order. 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.

Option -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.

FILES
/tmp/d?????
/usr/lib/diffh for -h

SEE ALSO
cmp(1), comm(1), ed(1).

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

BUGS
Editing scripts produced under the -e or -f option are naive about creating lines consisting of a single period (.)

- 1 -
NAME
diff3 — 3-way differential file comparison

SYNOPSIS
diff3 [ --ex3 ] file1 file2 file3

DESCRIPTION
Diff3 compares three versions of a file, and publishes disagreeing ranges of
text flagged with these codes:

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

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

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

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

Under the --ex option, diff3 publishes a script for the editor ed that will
incorporate into file1 all changes between file2 and file3, i.e., the changes
that normally would be flagged ...... and ......3. Option --x (--3) produces a
script to incorporate only changes flagged ...... (--3).

The following command will apply the resulting script to file1.

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

FILES
/tmp/d3*
/usr/lib/diff3prog

SEE ALSO
diff(1).

BUGS
Text lines that consist of a single . will defeat --e.
Files longer than 64K bytes won’t work.
NAME
diffmk — mark differences between files

SYNOPSIS
diffmk name1 name2 name3

DESCRIPTION
Diffmk compares two versions of a file and creates a third file that includes
"change mark" commands for nroff(1) or troff(1). Name1 and name2 are
the old and new versions of the file. Diffmk generates name3, which con­
tains the lines of name2 plus inserted formatter "change mark" (.me) requests. When name3 is formatted, changed or inserted text is shown by |
at the right margin of each line. The position of deleted text is shown by a
single *.

If anyone is so inclined, he can use diffmk to produce listings of C (or
other) programs with changes marked. A typical command line for such
use is:

diffmk old.c new.c tmp; nroff macs tmp | pr

where the file macs contains:
.
.pl 1
.ll 77
.nf
.eo
.nc

The .ll request might specify a different line length, depending on the
nature of the program being printed. The .eo and .nc requests are probably
needed only for C programs.

If the characters | and * are inappropriate, a copy of diffmk can be edited to
change them (diffmk is a shell procedure).

SEE ALSO
diff(1), nroff(1).

BUGS
Aesthetic considerations may dictate manual adjustment of some output.
File differences involving only formatting requests may produce undesirable
output, i.e., replacing .sp by .sp 2 will produce a "change mark" on the
preceding or following line of output.
NAME
dircmp — directory comparison

SYNOPSIS
dircmp 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 in addition to a list that indicates whether the files common to both directories have the same contents.

SEE ALSO
cmp(1), diff(1).
NAME
dpd, odpd, lpd — HONEYWELL sending daemons, line printer daemon

SYNOPSIS
/usr/lib/dpd
/usr/lib/odpd
/usr/lib/lpd

DESCRIPTION
Dpd and odpd are the daemons for the 200-series DATA-PHONE® set and for the Murray Hill Spider network. They are designed to submit jobs to the HONEYWELL 6000 computer via the GRTS interface. For systems with both Spider and DATA-PHONE connections to the MH HONEYWELL 6000 computer, dpd is the Spider daemon, and odpd is the DATA-PHONE set daemon, and is used automatically as a backup when the Spider link is down. On other systems, there is only one daemon, dpd, which uses the DATA-PHONE set. Lpd is the daemon for the line printer.

Dpd and odpd use the directory /usr/spool/dpd. Lpd uses the directory /usr/spool/lpd. The file lock in either directory is used to prevent two daemons from becoming active. After the program has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with “df”. Each such file is submitted as a job. Each line of a job file must begin with a key character to specify what to do with the remainder of the line.

S directs dpd to generate a unique snumb card. The snumb number is generated from the file snumb in the spooling directory in the case of the DATA-PHONE set daemon, or it is read from the PDP-8 that interfaces to GCOS in the case of the Spider daemon. This key character is not used by lpd.

L specifies that the remainder of the line is to be sent as a literal.

I is the same as L, but signals the $ IDENT card which is to be mailed back by the mail option.

B specifies that the rest of the line is a file name. That file is to be sent as binary cards.

F is the same as B except a form-feed is prepended to the file.

U specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.

M is followed by a user ID; after the job is sent, a message is mailed to the user via the mail(1) command to verify the sending of the job.

N is followed by a user file name, to be sent back under the mail option. (Not used by lpd).

Q is followed by a string of characters, which is a message to be sent back to the user under the mail option. (Not used by lpd).

Any error encountered will cause the daemon to drop the call, wait up to 20 minutes, (only 10 seconds for lpd), and start over. This means that an improperly constructed “df” file may cause the same job to be submitted every 20 minutes.

Dpd is automatically initiated by all of the GCOS commands, (dpr, gcat, fget, and fsend) and by /etc/rc. On systems with both dpd daemons, odpd is automatically initiated by dpd on certain errors from Spider. Lpd is automatically initiated by the line printer command, lpr.

To restart dpd or lpd (in the case of hardware or software malfunction), it is necessary to first kill the old daemon (if it is still alive), and remove the lock file (if present), before initiating the new daemon. This is done automatically by /etc/rc when the system is brought up, in case there were
any jobs left in the spooling directory when the system last went down.

FILES
/usr/spool/dpd/* spool area for GCOS daemons.
/usr/spool/lpd/* spool area for line printer daemon.
/etc/passwd to get the user’s name.
/dev/dn? ACU device for use with the DATA-PHONE set.
/dev/lp line printer device.

SEE ALSO
dpr(1C), fget(1C), fget.demon(1C), fsend(1C), gcat(1C), lpr(!).
NAME
dpr - off-line print

SYNOPSIS
dpr [ -destination ] [ options ] [ files ]

DESCRIPTION
Dpr causes the named files to be printed off-line at the specified destination, by GCOS at the Murray Hill Computation Center. GCOS identification must appear in the UNIX password file (see passwd(5)), or be supplied by the -i option. If no files are listed the standard input is assumed; thus dpr may be used as a filter.

The destination is a two-character code which is taken to be a Murray Hill GCOS "station id." Useful codes are rl for quality print, and q1 for quality print with special ribbon, both on regular wide paper. The codes r2 and q2 give the same print on narrow paper. The default destination is on-line at the Murray Hill Computation Center.

The following options, each as a separate argument, and in any combination (multiple outputs are permitted), may be given before or after the destination:

- c Makes a copy of the file to be sent before returning to the user.
- r Removes the file after sending it.
- f Uses the next argument as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when dpr is being used as a filter).
- i Supplies the GCOS "ident card" image as the parameter -i Mxxxx,Myyy where Mxxxx is the GCOS job number and Myyy the GCOS bin number.
- m When transmission is complete, reports by mail(1) the so-called snumb of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is the default option.
- n Does not report the completion of transmission by mail(1).
- sn Submits job to GCOS with service grade n (n=1, 2, 3). Default is -s2.

EXAMPLES
The command:

    dpr -r -n error1 error2

will send the files error1 and error2 to GCOS for printing, removing the files after they have been sent, but not sending mail. The line:

    pr file1 | dpr -s1 -f job1 -r1

will send the output of pr to GCOS for printing on the quality printer with service grade 1, and will send mail that job1 has been sent.

FILES
/etc/passwd user's identification and GCOS ident card.
/usr/lib/dpd sending daemon.
/usr/spool/dpd/* spool area.

SEE ALSO
dpd(1C), fget(1C), fsend(1C), gcat(1C).
NAME
du — summarize disk usage

SYNOPSIS
du [ -ars ] [ names ]

DESCRIPTION
Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the names argument. The block count includes the indirect blocks of the file. If names is missing, . is used.

The optional argument -s causes only the grand total (for each of the specified names) to be given. The optional argument -a causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. The -r option will cause du to generate messages in such instances.

A file with two or more links is only counted once.

BUGS
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 count the excess files more than once.
Files with holes in them will get an incorrect block count.
NAME
dump — incremental file system dump

SYNOPSIS
dump [ key [ arguments ] file-system ]

DESCRIPTION
Dump copies to magnetic tape all files changed after a certain date in the file-system. The key specifies the date and other options about the dump. Key consists of characters from the set 0123456789fuusd.

f Place the dump on the next argument file instead of the tape.

u If the dump completes successfully, write the date of the beginning of the dump on file /etc/ddate. This file records a separate date for each file system and each dump level.

0—9 This number is the "dump level". All files modified since the last date stored in the file /etc/ddate for the same file system at lesser levels will be dumped. If no date is determined by the level, the beginning of time is assumed; thus the option 0 causes the entire file system to be dumped.

s The size of the dump tape is specified in feet. The number of feet is taken from the next argument. When the specified size is reached, the dump will wait for reels to be changed. The default size is 2,300 feet.

d The density of the tape, expressed in BPI, is taken from the next argument. This is used in calculating the amount of tape used per write. The default is 1600.

If no arguments are given, the key is assumed to be 9u and a default file system is dumped to the default tape.

Now a short suggestion on how to perform dumps. Start with a full level-0 dump: dump 0u. Next, periodic level-9 dumps should be made on an exponential progression of tapes. (Sometimes called Tower of Hanoi: 1, 2, 1, 3, 1, 2, 1, 4, ...; tape 1 used every other time, tape 2 is used every fourth, tape 3 is used every eighth, etc.): dump 9u. When the level-9 incremental approaches a full tape (about 78,000 blocks at 1600 BPI blocked 20 blocks per record), a level-1 dump should be made: dump 1u. After this, the exponential series should progress as if uninterrupted. These level-9 dumps are based on the level-1 dump, which is based on the level-0 full dump. This progression of levels of dumps can be carried as far as desired.

FILES
default file system and tape vary with installation.
/etc/ddate: record dump dates of file system/level.

SEE ALSO
cpio(1), restor(1M), volcopy(1M), dump(5).

DIAGNOSTICS
If the dump requires more than one tape, it will ask you to change tapes. Reply with a new-line after this has been done.

BUGS
Sizes arc based on 1600 BPI blocked tape. The raw magnetic tape device has to be used to approach these densities. Read errors on the file system are ignored. Write errors on the magnetic tape are usually fatal.
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 \
\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.

Echo is useful for producing diagnostics in command files and for sending
known data into a pipe.

SEE ALSO
sh(1).
NAME
ed — text editor

SYNOPSIS
ed [ - ] [ -x ] [ file ]

DESCRIPTION
Ed is the standard text editor. If the file argument is given, ed simulates an e command (see below) on the named file; that is to say, the file is read into ed's buffer so that it can be edited. The optional - suppresses the printing of character counts by e, r, and w commands, of diagnostics from e and q commands, and of the ! prompt after a !shell command. If -x is present, an x command is simulated first to handle an encrypted file. Ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a w (write) command is given. The copy of the text being edited resides in a temporary file called the buffer. There is only one buffer.

Commands to ed have a simple and regular structure: zero, one, or two addresses followed by a single-character command, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While ed is accepting text, it is said to be in input mode. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports 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 REs match a single character:

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 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).
   c. $ (currency symbol), which is special at the end of an entire RE (see 3.2 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 in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character except new-line and the remaining characters in the string. The ^ has this special meaning only if it occurs first in the string. The minus (−) may be used to indicate a range of consecutive ASCII characters; for example, [0−9J] is equivalent to [0123456789]. The − loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (J) 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 (J) 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.

The following rules may be used to construct REs 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 ^\(.\wedge\)\1$ matches a line consisting of two repeated appearances of the same string.

Finally, an entire RE may be constrained to match only an initial segment or final segment of a line (or both):

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an initial segment of a line.

3.2 A currency symbol ($) at the end of an entire RE constrains that RE to match a final segment of a line. The construction ^entire RES$ 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 $n-th line of the buffer.
4. 'x addresses the line marked with the mark name character x, which must be a lower-case letter. Lines are marked with the k command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching forward from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before FILES below.
6. A RE enclosed in question marks (?) addresses the first line found by searching backward from the line preceding the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before FILES below.
7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g., -5 is understood to mean -.5.
9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -. ) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair 1,$, while a semicolon (;) stands for the pair ..$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6.
above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of ed commands, the default addresses are shown in parentheses. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except e, f, r, or w) may be suffixed by p or by l, in which case the current line is either printed or listed, respectively, as discussed below under the p and l commands.

\( .a \)<text>

The append command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer.

\( .c \)<text>

The change command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

\( .d \)<text>

The delete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

\( e \) file

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the f command). The number of characters read is typed; file is remembered for possible use as a default file name in subsequent e, r, and w commands. If file begins with !, the rest of the line is taken to be a shell (sh(1)) command whose output is to be read. Such a shell command is not remembered as the current file name. See also DIAGNOSTICS below.

\( E \) file

The Edit command is like e, except that the editor does not check to see if any changes have been made to the buffer since the last w command.

\( f \) file

If file is given, the file-name command changes the currently-remembered file name to file; otherwise, it prints the currently-remembered file name.

\( (1, S)g/RE/command list \)

In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given command list is executed with . initially set to that line. A single command or the first of a list of commands appears
on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; \textit{a}, \textit{i}, and \textit{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 \textit{p} command. The \textit{g}, \textit{G}, \textit{v}, and \textit{V} commands are not permitted in the command list. See also \textit{BUGS} and the last paragraph before \textit{FILES} below.

\textbf{(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, \textit{.} is changed to that line, and any one command (other than one of the \textit{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 \textit{G}. Note that the commands input as part of the execution of the \textit{G} command may address and affect any lines in the buffer. The \textit{G} command can be terminated by an interrupt signal (ASCII \texttt{DEL} or \texttt{BREAK}).

\textbf{h}

The \texttt{help} command gives a short error message that explains the reason for the most recent \texttt{?} diagnostic.

\textbf{H}

The \texttt{Help} command causes \textit{ed} to enter a mode in which error messages are printed for all subsequent \texttt{?} diagnostics. It will also explain the previous \texttt{?} if there was one. The \texttt{H} command alternately turns this mode on and off; it is initially off.

\textbf{(,)i}

\texttt{<text>}

\texttt{.}

The \textit{insert} command inserts the given text before the addressed line; \texttt{.} is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the \textit{a} command only in the placement of the input text. Address 0 is not legal for this command.

\textbf{(. ,+1)j}

The \textit{join} command joins contiguous lines by removing the appropriate new-line characters. If only one address is given, this command does nothing.

\textbf{(,)kx}

The \textit{mark} command marks the addressed line with name \textit{x}, which must be a lower-case letter. The address 'x then addresses this line; \texttt{.} is unchanged.

\textbf{(.)l}

The \textit{list} command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., \texttt{tab}, backspace) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An \texttt{l} command may be appended to any other command other than \texttt{e, f, r, or w}.

\textbf{(. ,)ma}

The \textit{move} command repositions the addressed line(s) after
the line addressed by \textit{a}. Address \texttt{0} is legal for \textit{a} and causes
the addressed line(s) to be moved to the beginning of the file; it is an error if address \textit{a} falls within the range of moved
lines; \texttt{.} is left at the last line moved.

\texttt{(...n)}

The \texttt{number} command prints the addressed lines, preceding
each line by its line number and a tab character; \texttt{.} is left at
the last line printed. The \texttt{n} command may be appended to
any other command other than \texttt{e, f, r,} or \texttt{w}.

\texttt{(...p)}

The \texttt{print} command prints the addressed lines; \texttt{.} is left at the
last line printed. The \texttt{p} command may be appended to
any other command other than \texttt{e, f, r,} or \texttt{w}; for example, \texttt{dp}
deletes the current line and prints the new current line.

\texttt{P}

The editor will prompt with a * for all subsequent commands.
The \texttt{P} command alternately turns this mode on and off; it is
initially off.

\texttt{q}

The \texttt{quit} command causes \texttt{ed} to exit. No automatic write of a
file is done (but see \texttt{DIAGNOSTICS} below).

\texttt{Q}

The editor exits without checking if changes have been made
in the buffer since the last \texttt{w} command.

\texttt{(S)r file}

The \texttt{read} command reads in the given file after the addressed
line. If no file name is given, the currently-remembered file
name, if any, is used (see \texttt{e} and \texttt{f} commands). The
currently-remembered file name is not changed unless \texttt{file} is
the very first file name mentioned since \texttt{ed} was invoked.
Address \texttt{0} is legal for \texttt{r} and causes the file to be read at the
beginning of the buffer. If the read is successful, the number
of characters read is typed; \texttt{.} is set to the last line read in. If
\texttt{file} begins with \texttt{!}, the rest of the line is taken to be a shell
\texttt{(sh(1))} command whose output is to be read. Such a shell
command is not remembered as the current file name.

\texttt{(...s)/RE/replacement/} or
\texttt{(...s)/RE/replacement/g}

The \texttt{substitute} command searches each addressed line for an
occurrence of the specified \texttt{RE}. In each line in which a match
is found, all (non-overlapped) matched strings are replaced by
the \texttt{replacement} if the global replacement indicator \texttt{g}
appears after the command. If the global indicator does not
appear, only the first occurrence of the matched string is
replaced. It is an error for the substitution to fail on all
addressed lines. Any character other than space or new-line
may be used instead of \texttt{/} to delimit the \texttt{RE} and the \texttt{repla-
cement}; \texttt{.} is left at the last line on which a substitution
occurred. See also the last paragraph before \texttt{FILES} below.

An ampersand (\texttt{&}) appearing in the \texttt{replacement} is replaced
by the string matching the \texttt{RE} on the current line. The
special meaning of \texttt{&} in this context may be suppressed by
preceding it by \texttt{\textbackslash}. As a more general feature, the characters
where \( n \) is a digit, are replaced by the text matched by the \( n \)-th regular subexpression of the specified RE enclosed between \( \backslash( \) and \( \backslash) \). When nested parenthesized subexpressions are present, \( n \) is determined by counting occurrences of \( \backslash( \) 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 \( \backslash \).

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 \( \backslash \). Such substitution cannot be done as part of a \( g \) or \( v \) command list.

\((..)ta\) 
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); \( \cdot \) is left at the last line of the copy.

\( u \) 
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.

\((1,5)v/RE/command list\) 
This command is the same as the global command \( g \) except that the \( command list \) is executed with \( . \) initially set to every line that does not match the RE.

\((1,5)V/RE/\) 
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.

\((1,5)w/file\) 
The write command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your \texttt{umask} setting (see \texttt{sh(1)}) dictates otherwise. The currently-remembered file name is \textit{not} changed unless \texttt{file} is the very first file name mentioned since \texttt{ed} was invoked. If no file name is given, the currently-remembered file name, if any, is used (see \texttt{e} and \texttt{f} commands); \( . \) is unchanged. If the command is successful, the number of characters written is typed.

If \texttt{file} begins with \( ! \), the rest of the line is taken to be a shell (\texttt{sh(1)}) command whose output is to be read. Such a shell command is \textit{not} remembered as the current file name.

\( X \) 
A key string is demanded from the standard input. Subsequent \( e, r, \) and \( w \) commands will encrypt and decrypt the text with this key by the algorithm of \texttt{crypt(1)}. An explicitly empty key turns off encryption.

\((5)=\) 
The line number of the addressed line is typed; \( . \) is unchanged by this command.

\(!shell command\) 
The remainder of the line after the \( ! \) is sent to the UNIX shell
(sh(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

\((.+1)\)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to \(.+1p\); it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, ed prints a ? and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, ed discards ASCII NUL characters and all characters after the last new-line. Files (e.g., a.out) that contain characters not in the ASCII set (bit 8 on) cannot be edited by ed.

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:

\[ s/s1/s2 \]
\[ g/s1 \]
\[ ?s1 \]
\[ s/sl/s2 \]
\[ g/sl \]
\[ ?sl \]

FILES

/tmp/e# temporary; # is the process number.
ed.hup work is saved here if the terminal is hung up.

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 -command-line option inhibits this feature.

SEE ALSO

crypt(1), grep(1), sed(1), sh(1).
A Tutorial Introduction to the UNIX Text Editor by B. W. Kernighan.
Advanced Editing on UNIX by B. W. Kernighan.

CAVEATS AND BUGS

A ! command cannot be subject to a g or a v command.
The / command and the ! escape from the e, r, and w commands cannot be used if the the editor is invoked from a restricted shell (see sh(1)).
The sequence \m in a RE does not match any character.
The l command mishandles DEL.
Files encrypted directly with the crypt(1) command with the null key cannot be edited.
Because 0 is an illegal address for the w command, it is not possible to create an empty file with ed.
NAME
efl — Extended Fortran Language

SYNOPSIS
efl [ options ] [ files ]

DESCRIPTION
Efl compiles a program written in the EFL language into clean Fortran on the standard output. Efl provides the C-like control constructs of ratfor(1):

- statement grouping with braces.
- decision-making:
  - if, if-else, and select-case (also known as switch-case);
  - while, for, Fortran do, repeat, and repeat ... until loops;
  - multi-level break and next.

EFL has C-like data structures, e.g.:

```
struct {
  integer flags(3)
  character(8) name
  long real coords(2)
} table(100)
```

The language offers generic functions, assignment operators (+=, &=, etc.), and sequentially evaluated logical operators (&& and ||). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

- free-form input:
  - multiple statements per line; automatic continuation; statement label names (not just numbers).
- comments:
  - `#` this is a comment.
- translation of relational and logical operators:
  - `>`, `>=`, `&`, etc., become `.GT.`, `.GE.`, `.AND.`, etc.
- return expression to caller from function:
  - `return (expression)`
- defines:
  - `define name replacement`
- includes:
  - `include file`

Efl understands several option arguments: `-w` suppresses warning messages, `-#` suppresses comments in the generated program, and the default option `-C` causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an EFL option as if it had appeared in an option statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target machine may be selected by one of the choices: `system=unix`, `system=gcos`, or `system=cray`. The default setting of the system option is the same as the machine the compiler is running on. Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.
EFL is best used with f77(1).

SEE ALSO
cc(1), f77(1), ratfor(1).

*The Programming Language EFL* by S.I. Feldman.
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.

SEE ALSO
sh(1), exec(2), profile(5), environ(7).
NAME
eqn, neqn, checkeq — format mathematical text for nroff or troff

SYNOPSIS
eqn [ -dxy ] [ -pn ] [ -sn ] [ -fn ] [ files ]
neqn [ -dxy ] [ -pn ] [ -sn ] [ -fn ] [ files ]
checkeq [ files ]

DESCRIPTION
Eqn is a troff(1) preprocessor for typesetting mathematical text on a Wang Laboratories, Inc. C/A/T phototypeetter, while neqn is used for the same purpose with nroff(1) on typewriter-like terminals. Usage is almost always:

eqn files | troff
neqn files | nroff

or equivalent.

If no files are specified, these programs read from the standard input. A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a line beginning with .EN. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as delimiters; subsequent text between delimiters is then treated as eqn input. Delimiters may be set to characters x and y with the command-line argument -dxy or (more commonly) with delim xy between .EQ and .EN. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by delim off. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program checkeq reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within eqn are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character such as x could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords sub and sup. Thus x sub j makes x_j, a sub k sup 2 produces a_k^2, while e^{z^2+y^2} is made with e sup {x sup 2 + y sup 2}. Fractions are made with over: a over b yields \( \frac{a}{b} \); sqrt makes square roots: \( \sqrt{ax^2 + bx + c} \) results in \( \frac{1}{\sqrt{ax^2 + bx + c}} \).

The keywords from and to introduce lower and upper limits: \( \lim_{n \to \infty} \sum_{i=0}^{n} x_i \) is made with \( \lim_{n \to \infty} \{ n \to \inf \} \) \( \sum_{0}^{n} x \) sub i. Left and right brackets, braces, etc., of the right height are made with left and right:

left [ x sup 2 + y sup 2 over alpha right ] \( \approx \approx \) 1 produces \( \left[ \frac{x^2 + y^2}{\alpha} \right] = 1 \).

Legal characters after left and right are braces, brackets, bars, c and f for ceiling and floor, and ** for nothing at all (useful for a right-side-only bracket). A left thing need not have a matching right thing.
Vertical piles of things are made with \texttt{pile}, \texttt{lpile}, \texttt{epile}, and \texttt{rpile}: \quad \texttt{pile \{a above b above c\}} produces \texttt{b}. Piles may have arbitrary numbers of \texttt{c} elements; \texttt{lpile} left-justifies, \texttt{pile} and \texttt{epile} center (but with different vertical spacing), and \texttt{rpile} right-justifies. Matrices are made with \texttt{matrix}: \quad \texttt{matrix \{ lcol \{ x sub i above y sub 2 \} ccol \{ 1 above 2 \}\} produces \quad x_i \quad 1 \quad y_{22}.}

In addition, there is \texttt{reol} for a right-justified column.

Diacritical marks are made with \texttt{dot}, \texttt{dotdot}, \texttt{hat}, \texttt{tilde}, \texttt{bar}, \texttt{vec}, \texttt{dyad}, and \texttt{under}: \quad \texttt{x dot} = f(t) \texttt{bar} is \texttt{\ddot{x} = f'(t)}, \quad \texttt{y dotdot bar} \texttt{\sim} = \texttt{n under} \texttt{is \dddot{y} = a}, \texttt{and x vec} \texttt{\sim} = \texttt{\sim y dyad is \vec{x} = \vec{y}.}

Point sizes and fonts can be changed with \texttt{size \pm n}, \texttt{roman}, \texttt{italic}, \texttt{bold}, and \texttt{font n}. Point sizes and fonts can be changed globally in a document by \texttt{gsize n} and \texttt{gfont n}, or by the command-line arguments \texttt{-sn} and \texttt{-fn}.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument \texttt{-pn}.

Successive display arguments can be lined up. Place \texttt{mark} before the desired lineup point in the first equation; place \texttt{lineup} at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with \texttt{define}: \quad \texttt{define thing \% replacement \%}

defines a new token called \texttt{thing} that will be replaced by \texttt{replacement} whenever it appears thereafter. The \% may be any character that does not occur in \texttt{replacement}.

Keywords such as \texttt{sum (\Sigma)}, \texttt{int (\int)}, \texttt{inf (\infty)}, and shorthands such as \texttt{>= (\geq), != (\neq), and -> (\rightarrow)} are recognized. Greek letters are spelled out in the desired case, as in \texttt{alpha (\alpha)}, or \texttt{GAMMA (\Gamma)}. Mathematical words such as \texttt{sin}, \texttt{cos}, and \texttt{log} are made Roman automatically. \texttt{Troff(1)} four-character escapes such as \texttt{\(dd (\dagger)} and \texttt{\(bs (\oplus)} may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with \texttt{troff(1)} when all else fails. Full details are given in the manual cited below.

\texttt{SEE ALSO}


\texttt{New Graphic Symbols for EQN and NEQN} by C. Scrocca.

\texttt{mm(1), mmt(1), tbl(1), troff (1), eqnchar(7), mm(7), mv(7)}.

\texttt{BUGS}

To embolden digits, parentheses, etc., it is necessary to quote them, as in \texttt{bold "12.3"}.

See also \texttt{BUGS} under \texttt{troff(1)}.
NAME
errdead — extract error records from dump

SYNOPSIS
/etc/errdead dumpfile [ namelist ]

DESCRIPTION
When hardware errors are detected by the system, an error record that contains information pertinent to the error is generated. If the error-logging daemon errdemon(1M) is not active or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. Errdead examines a system dump (or memory), extracts such error records, and passes them to errpt(1M) for analysis.

The dumpfile specifies the file (or memory) that is to be examined. The system namelist is specified by namelist; if not given, /unix is used.

FILES
/unix system namelist
/usr/bin/errpt analysis program
/usr/tmp/errXXXXXX temporary file

DIAGNOSTICS
Diagnostics may come from either errdead or errpt. In either case, they are intended to be self-explanatory.

SEE ALSO
errdemon(1M), errpt(1M).
NAME
errdemon — error-logging daemon

SYNOPSIS
/etc/errdemon [ file ]

DESCRIPTION
The error logging daemon *errdemon* collects error records from the operating system by reading the special file /dev/error and places them in file. If file is not specified when the daemon is activated, /usr/adm/errfile is used. Note that file is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon*; that responsibility is left to *errpt(1M)*. The error-logging daemon is terminated by sending it a software kill signal (see *signal(2)*). Only the super-user may start the daemon, and only one daemon may be active at any time.

FILES
/dev/error source of error records
/usr/adm/errfile repository for error records

DIAGNOSTICS
The diagnostics produced by *errdemon* are intended to be self-explanatory.

SEE ALSO
errpt(1M), errstop(1M), kill(1), err(4).
NAME

errpt - process a report of logged errors

SYNOPSIS

errpt [-a] [-dev]... [-int] [-mem] [-s date] [-e date] [-pn] [-f] [files]

DESCRIPTION

Errpt processes data collected by the error logging mechanism (errdemon(1M)) and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, errpt attempts to use /usr/adm/errfile as file.

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unable to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times that errpt has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped, and any time changes (via date(1)) that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report may be limited to certain records in the following ways:

-s date Ignore all records posted earlier than date, where date has the form mmddhhmmyy, consistent in meaning with the date(1) command.
-e date Ignore all records posted later than date, whose form is as described above.
-a Produce a detailed report that includes all error types.
-dev A detailed report is limited to dev, a block device identifier. Errpt is familiar with the common form of identifiers (e.g., rs03, R.S04; see Section 4 of this volume). Currently, the block devices for which errors are logged are RP03, RP04, RP05, RP06, RS03, RS04, TU10, TU16, RK05, and RF11.
-int Include in a detailed report errors of the stray-interrupt type.
-mem Include in a detailed report errors of the memory-parity type.
-p n Limit the size of a detailed report to n pages.
-f In a detailed report, limit the reporting of block device errors to unrecovered errors.

FILES

/usr/adm/errfile default error file

SEE ALSO

errdemon(1M), errfile(5).
NAME
errstop — terminate the error-logging daemon

SYNOPSIS
/etc/errstop [ namelist ]

DESCRIPTION
The error-logging daemon errdemon(1M) is terminated by using errstop. This is accomplished by executing ps(1) to determine the daemon's identity and then sending it a software kill signal (see signal(2)); /unix is used as the system namelist if none is specified. Only the super-user may use errstop.

FILES
/unix default system namelist

DIAGNOSTICS
The diagnostics produced by errstop are intended to be self-explanatory.

SEE ALSO
errdemon(1M), ps(1), kill(2).
NAME
expr — evaluate arguments as an expression

SYNOPSIS
expr arguments

DESCRIPTION
The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that 0 is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| expr
returns the first expr if it is neither null nor 0, otherwise returns the second expr.

expr \\& expr
returns the first expr if neither expr is null or 0, otherwise returns 0.

expr \{ =, \>, \>=, \<, \<=, != \} expr
returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

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 characters matched (0 on failure). Alternatively, the \(. . . \) pattern symbols can be used to return a portion of the first argument.

EXAMPLES
1. a="expr $a + 1"
   adds 1 to the shell variable a.

2. if "For $a equal to either "/usr/abc/file" or just "file"
   expr $a : ".*/\(.\*)" \| $a
returns the last segment of a path name (i.e., file). Watch out for / alone as an argument: expr will take it as the division operator (see BUGS below).
3.  \$ A better representation of example 2.
    \$expr  //\$a : .*/(.*)

    The addition of the // characters eliminates any ambiguity
    about the division operator and simplifies the whole expres
    sion.

4.  \$expr  $\$VAR : .*\$ returns the number of characters in $\$VAR.

SEE ALSO
    ed(1), sh(1).

EXIT CODE
    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.

DIAGNOSTICS
    syntax error  for operator/operand errors
    non-numeric argument  if arithmetic is attempted on such a string

BUGS
    After argument processing by the shell, \$expr cannot tell the difference
    between an operator and an operand except by the value.  If $a is an =,
    the command:
      \$expr  $a = -
    looks like:
      \$expr  - -

    as the arguments are passed to \$expr (and they will all be taken as the =
    operator).  The following works:
      \$expr  X$a = X=
NAME
f77 — Fortran 77 compiler

SYNOPSIS
f77 [ options ] files

DESCRIPTION
F77 is the UNIX Fortran 77 compiler; it accepts several types of files arguments:

- Arguments whose names end with .f are taken to be Fortran 77 source programs; they are compiled and each object program is left in the current directory in a file whose name is that of the source, with .o substituted for .f.
- Arguments whose names end with .r or .e are taken to be RATFOR or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by f77, producing .o files.
- In the same way, arguments whose names end with .c or .s are taken to be C or assembly source programs and are compiled or assembled, producing .o files.

The following options have the same meaning as in cc(1) (see ld(1) for link editor options):

- c Suppress link editing and produce .o files for each source file.
- p Prepare object files for profiling (see prof(1)).
- O Invoke an object-code optimizer.
- S Compile the named programs and leave the assembler-language output in corresponding files whose names are suffixed with .s. (No .o files are created.)
- ooutput Name the final output file output, instead of a.out.
- f In systems without floating-point hardware, use a version of f77 that handles floating-point constants and links the object program with the floating-point interpreter.

The following options are peculiar to f77:

- onetrip Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)
- u Make the default type of a variable "undefined", rather than using the default Fortran rules.
- w Suppress all warning messages. If the option is -w66, only Fortran 66 compatibility warnings are suppressed.
- F Apply EFL and RATFOR preprocessor to relevant files, put the result in files whose names have their suffix changed to .of. (No .o files are created.)
- m Apply the M4 preprocessor to each EFL or RATFOR source file before transforming with the ratfor(1) or efl(1) processors.
- E The remaining characters in the argument are used as an EFL flag argument whenever processing a .e file.
- R The remaining characters in the argument are used as a RATFOR flag argument whenever processing a .r file.

Other arguments are taken to be either link-editor option arguments or f77-compilable object programs (typically produced by an earlier run), or libraries of f77-compilable routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the default name a.out.
FILES
file.[fresc]   input file
file.o         object file
a.out          linked output
./fort[pid].?  temporary
/usr/lib/f77pass1 compiler
/lib/c1         pass 2
/lib/c2       optional optimizer
/usr/lib/libF77.a intrinsic function library
/usr/lib/libI77.a Fortran I/O library
/lib/libc.a   C library; see Section 3 of this Manual.

SEE ALSO
A Portable Fortran 77 Compiler by S. I. Feldman and P. J. Weinberger
cc(1), efl(1), ld(1), m4(1), prof(1), ratfor(1).

DIAGNOSTICS
The diagnostics produced by f77 itself are intended to be self-explanatory.
Occasional messages may be produced by the link editor ld(1).
NAME
factor, primes — factor a number, generate large primes

SYNOPSIS
factor [ number ]
primes

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 $2^{30}$ (about $7.2 \times 10^{16}$) 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. It takes 1 minute to factor a prime near $10^{14}$ on a PDP-11.

When primes is invoked, it waits for a number to be typed in. If you type in a positive number less than $2^{30}$ it will print all primes greater than or equal to this number.

DIAGNOSTICS
"Ouch" for input out of range or for garbage input.
NAME
fget — retrieve files from the HONEYWELL 6000

SYNOPSIS
fget [ options ] [ files ]

DESCRIPTION
fget arranges to have one or more GCOS files sent to UNIX. GCOS identification must appear in the UNIX password file (see passwd(5)), or be supplied by the -i option. Normally, the files retrieved will appear in the UNIX user's current directory under the GCOS file name.

The GCOS catalog from which the files are obtained depends on the form of the file name argument. If the file name has only embedded slashes, then it is assumed to be a full GCOS path name and that file is retrieved. If the file name has no embedded slashes or begins with a slash, then the GCOS catalog from which the file is retrieved is the same as the UNIX login name of the person who issues the command. If, however, a user has a different name in the third field of the GCOS "ident card image" (which image is extracted from the UNIX password file — see passwd(5)), this name is taken as the GCOS catalog name. Whatever GCOS catalog is finally used, the files must either have general read permission or the user must have arranged that the user ID network has read permission on that catalog (see fsend(1C)). This can be accomplished with the GCOS command:

    filesys mc <user ID>,(r)/network/

The UNIX file into which the retrieved GCOS file will ultimately be written is initialized with one line containing the complete GCOS file name. If the file contains the initial line for an extended period, it means that GCOS is down or something has gone horribly wrong and you should try again.

The following options, each as a separate argument (or in the case of -d and -u, as two separate arguments), may appear in any order, but must precede all file arguments.

- a  Retrieve files as ASCII (default).
- b  Retrieve files as binary.
- d  Use the next argument as the UNIX directory into which retrieved files are written.
- i  Supply the GCOS "ident card" image as the parameter -iMxxxx,Myyy where Mxxxx is the GCOS job number and Myyy the GCOS bin number.
- m  When the request has been forwarded to GCOS, report by mail(1) the so-called snumb of the receiving job; mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran or that UNIX retrieved the output. This is the default option.
- n  Do not report the forwarding of the request by mail(1).
- o  Print the on-line GCOS accounting output.
- t  Toss out the on-line GCOS accounting output. This is the default option.
- sn Submit job to GCOS with service grade n (n=1, 2, 3). Default is -s1.
- u  Use the next argument as the GCOS catalog name for all files.

EXAMPLES
The command:

    fget -u gcosme -t -n -d /usr/me/test file1 file2

will retrieve the GCOS files gcosme/file1 and gcosme/file2, as the UNIX files /usr/me/test/file1 and /usr/me/test/file2, respectively, but will not
generate any mail or GCOS accounting output as a result of the transaction.

FILES
/etc/passwd user's identification and GCOS ident card.
/usr/lib/dpd sending daemon.
/usr/spool/dpd/* spool area.
/usr/lib/fget.demon retrieval daemon.

SEE ALSO
dpd(1C), dpr(1C), fsend(1C), fget.demon(1C), passwd(5).
NAME
fget.demon, fget.odemon — file retrieval daemons

SYNOPSIS
/usr/lib/fget.demon time
/usr/lib/fget.odemon time

DESCRIPTION
Fget.demon and fget.odemon are the retrieval daemons for the 200-series
DATA-PHONE® set and for the Murray Hill Spider network. They are
designed to retrieve files that have been requested by fget(1C) from the MH
HONEYWELL 6000 computer. The argument time is the number of seconds
for fget.demon to wait for files to appear from GRTS. The default is 6 minu-
tes. Fget.demon is automatically initiated by fget(1C), and by cron(1M).

On systems with both Spider and DATA-PHONE connections to the
HONEYWELL 6000 computer, fget.demon uses Spider, and fget.odemon uses
the DATA-PHONE set, and is called automatically as a backup when the Spi-
der connection is down. On other systems, there is only one fget daemon,
fget.demon, which use the DATA-PHONE set.

The fget daemons use the spooling directory /usr/spool/dpd. The file
glock in that directory is used to prevent two daemons from becoming
active. After the program has successfully set the lock, it forks and the
main path exits, thus spawning the daemon. GRTS is interrogated for any
output for the daemon’s station-id. If none, fget.demon will wait up to time
seconds, interrogating GRTS every minute or so to see if any output has
arrived. All problems and successful transactions are recorded in the errors
file in the spooling directory.

To restart fget.demon (in the case of hardware or software malfunction), it
is necessary to first kill the old fget.demon (if still alive), and remove the
lock file (if present), before initiating fget.demon. This is done automati-
cally by /etc/re when the system is brought up, in case there are any files
waiting to come over.

FILES
/usr/spool/dpd/* spool area.
/dev/dn? ACU device.

SEE ALSO
dpd(1C), fget(1C).
NAME
file — determine file type

SYNOPSIS
file [-f] file ...

DESCRIPTION
File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, file examines the first 512 bytes and tries to guess its language. If an argument is an executable a.out, file will print the version stamp, provided it is greater than 0 (see the description of the -V option in ld(1)).

If the -f option is given, the next argument is taken to be a file containing the names of the files to be examined.
FIND(1)

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 (i.e., one or more path names) seeking files that match a
boolean expression written in the primaries given below. In the descrip-
tions, the argument \( n \) is used as a decimal integer where \(+n\) means more
than \( n \), \(-n\) means less than \( n \) and \( n \) means exactly \( n \).

- name file True if file matches the current file name. Normal shell
  argument syntax may be used if escaped (watch out for [, ?, and *).

- perm onum True if the file permission flags exactly match the octal
  number onum (see chmod(1)). If onum is prefixed by a
  minus sign, more flag bits (017777, see stat(2)) become
  significant and the flags are compared:
  
  \((flags\&onum)==onum\)

- type c True if the type of the file is c, where c is b, c, d, p, or f
  for block special file, character special file, directory, fifo
  (a.k.a named pipe), or plain file.

- links n True if the file has \( n \) links.

- user uname True if the file belongs to the user uname. If uname is
  numeric and does not appear as a login name in the
  /etc/passwd file, it is taken as a user ID.

- group gname True if the file belongs to the group gname. If gname is
  numeric and does not appear in the /etc/group file, it is
  taken as a group ID.

- size n True if the file is \( n \) blocks long (512 bytes per block).

- atime n True if the file has been accessed in \( n \) days.

-mtime n True if the file has been modified in \( n \) days.

- ctime n True if the file has been changed in \( n \) days.

- exec cmd True if the executed cmd returns a zero value as exit sta-
tus. The end of cmd must be punctuated by an escaped
semicolon. A command argument {} is replaced by the
current path name.

- ok cmd Like -exec except that the generated command line is
  printed with a question mark first, and is executed only if
  the user responds by typing y.

- print Always true; causes the current path name to be printed.

- cpio device Write the current file on device in cpio(5) format (5120
  byte records).

- newer file True if the current file has been modified more recently
  than the argument file.

( expression ) True if the parenthesized expression is true (parentheses
are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of
decreasing precedence):

-
1) The negation of a primary (\( \neg \) is the unary \textit{not} operator).

2) Concatenation of primaries (the \textit{and} operation is implied by the juxta-position of two primaries).

3) Alternation of primaries (\( \lor \) is the \textit{or} operator).

**EXAMPLE**

To remove all files named \texttt{a.out} or \texttt{.o} that have not been accessed for a week:

```
find / \( -name a.out -o -name '.o' \) -atime +7 -exec rm {} 
```

**FILES**

```
/\etc/passwd, /\etc/group
```

**SEE ALSO**

\texttt{cpio(1), sh(1), test(1), stat(2), cpio(5), fs(5)}. 

---
NAME

fsck — file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [ -y ] [ -n ] [ -sX ] [ -SX ] [ -t file ] [ file-system ]

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for UNIX file systems. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond yes or no. If the operator does not have write permission fsck will default to a -n action.

Fsck has more consistency checks than its predecessors check, dcheck, fcheck, and icheck combined.

The following flags are interpreted by fsck:

- y Assume a yes response to all questions asked by fsck.
- n Assume a no response to all questions asked by fsck; do not open the file system for writing.
- sX Ignore the actual free list and (unconditionnally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The -sX option allows for creating an optimal free-list organization. The following forms of X are supported for the following devices:

- s3 (RP03)
- s4 (RP04, RP05, RP06)
- sBlocks-per-cylinder:Blocks-to-skip (for anything else)

If X is not given, the values used when the file system was created are used. If these values were not specified, then the value 400:9 is used.

- SX Conditionally reconstruct the free list. This option is like -sX above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using -S will force a no response to all questions asked by fsck. This option is useful for forcing free list reorganization on uncontaminated file systems.

- t If fsck cannot obtain enough memory to keep its tables, it uses a scratch file. If the -t option is specified, the file named in the next argument is used as the scratch file, if needed. Without the -t flag, fsck will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when fsck completes.

If no file-systems are specified, fsck will read a list of default file systems from the file /etc/checklist.
Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   - Incorrect number of blocks.
   - Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   - File pointing to unallocated inode.
   - Inode number out of range.
8. Super Block checks:
   - More than 65536 inodes.
   - More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator’s concurrence, reconnected by placing them in the lost+found directory. The name assigned is the inode number. The only restriction is that the directory lost+found must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making lost+found, copying a number of files to the directory, and then removing them (before fsck is executed).

Checking the raw device is almost always faster.

FILES

/etc/checklist contains default list of file systems to check.

DIAGNOSTICS

The diagnostics produced by fsck are intended to be self-explanatory.

SEE ALSO

checklist(5), fs(5), crash(8).

BUGS

Inode numbers for . and .. in each directory should be checked for validity.

-g and -b options from check should be available in fsck.
NAME
fscv — convert files between PDP-11 and VAX-11/780 systems

SYNOPSIS
/etc/fscv -v ispecial [ ospecial ]
/etc/fscv -p ispecial [ ospecial ]

DESCRIPTION
Fscv converts file systems between PDP-11 and VAX-11/780 formats. The super block, free list, and inodes are converted to the format of the output file. Fscv may be executed on PDP-11 and VAX processors. The mandatory flag specifies the format of the converted file system:

- v Convert file system from PDP-11 to VAX format.
- p Convert file system from VAX to PDP-11 format.

Ispecial is the name of a special file containing a file system to be converted (e.g.; /dev/rrp1). The optional ospecial is the name of the special file to receive the results of the conversion. If ospecial is specified the entire contents of ispecial are copied to ospecial before the conversion is performed. If ospecial is not specified an in-place conversion of ispecial is performed. The following items should be noted before executing fscv:

1. A file system consistency check (fsck(1M)) should be performed on ispecial immediately prior to executing fscv.
2. Neither ispecial nor the optional ospecial should contain a mounted file system during execution of fscv. Modification to either the input or the output file system while fscv is executing will probably corrupt the converted file system.
3. A backup of ispecial (see volcopy(1M)) is highly recommended if an in-place conversion is to be performed. System crashes, I/O errors, etc., during execution of fscv may destroy the file system contained in ispecial. Also, if the optional ospecial is specified any data contained in that special file will be over written.
4. If the optional ospecial is specified, this special file must be large enough to contain the entire contents of ispecial. See the appropriate special files in section 4.

EXAMPLES
Copy and convert a file system from PDP-11 to VAX format:
/etc/fscv -v /dev/rrp0 /dev/rrp10
Perform an in-place conversion from VAX to PDP-11 format:
/etc/fscv -p /dev/rrp10

BUGS
The boot block is not modified during conversion. The resulting file system will not be bootable. No data contained in the files of the file system are modified.

SEE ALSO
fsck(1M), volcopy(1M).
NAME
fsdb — file system debugger

SYNOPSIS
/etc/fsdb special [ - ]

DESCRIPTION
Fsdb can be used to patch up a damaged file system after a crash. It has
conversions to translate block and i-numbers into their corresponding disk
addresses. Also included are mnemonic offsets to access different parts of
an i-node. These greatly simplify the process of correcting control block
entries or descending the file system tree.

Fsdb contains several error checking routines to verify i-node and block
addresses. These can be disabled if necessary by invoking fsdb with the
optional - argument or by the use of the O symbol. (Fsdb reads the i-size
and f-size entries from the superblock of the file system as the basis for
these checks.)

Numbers are considered decimal by default. Octal numbers must be
prefixed with a zero. During any assignment operation, numbers are
checked for a possible truncation error due to a size mismatch between
source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as
block I/O. A buffer management routine is used to retain commonly used
blocks of data in order to reduce the number of read system calls. All
assignment operations result in an immediate write-through of the
corresponding block.

The symbols recognized by fsdb are:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>absolute address</td>
</tr>
<tr>
<td>i</td>
<td>convert from i-number to i-node address</td>
</tr>
<tr>
<td>b</td>
<td>convert to block address</td>
</tr>
<tr>
<td>d</td>
<td>directory slot offset</td>
</tr>
<tr>
<td>+, -</td>
<td>address arithmetic</td>
</tr>
<tr>
<td>q</td>
<td>quit</td>
</tr>
<tr>
<td>&gt;, &lt;</td>
<td>save, restore an address</td>
</tr>
<tr>
<td>=</td>
<td>numerical assignment</td>
</tr>
<tr>
<td>= +</td>
<td>incremental assignment</td>
</tr>
<tr>
<td>= -</td>
<td>decremental assignment</td>
</tr>
<tr>
<td>= '</td>
<td>character string assignment</td>
</tr>
<tr>
<td>o</td>
<td>error checking flip flop</td>
</tr>
<tr>
<td>p</td>
<td>general print facilities</td>
</tr>
<tr>
<td>f</td>
<td>file print facility</td>
</tr>
<tr>
<td>B</td>
<td>byte mode</td>
</tr>
<tr>
<td>W</td>
<td>word mode</td>
</tr>
<tr>
<td>D</td>
<td>double word mode</td>
</tr>
<tr>
<td>!</td>
<td>escape to shell</td>
</tr>
</tbody>
</table>

The print facilities generate a formatted output in various styles. The
current address is normalized to an appropriate boundary before printing
begins. It advances with the printing and is left at the address of the last
item printed. The output can be terminated at any time by typing the
delete character. If a number follows the p symbol, that many entries are
printed. A check is made to detect block boundary overflows since logically
sequential blocks are generally not physically sequential. If a count of zero
is used, all entries to the end of the current block are printed. The print
options available are:
FSDB(1M)

i  print as i-nodes
  
d  print as directories
  
o  print as octal words
  
e  print as decimal words
  
c  print as characters
  
b  print as octal bytes

The f symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the f symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A .B or .D is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

- md: mode
- ln: link count
- uid: user ID number
- gid: group ID number
- s0: high byte of file size
- s1: low word of file size
- af: data block numbers (0 — 12)
- at: access time
- mt: modification time
- maj: major device number
- min: minor device number

EXAMPLES

386i  prints i-number 386 in an i-node format. This now becomes the current working i-node.

ln=4  changes the link count for the working i-node to 4.

ln=+1  increments the link count by 1.

fc  prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd  prints the first 32 directory entries for the root i-node of this file system.

d5i.fc  changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first 512 bytes of the file are then printed in ASCII.

1b.p0o  prints the superblock of this file system in octal.
2i.a0b.d7 = 3 changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.

d7.nm = "name" changes the name field in the directory slot to the given string. Quotes are optional when used with nm if the first character is alphabetic.

SEE ALSO
fsck(1M), dir(5), fs(5).
NAME
fsend  —  send files to the HONEYWELL 6000

SYNOPSIS
fsend [ options ] [ files ]

DESCRIPTION
Fsend arranges to have one or more UNIX files sent to HONEYWELL GCOS. GCOS identification must appear in the UNIX password file (see passwd(5)), or be supplied by the -i option. If no names appear, the standard input is sent; thus fsend may be used as a filter.

Normally, the catalog on the HONEYWELL file system in which the new file will appear is the same as the UNIX login name of the person who issues the command. If, however, a user has a different name in the third field of the GCOS "ident card image" (which image is extracted from the UNIX password file; see passwd(5)), this name is taken as the GCOS catalog name. Whatever GCOS catalog is finally used, the user must have arranged that the user ID "network" has create permission on that catalog, or read and write permission on the individual files. The latter is more painful but preferred if access to other files in the catalog is to be fully controlled. This can be accomplished with the GCOS commands:

    filsys mc <user ID>,(c)/network/

or

    filsys cf <file>,(r,w)/network/,b/<initial-size>,unlimited/

The name of the GCOS file is ordinarily the same as the name of the UNIX file. When the standard input is sent, the GCOS file is normally taken to be pipe.end.

The following options, each as a separate argument, (or in the case of -u and -f, as two separate arguments), may appear in any order, but must precede all file name arguments.

- a  Send succeeding files as ASCII (default). If the last character of the file is not a new-line, one is added. All other characters are preserved.
- b  Send succeeding files as binary. Each UNIX byte is right justified in a GCOS byte and the bytes packed into 120-byte logical records (30 GCOS words). The last record is padded out with NULs.
- c  Make copies of the files to be sent before returning to the user.
- r  Remove the files after sending them.
- f  Use the next argument as the GCOS file name for the succeeding file.
- i  Supply the GCOS "ident card" image as the parameter
    -iMxxxx,Myyy where Mxxxx is the GCOS job number and Myyy the GCOS bin number.
- m  When transmission is complete, report by mail(1) the so-called
    snumb of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is the default option.
- n  Do not report the completion of transmission by mail(1).
- o  Print the on-line GCOS accounting output.
- t  Toss out the on-line GCOS accounting output. This is the default option.
- sn  Submit job to GCOS with service grade n (n=1, 2, 3). Default is -s1.
- u  Use the next argument as the GCOS catalog name for all files.
Send succeeding files to be archived by the GCOS archive command.

**EXAMPLE**

The command:

```
fsend -t -u unixsup -b -f gfile ufile
```

will send the binary UNIX file `ufile` to become the GCOS file `unixsup/gfile`, and will not produce any on-line GCOS accounting output.

**FILES**

```
/etc/passwd  user's identification and GCOS ident card.
/usr/lib/dpd  sending daemon.
/usr/spool/dpd/*  spool area.
```

**SEE ALSO**

dpd(1C), dpr(1C), fget(1C), gcat(1C), mail(1).
NAME
fwtmp, wtmpfix — manipulate wtmp records

SYNOPSIS
fwtmp [-ic]
wtmpfix [files]

DESCRIPTION
**Fwtmp**
Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in wtmp to formatted ASCII records. The ASCII version is useful to enable editing, via ed(1), bad records or general purpose maintenance of the file.

The argument -ic is used to denote that input is in ASCII form, and output is to be written in binary form.

**Wtmpfix**
Wtmpfix examines the standard input or named files in wtmp format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A - can be used in place of files to indicate the standard input. If time/date corrections are not made, acctcon1 will fault when it encounters certain date change records.

Each time the date is set while operating in multi-user mode, a pair of date change records are written to /usr/adm/wtmp. The first record is the old date denoted by | in the name field. The second record specifies the new date and is denoted by a { in the name field. Wtmpfix uses these records to synchronize all time stamps in the file.

FILES
/usr/adm/wtmp
/usr/include/utmp.h

SEE ALSO
acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M), acct(2), acct(5), utmp(5).
NAME

gcat - send phototypesetter output to the HONEYWELL 6000

SYNOPSIS

gcat [ options ] [ files ]

DESCRIPTION

Gcat arranges to have troff(1) output sent to the phototypesetter or debugging devices (STARE or line printer) attached to the HONEYWELL system. GCOS identification must appear in the UNIX password file (see passwd(5)), or be supplied by the -i option. If no file name appears, the standard input is sent; thus gcat may be used as an output pipe for troff(1).

The option -g (for GCOS) must be used with the troff(1) command to make things work properly. This command string sends output to the GCOS phototypesetter:

troff -g file | gcat

The following options, each as a separate argument, and in any combination (multiple outputs are permitted), may be given after gcat:

- ph Send output to the phototypesetter. This is a default option.
- st Send output to STARE for fast turn-around.
- tx Send output as text to the line printer (useful for checking spelling, hyphenation, pagination, etc.).
- du Send output to the line printer, dummied up to make the format correct. Because many characters are dropped, the output is unreadable, but useful for seeing the shape (margins, etc.) of the document.
- c Make a copy of the file to be sent before returning to the user.
- r Remove the file after sending it.
- f Use the next argument as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs, especially when gcat is being used as a filter).
- i Supply the GCOS "ident card" image as the parameter -iMxxxx,Myyy where Mxxxx is the GCOS job number and Myyy the GCOS bin number.
- m When transmission is complete, report by mail(1) the so-called snumb of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is a default option.
- n Do not report the completion of transmission by mail(1).
- e Print the on-line GCOS accounting output.
- t Toss out the on-line GCOS accounting output. This is a default option.
- sn Submit job to GCOS with service grade n (n = 1, 2, 3). Default is -s1.

If none of the output options are specified, phototypesetter output (-ph) is assumed by default.

EXAMPLE

The command:

troff -g myfile | gcat -st -im1234,M567,myname -f myfile

will send the output of troff(1) to STARE, with the GCOS "ident card" specifying "M1234,M567,MYNAME", and will report back that myfile has been sent.

FILES
SEE ALSO

dpd(1C), dpr(1C), fget(1C), fsend(1C), troff(1).
NAME
gcosmail — send mail to HIS user

SYNOPSIS
gcosmail [ option ... ] [ HISuserid ... ]

DESCRIPTION
Gcosmail takes the standard input up to an end of file and sends it as mail to the named users on the HONEYWELL 6000 system, using the HIS mail command. The following options are recognized by gcosmail:

- f Use the next argument as a dummy file name to report back in the mail. (This is useful for distinguishing multiple runs).
- i Supply the GCOS "ident card" image as the parameter
  -iMxxxx,Myyy where Mxxxx is the GCOS job number and Myyy is the GCOS bin number.
- m When transmission is complete, report by mail(1) the so-called snumb of the receiving GCOS job. The mail is sent by the UNIX daemon; there is no guarantee that the GCOS job ran successfully. This is a default option.
- a Do not report the completion of transmission by mail(1).
- o Print the on-line GCOS accounting output.
- t Toss out the on-line GCOS accounting output. This is a default option.
- s Submit job to GCOS with service grade n (n = 1, 2, 3). Default is -s 1.

FILES
/etc/passwd user's identification and GCOS ident card.
/usr/lib/dpd sending daemon.
/usr/spool/dpd/* spool area.

SEE ALSO
dpd(1C), dpr(1C), fsend(1C).
NAME
hpd, erase, hardcopy, tekset, td — graphical device routines and filters

SYNOPSIS
hpd [-options] [GPS file ...]
erase
hardcopy
tekset
td [-eurn] [GPS file ...]

DESCRIPTION
All of the commands described below reside in /usr/bin/graf (see graphics(1G)).

hpd  Hpd translates a GPS (see gps(5)), to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in file unless the -u or -r option is provided. If no file is given, the standard input is assumed. Options are:
   cn Select character set n, n between 0 and 5 (see the HP7221A Plotter Operating and Programming Manual, Appendix A).
   pn Select pen numbered n, n between 1 and 4 inclusive.
   rn Window on GPS region n, n between 1 and 25 inclusive.
   sn Slant characters n degrees clockwise from the vertical.
   u Window on the entire GPS universe.
   xdn Set x displacement of the viewport’s lower left corner to n inches.
   xvn Set width of viewport to n inches.
   ydn Set y displacement of the viewport’s lower left corner to n inches.
   yvn Set height of viewport to n inches.

erase  Erase sends characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy  When issued at a Tektronix display terminal with a hard copy unit, hardcopy generates a screen copy on the unit.

tekset  Tekset sends characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td  Td translates a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in file unless the -u or -r option is provided. If no file is given, the standard input is assumed. Options are:
   e  Do not erase screen before initiating display.
   rn  Display GPS region n, n between 1 and 25 inclusive.
   u  Display the entire GPS universe.

SEE ALSO
  graphics(1G), ged(1G), gps(5).
NAME

ged — graphical editor

SYNOPSIS

ged [-euRrn] [GPS file ...]

DESCRIPTION

Ged is an interactive graphical editor used to display, construct, and edit GPS files on Tektronix 4010 series display terminals. If GPS file(s) are given, ged reads them into an internal display buffer and displays the buffer. The GPS in the buffer can then be edited. If — is given as a file name, ged reads a GPS from the standard input.

Ged accepts the following command line options:

- e  Do not erase the screen before the initial display.
- n Display region number n.
- u Display the entire GPS universe.
- R Restricted shell invoked on use of !.

A GPS file is composed of instances of three graphical objects: lines, arc, and text. Arc and lines objects have a start point, or object-handle, followed by zero or more points, or point-handles. Text has only an object-handle. The objects are positioned within a Cartesian plane, or universe, having 64K (-32K to +32K) points, or universe-units, on each axis. The universe is divided into 25 equal sized areas called regions. Regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

Ged maps rectangular areas, called windows, from the universe onto the display screen. Windows allow the user to view pictures from different locations and at different magnifications. The universe-window is the window with minimum magnification, i.e. the window that views the entire universe. The home-window is the window that completely displays the contents of the display buffer.

COMMANDS

Ged commands are entered in stages. Typically each stage ends with a <cr> (return). Prior to the final <cr> the command may be aborted by typing rubout. The input of a stage may be edited during the stage using the erase and kill characters of the calling shell. The prompt * indicates that ged is waiting at stage 1.

Each command consists of a subset of the following stages:

1. Command line

A command line consists of a command name followed by argument(s) followed by a <cr>. A command name is a single character. Command arguments are either option(s) or a file-name. Options are indicated by a leading -.

2. Text

Text is a sequence of characters terminated by an unescaped <cr>. (120 lines of text maximum.)

3. Points

Points is a sequence of one or more screen locations (maximum of 30) indicated either by the terminal crosshairs or by name. The prompt for entering points is the appearance of the crosshairs. When the crosshairs are visible, typing:

sp (space) enters the current location as a point. The point is identified with a number.
S_n enters the previous point numbered n.

>_x labels the last point entered with the upper case letter x.

S_x enters the point labeled x.

. establishes the previous points as the current points. At the start of a command the previous points are those locations given with the previous command.

= echoes the current points.

$.n enters the point numbered n from the previous points.

# erases the last point entered.

@ erases all of the points entered.

4. Pivot The pivot is a single location, entered by typing <cr> or by using the $ operator, and indicated with a •.

5. Destination The destination is a single location entered by typing <cr> or by using $.

COMMAND SUMMARY

In the summary, characters typed by the user are printed in bold. Command stages are printed in italics. Arguments surrounded by brackets "[]" are optional. Parentheses "( )" surrounding arguments separated by "or" means that exactly one of the arguments must be given.

Construct commands:

Arc [echo, style, weight] points

Box [echo, style, weight] points

Circle [echo, style, weight] points

Hardware [echo] text points

Lines [echo, style, weight] points

Text [angle, echo, height, mid-point, right-point, text, weight] text points

Edit commands:

Delete ( - (universe or view) or points )

Edit [angle, echo, height, style, weight] ( - (universe or view) or points )

Kopy [echo, points, x] points pivot destination

Move [echo, points, x] points pivot destination

Rotate [angle, echo, kopy, x] points pivot destination

Scale [echo, factor, kopy, x] points pivot destination

View commands:
coordinates points
erase
new-display
object-handles ( - (universe or view) or points )
point-handles  ( ~ (labelled-points or universe or view) or points )
view  ( ~ (home or universe or region) or [ -x ] pivot destination )
x  [ -view ] points
zoom  [ -out ] points

Other commands:
quit or Quit
read  [ -angle, echo, height, mid-point, right-point, text, weight ]
       file-name [ destination ]
set  [ -angle, echo, factor, height, kopy, mid-point, points,
       right-point, style, text, weight, x ]
write  file-name
!command
?

Options:

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specified as an option, the default value for the parameter will be used (see set below). The format of command options is

- option [, option ]

where option is key[letter][value]. Flags take on the values of true or false indicated by + and - respectively. If no value is given with a flag, true is assumed.

Object options:

angle  Angle of n degrees.
echo  When true, echo additions to the display buffer.
factor  Scale factor is n percent.
height  Height of text is n universe-units (0 ≤ n < 1280).
kopy  When true, copy rather than move.
mid-point  When true, mid-point is used to locate text string.
points  When true, operate on points otherwise operate on objects.
right-point  When true, right-point is used to locate text string.
style  Line style set to one of following types:
so  solid
da  dashed
dd  dot-dashed
d  dotted
dl  long-dashed
text

When false, text strings are outlined rather than drawn.

weighttype

Sets line weight to one of following types:

\( n \) narrow
\( m \) medium
\( b \) bold

Area options:

home Reference the home-window.
out Reduce magnification.
regionn Reference region \( n \).
universe Reference the universe-window.
view Reference those objects currently in view.
x Indicate the center of the referenced area.

COMMAND DESCRIPTIONS

Construct commands:

Arc and Lines

behave similarly. Each consists of a command line followed by points. The first point entered is the object-handle. Successive points are point-handles. Lines connects the handles in numerical order. Arc fits a curve to the handles (currently a maximum of 3 points will be fit with a circular arc; splines will be added in a later version).

Box and Circle

are special cases of Lines and Arc, respectively. Box generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle would connect the first point entered with the last point. The first point is the object-handle. Point-handles are created at each of the vertices. Circle generates a circular arc centered about the point numbered zero and passing through the last point. The circle's object-handle coincides with the last point. A point-handle is generated 180 degrees around the circle from the object-handle.

Text and Hardware

generate text objects. Each consists of a command line, text and points. Text is a sequence of characters delimited by \(<er>\). Multiple lines of text may be entered by preceding a \(<cr>\) with a backslash (i.e. \\
\(<cr>\)). The Text command creates software generated characters. Each line of software text is treated as a separate text object. The first point entered is the object-handle for the first line of text. The Hardware command sends the characters in text uninterpreted to the terminal.

Edit commands:

Edit commands operate on portions of the display buffer called defined-areas. A defined-area is referenced either with an area option or interactively. If an area option is not given, the perimeter of the defined-area is indicated by points. If no point is entered, a small defined-area is built around the location of the \(<er>\). This is useful to reference a single point. If only one point is entered, the location of the \(<er>\) is taken in conjunction with the point to indicate a diagonal of a rectangle. A defined-area referenced by points will be outlined with dotted lines.

Delete

removes all objects whose object-handle lies within a defined-area. The universe option removes all objects and erases the screen.
Edit modifies the parameters of the objects within a defined-area. Parameters that can be edited are:

- **angle**: angle of text
- **height**: height of text
- **style**: style of lines and arc
- **weight**: weight of lines, arc, and text.

**Kopy** (or **Move**) copies (or moves) object- and/or point-handles within a defined-area by the displacement from the pivot to the destination.

**Rotate**

rotates objects within a defined-area around the pivot. If the kopy flag is true then the objects are copied rather than moved.

**Scale**

For objects whose object-handles are within a defined-area, point displacements from the pivot are scaled by factor percent. If the kopy flag is true then the objects are copied rather than moved.

**View commands:**

- **coordinates** prints the location of point(s) in universe- and screen-units.
- **erase** clears the screen (but not the display buffer).
- **new-display** erases the screen then displays the display buffer.
- **object-handles** (or point-handles) labels object- (and/or point-handles) that lie within the defined-area with O (or P). point-handles identifies labelled points when the labelled-points flag is true.
- **view** moves the window so that the universe point corresponding to the pivot coincides with the screen point corresponding to the destination. Options for home, universe, and region display particular windows in the universe.
- **x** indicates the center of a defined-area. Option view indicates the center of the screen.
- **zoom** decreases (zoom out) or increases the magnification of the viewing window based on the defined-area. For increased magnification, the window is set to circumscribe the defined-area. For a decrease in magnification the current window is inscribed within the defined-area.

**Other commands:**

- **quit** or **Quit** exit from ged. quit responds with ? if the display buffer has not been written since the last modification.
- **read** inputs the contents of a file. If the file contains a GPS it is read directly. If the file contains text it is converted into text object(s). The first line of a text file begins at destination.
- **set** when given option(s) resets default parameters, otherwise it prints current default values.
- **write** outputs the contents of the display buffer to a file.
! escapes *ged* to execute a UNIX command.
? lists *ged* commands.

SEE ALSO

*graphics*(1G), *gdev*(1G), *rsh*(1), *gps*(5).

*A Tutorial Introduction to the Graphical Editor* by A. R. Feuer.
NAME
get — get a version of an SCCS file

SYNOPSIS
[-l[p]] [-p] [-m] [-n] [-s] [-b] [-g] [-t] file ...

DESCRIPTION
Get generates an ASCII text file from each named SCCS file according to the
specifications given by its keyletter arguments, which begin with —. The
arguments may be specified in any order, but all keyletter arguments apply
to all named SCCS files. If a directory is named, get behaves as though
each file in the directory were specified as a named file, except that non-
SCCS files (last component of the path name does not begin with s.) and
unreadable files are silently ignored. If a name of — is given, the standard
input is read; each line of the standard input is taken to be the name of an
SCCS file to be processed. Again, non-SCCS files and unreadable files are
silently ignored.

The generated text is normally written into a file called the g-file whose
name is derived from the SCCS file name by simply removing the leading s.; (see also FILES, below).

Each of the keyletter arguments is explained below as though only one
SCCS file is to be processed, but the effects of any keyletter argument
applies independently to each named file.

-rSID The SCCS IDENTIFICATION string (SID) of the version (delta) of
an SCCS file to be retrieved. Table 1 below shows, for the most
useful cases, what version of an SCCS file is retrieved (as well as
the SID of the version to be eventually created by delta(l) if the
-e keyletter is also used), as a function of the SID specified.

-ecutoff Cutoff date-time, in the form:
YY[MM][DD][HH][MM][SS]]

No changes (deltas) to the SCCS file which were created after the
specified cutoff date-time are included in the generated ASCII
text file. Units omitted from the date-time default to their
maximum possible values; that is, —c7502 is equivalent to
-c750228235959. Any number of non-numeric characters may
separate the various 2 digit pieces of the cutoff date-time. This feature allows one to specify a cutoff date in the form:
"—c77/2/2 9:22:25". Note that this implies that one may use the %E% and %U% identification keywords (see below) for
nested gets within, say the input to a send(lC) command:

"!get —c%E% %U%" s.file

-e Indicates that the get is for the purpose of editing or making a
change (delta) to the SCCS file via a subsequent use of delta(l).
The —e keyletter used in a get for a particular version (SID) of
the SCCS file prevents further gets for editing on the same SID
until delta is executed or the j (joint edit) flag is set in the SCCS
file (see admin(l)). Concurrent use of get —e for different
SIDs is always allowed.

If the g-file generated by get with an —e keyletter is accidentally
ruined in the process of editing it, it may be regenerated by re-
executing the get command with the —k keyletter in place of the
-e keyletter.
SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see \texttt{admin(1)}) are enforced when the \texttt{\textendash e} keyletter is used.

\texttt{-b} Used with the \texttt{\textendash e} keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the \texttt{b} flag is not present in the file (see \texttt{admin(1)}) or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.) Note: A branch delta may always be created from a non-leaf delta.

\texttt{-i\textit{list}} A list of deltas to be included (forced to be applied) in the creation of the generated file. The \textit{list} has the following syntax:

\begin{verbatim}
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID SID
\end{verbatim}

SID, the SCCS Identification of a delta, may be in any form shown in the “SID Specified” column of Table 1. Partial SIDs are interpreted as shown in the “SID Retrieved” column of Table 1.

\texttt{-x\textit{list}} A list of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the \texttt{-i} keyletter for the \textit{list} format.

\texttt{-k} Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The \texttt{-k} keyletter is implied by the \texttt{-e} keyletter.

\texttt{-l[p]} Causes a delta summary to be written into an \texttt{l-file}. If \texttt{-lp} is used then an \texttt{l-file} is not created; the delta summary is written on the standard output instead. See \texttt{FILES} for the format of the \texttt{l-file}.

\texttt{-p} Causes the text retrieved from the SCCS file to be written on the standard output. No \texttt{g-file} is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the \texttt{-s} keyletter is used, in which case it disappears.

\texttt{-s} Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

\texttt{-m} Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

\texttt{-n} Causes each generated text line to be preceded with the \texttt{\%M\%} identification keyword value (see below). The format is: \texttt{\%M\%} value, followed by a horizontal tab, followed by the text line. When both the \texttt{-m} and \texttt{-n} keyletters are used, the format is: \texttt{\%M\%} value, followed by a horizontal tab, followed by the \texttt{-m} keyletter generated format.

\texttt{-g} Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an \texttt{l-file}, or to verify the existence of a particular SID.

\texttt{-t} Used to access the most recently created (“top”) delta in a given release (e.g., \texttt{-r1}), or release and level (e.g., \texttt{-r1.2}).
The delta sequence number of the SCCS file delta (version) to be retrieved (see sccsfile(5)). This keyletter is used by the *comb(1)* command; it is not a generally useful keyletter, and users should not use it. If both the -r and -a keyletters are specified, the -a keyletter is used. Care should be taken when using the -a keyletter in conjunction with the -e keyletter, as the SID of the delta to be created may not be what one expects. The -r keyletter can be used with the -a and -e keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the -e keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the -i keyletter is used included deltas are listed following the notation "Included"; if the -x keyletter is used, excluded deltas are listed following the notation "Excluded".

<table>
<thead>
<tr>
<th><strong>TABLE 1. Determination of SCCS Identification String</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SID</strong> Specified</td>
</tr>
<tr>
<td>none‡</td>
</tr>
<tr>
<td>none‡</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>R.L</td>
</tr>
<tr>
<td>R.L</td>
</tr>
<tr>
<td>R.L</td>
</tr>
<tr>
<td>R.L.B</td>
</tr>
<tr>
<td>R.L.B</td>
</tr>
<tr>
<td>R.L.B.S</td>
</tr>
<tr>
<td>R.L.B.S</td>
</tr>
<tr>
<td>R.L.B.S</td>
</tr>
</tbody>
</table>

- **"R", "L", "B", and "S"** are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "mR.mL" means "the maximum level number within release R"; "mR.(mB + 1).1" means "the first sequence number on the new branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

- **"hR"** is the highest existing release that is lower than the specified, nonexistent, release R.
This is used to force creation of the first delta in a new release.

The -b keyletter is effective only if the b flag (see admin(1)) is present in the file. An entry of - means "irrelevant".

This case applies if the d (default SID) flag is not present in the file. If the d flag is present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

**IDENTIFICATION KEYWORDS**

Identifying information is inserted into the text retrieved from the SCCS file by replacing **identification keywords** with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>%M%</td>
<td>Module name: either the value of the m flag in the file (see admin(1)), or if absent, the name of the SCCS file with the leading s. removed.</td>
</tr>
<tr>
<td>%I%</td>
<td>SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.</td>
</tr>
<tr>
<td>%R%</td>
<td>Release.</td>
</tr>
<tr>
<td>%L%</td>
<td>Level.</td>
</tr>
<tr>
<td>%B%</td>
<td>Branch.</td>
</tr>
<tr>
<td>%S%</td>
<td>Sequence.</td>
</tr>
<tr>
<td>%D%</td>
<td>Current date (YY/MM/DD).</td>
</tr>
<tr>
<td>%H%</td>
<td>Current date (MM/DD/YY).</td>
</tr>
<tr>
<td>%T%</td>
<td>Current time (HH:MM:SS).</td>
</tr>
<tr>
<td>%E%</td>
<td>Date newest applied delta was created (YY/MM/DD).</td>
</tr>
<tr>
<td>%G%</td>
<td>Date newest applied delta was created (MM/DD/YY).</td>
</tr>
<tr>
<td>%U%</td>
<td>Time newest applied delta was created (HH:MM:SS).</td>
</tr>
<tr>
<td>%Y%</td>
<td>Module type: value of the t flag in the SCCS file (see admin(1)).</td>
</tr>
<tr>
<td>%F%</td>
<td>SCCS file name.</td>
</tr>
<tr>
<td>%P%</td>
<td>Fully qualified SCCS file name.</td>
</tr>
<tr>
<td>%Q%</td>
<td>The value of the q flag in the file (see admin(1)).</td>
</tr>
<tr>
<td>%C%</td>
<td>Current line number. This keyword is intended for identifying messages output by the program such as &quot;this shouldn't have happened&quot; type errors. It is not intended to be used on every line to provide sequence numbers.</td>
</tr>
<tr>
<td>%Z%</td>
<td>The 4-character string @( #) recognizable by what(1).</td>
</tr>
<tr>
<td>%W%</td>
<td>A shorthand notation for constructing what(1) strings for UNIX program files. %W% = %Z%%M%&lt;horizontal-tab&gt;%I%</td>
</tr>
<tr>
<td>%A%</td>
<td>Another shorthand notation for constructing what(1) strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%</td>
</tr>
</tbody>
</table>

**FILES**

Several auxiliary files may be created by get. These files are known generically as the g-file, l-file, p-file, and z-file. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form s.module-name, the auxiliary files are named by replacing the leading s with the tag. The g-file is an exception to this scheme: the g-file is named by removing the s. prefix. For example, s.xyz.c, the auxiliary file names would be xyz.c, l.xyz.c, p.xyz.c, and z.xyz.c, respectively.

The g-file, which contains the generated text, is created in the current directory (unless the -p keyletter is used). A g-file is created in all cases, whether or not any lines of text were generated by the get. It is owned by the real user. If the -k keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in
the current directory.

The *I*-file contains a table showing which deltas were applied in generating the retrieved text. The *I*-file is created in the current directory if the -I keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *I*-file have the following format:

a. A blank character if the delta was applied;
   • otherwise.
b. A blank character if the delta was applied or wasn't applied and ignored;
   • if the delta wasn't applied and wasn't ignored.
c. A code indicating a "special" reason why the delta was or was not applied:
   "I": Included.
   "X": Excluded.
   "C": Cut off (by a -c keyletter).
d. Blank.
e. SCCS identification (SID).
f. Tab character.
g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
h. Blank.
i. Login name of person who created delta.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p*-file is used to pass information resulting from a *get* with an -e keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an -e keyletter for the same SID until *delta* is executed or the joint edit flag, j, (see *admin*(1)) is set in the SCCS file. The *p*-file is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p*-file is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the -i keyletter argument if it was present, followed by a blank and the -x keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p*-file at any time; no two lines can have the same new delta SID.

The *z*-file serves as a lock-out mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z*-file is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p*-file apply for the *z*-file. The *z*-file is created mode 444.

SEE ALSO


DIAGNOSTICS
Use *help*(1) for explanations.

BUGS
If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the -e keyletter is used.
NAME
gopt — parse command options

SYNOPSIS
set -- "getopt optstring $*"

DESCRIPTION
Getopt is used to break up options in command lines for easy parsing by
shell procedures, and to check for legal options. Optstring is a string of
recognized option letters (see getopt(3C)); if a letter is followed by a colon,
the option is expected to have an argument which may or may not be
separated from it by white space. The special option -- is used to delimit
the end of the options. Getopt will place -- in the arguments at the end
of the options, or recognize it if used explicitly. The shell arguments ($1
$2 ...) are reset so that each option is preceded by a - and in its own
shell argument; each option argument is also in its own shell argument.

DIAGNOSTICS
Getopt prints an error message on the standard error when it encounters an
option letter not included in optstring.

EXAMPLE
The following code fragment shows how one might process the arguments
for a command that can take the options a and b, and the option o, which
requires an argument.

```
set -- "getopt abo: $*"
if [ $? != 0 ] then
    echo $USAGE
    exit 2
fi
for i in $*
do
case $i in
    -a | -b) FLAG=$i; shift;;
    -o) OARG=$2; shift; shift;;
    --) shift; break;;
esacct
done
```

This code will accept any of the following as equivalent:

```
cmd -a oarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO
sh(1), getopt(3C).
NAME
graph — draw a graph

SYNOPSIS
graph [ options ]

DESCRIPTION
Graph with no options takes pairs of numbers from the standard input as
abscissas and ordinates of a graph. Successive points are connected by
straight lines. The graph is encoded on the standard output for display by
the tplot(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that
string is printed as a label beginning on the point. Labels may be surronded
with quotes , in which case they may be empty or contain blanks and
numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

- Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by -x).

-b Break (disconnect) the graph after each label in the input.

-c Character string given by next argument is default label for each point.

-g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

-l Next argument is label for graph.

-m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

-s Save screen, don’t erase before plotting.

-x [1] If 1 is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.

-y [1] Similarly for y.

-h Next argument is fraction of space for height.

-w Similarly for width.

-r Next argument is fraction of space to move right before plotting.

-u Similarly to move up before plotting.

-t Transpose horizontal and vertical axes. (Option -x now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the -s option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO
graphics(1G), spline(1G), tplot(1G).

BUGS
Graph stores all points internally and drops those for which there isn’t room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.
NAME
graphics — access graphical and numerical commands

SYNOPSIS
graphics [ -r ]

DESCRIPTION
Graphics appends the path name /usr/bin/graf to the current $PATH value, changes the primary shell prompt to ", and executes a new shell. The directory /usr/bin/graf contains all of the Graphics subsystem commands. If the -r option is given, access to the graphical commands is created in a restricted environment; that is, $PATH is set to /usr/rbin:/usr/bin:/usr/bin/graf and the restricted shell, rsh(1), is invoked. To restore the environment that existed prior to issuing the graphics command, type EOT (control-d on most terminals). To logoff from the graphics environment, type quit.

The command line format for a command in graphics is command name followed by argument(s). An argument may be a file name or an option string. A file name is the name of any UNIX file except those beginning with -. The file name — is the name for the standard input. An option string consists of - followed by one or more option(s). An option consists of a keyletter possibly followed by a value. Options may be separated by commas.

The graphical commands have been partitioned into four groups.

Commands that manipulate and plot numerical data; see stat(1G).

Commands that generate tables of contents; see toc(1G).

Commands that interact with graphical devices; see gdev(1G) and ged(1G).

A collection of graphical utility commands; see gutil(1G).

A list of the graphics commands can be generated by typing whatis in the graphics environment.

SEE ALSO
gdev(1G), ged(1G), gutil(1G), stat(1G), toc(1G), gps(5).
UNIX Graphics Overview by A. R. Feuer.
NAME
greek — select terminal filter

SYNOPSIS
greek [ -Tterminal ]

DESCRIPTION
Greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character TELETYPE® Model 37 terminal (which is the nroff(1) default terminal) for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, greek attempts to use the environment variable STERM (see environ(7)). The following terminals are recognized currently:

<table>
<thead>
<tr>
<th>Terminal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>DASI 300 in 12-pitch.</td>
</tr>
<tr>
<td>300-12</td>
<td>DASI 300s in 12-pitch.</td>
</tr>
<tr>
<td>300s</td>
<td>DASI 300s.</td>
</tr>
<tr>
<td>300s-12</td>
<td>DASI 300s in 12-pitch.</td>
</tr>
<tr>
<td>450</td>
<td>DASI 450.</td>
</tr>
<tr>
<td>450-12</td>
<td>DASI 450 in 12-pitch.</td>
</tr>
<tr>
<td>1620</td>
<td>Diablo 1620 (alias DASI 450)</td>
</tr>
<tr>
<td>1620-12</td>
<td>Diablo 1620 (alias DASI 450) in 12-pitch.</td>
</tr>
<tr>
<td>2621</td>
<td>Hewlett-Packard 2621, 2640, and 2645.</td>
</tr>
<tr>
<td>2640</td>
<td>Hewlett-Packard 2621, 2640, and 2645.</td>
</tr>
<tr>
<td>2645</td>
<td>Hewlett-Packard 2621, 2640, and 2645.</td>
</tr>
<tr>
<td>4014</td>
<td>Tektronix 4014.</td>
</tr>
<tr>
<td>hp</td>
<td>Hewlett-Packard 2621, 2640, and 2645.</td>
</tr>
<tr>
<td>tek</td>
<td>Tektronix 4014.</td>
</tr>
</tbody>
</table>

FILES
/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO
300(1), 300s(1), 4014(1), 450(1), eqn(1), greek(7), hp(1), mm(1), nroff(1), tplot(1G), environ(7), term(7).
NAME
grep, egrep, fgrep — search a file for a pattern

SYNOPSIS
grep [ options ] expression [ files ]
egrep [ options ] [ expression ] [ files ]
fgrep [ options ] [ strings ] [ files ]

DESCRIPTION
Commands of the grep family search the input files (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. Grep patterns are limited regular expressions in the style of ed(1); it uses a compact non-deterministic algorithm. Egrep patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. Fgrep patterns are fixed strings; it is fast and compact. The following options are recognized:

-v All lines but those matching are printed.
-x (Exact) only lines matched in their entirety are printed (fgrep only).
-c Only a count of matching lines is printed.
-l Only the names of files with matching lines are listed (once), separated by new-lines.
-n Each line is preceded by its relative line number in the file.
-b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
-s The error messages produced for nonexistent or unreadable files are suppressed (grep only).
-e expression
Same as a simple expression argument, but useful when the expression begins with a - (does not work with grep).
-f file
The regular expression (egrep) or strings list (fgrep) is taken from the file.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters $, *, [, ., |, (, ), and \ in expression, because they are also meaningful to the shell. It is safest to enclose the entire expression argument in single quotes ‘...’.

Fgrep searches for lines that contain one of the strings separated by new-lines.

Egrep accepts regular expressions as in ed(1), except for \( and \), with the addition of:
1. A regular expression followed by + matches one or more occurrences of the regular expression.
2. A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by | or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is [ ], then *, ?, +, then concatenation, then | and new-line.

SEE ALSO
ed(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.

BUGS
  Ideally there should be only one `grep`, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.
  Lines are limited to 256 characters; longer lines are truncated.
  `Egrep` does not recognize ranges, such as `[a-z]`, in character classes.
NAME
gutil  — graphical utilities

SYNOPSIS
command-name [options] [files]

DESCRIPTION
Below is a list of miscellaneous device independent utility commands found in /usr/bin/gutil. If no files are given, input is from the standard input. All output is to the standard output. Graphical data is stored in GPS format; see gps(5).

bel  — send bel character to terminal

cvrtot  [-=sstring fstring istring tstring ] [args ] — options converter
Cvrtot reformats args (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An arg is either a file name (a string not beginning with a -, or a - by itself) or an option string (a string of options beginning with a -). Output is of the form:

-option -option . . . file name(s)

All options appear singularly and preceding any file names. Options that take values (e.g., -rl.1) or are two letters long must be described through options to cvrtot.

Cvrtot is usually used with set in the following manner as the first line of a shell procedure:

    set - "cvrtot = [options] $@"

Options to cvrtot are:

sstring  String accepts string values.
fstring  String accepts floating point numbers as values.
istring  String accepts integers as values.
tstring  String is a two letter option name that takes no value.
String is a one or two letter option name.

gd  [GPS files ] — GPS dump
Gd prints a human readable listing of GPS.

gtop  [-rn u ] [GPS files ] — GPS to plot(5) filter
Gtop transforms a GPS into plot(5) commands displayable by plot(1G) filters. GPS objects are translated if they fall within the window that circumscribes the first file unless an option is given.
Options:

rn  translate objects in GPS region n.
u  translate all objects in the GPS universe.

pd  [plot(5) files ]  — plot(5) dump
Pd prints a human readable listing of plot(5) format graphical commands.

ptog  [plot(5) files ]  — plot(5) to GPS filter
Ptog transforms plot(5) commands into a GPS.

quit  — terminate session

remcom  [files ]  — remove comments
Remcom copies its input to its output with comments removed. Comments are as defined in C (i.e., /* comment */).
whatis 

[ -o ] [ names ] — brief online documentation

Whatis prints a brief description of each name given. If no name is given, then the current list of description names is printed. whatis \* prints out every description.

Option:

- just print command options

tyoo 

file — pipe fitting

Tyoo is a piping primitive that deposits the output of a pipeline into a file used in the pipeline. Note that, without tyoo, this is not usually successful as it causes a read and write on the same file simultaneously.

SEE ALSO

graphics(1G), gps(5).
NAME
help — ask for help

SYNOPSIS
help [args]

DESCRIPTION
Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, help will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

- **type 1** Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., ge6, for message 6 from the get command).
- **type 2** Does not contain numerics (as a command, such as get)
- **type 3** Is all numeric (e.g., 212)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

FILES
/usr/lib/help directory containing files of message text.

DIAGNOSTICS
Use help(1) for explanations.
NAME
hp — handle special functions of HP 2640 and 2621-series terminals

SYNOPSIS
hp [-e] [-m]

DESCRIPTION
Hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most nroff(1) output. Typical uses are:

    nroff -h files ... | hp
    nroff -h -s ... files | hp

In the latter case, nroff will stop at the beginning of each page (including the first) and wait for you to hit line-feed (control-j) before resuming output.

Regardless of the hardware options on your terminal, hp tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags are as follows:

- e It is assumed that your terminal has the "display enhancements" feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, hp assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark-on-light, rather than the usual light-on-dark.

- m Requests minimization of output by removal of new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, hp provides the same set as does 300(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from neqn(1).

DIAGNOSTICS
"line too long" if the representation of a line exceeds 1,024 characters.
The exit codes are 0 for normal termination, 2 for all errors.

SEE ALSO
300(1), col(1), greek(1), neqn(1), tbl(1), troff(1).

BUGS
An "overstriking sequence" is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first printing character is shown (again, underlined or Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text "disappear"; in particular, tables generated by tbl(1) that contain vertical lines will often be missing the lines of text that contain the "foot" of a
vertical line, unless the input to `hp` is piped through `col(1)`.
Although some terminals do provide numerical superscript characters, no attempt is made to display them.
NAME

hyphen — find hyphenated words

SYNOPSIS

hyphen files

DESCRIPTION

Hyphen finds all the hyphenated words in files and prints them on the standard output. If no arguments are given, the standard input is used. Thus hyphen may be used as a filter.

BUGS

Hyphen can't cope with hyphenated italic (i.e., underlined) words; it will often miss them completely, or mangle them. Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.
NAME
id — print user and group IDs and names

SYNOPSIS
id

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.

SEE ALSO
logname(1), getuid(2), getgid(2).
NAME
install — install commands

SYNOPSIS
install [ -c dira ] [ -f dirb ] [ -i ] [ -n dirc ] [ -o ] [ -s ]
file [ dirx ... ]

DESCRIPTION
Install is a command most commonly used in "makefiles" (see make(1))
to install a file (updated target file) in a specific place within a file system.
Each file is installed by copying it into the appropriate directory, thereby
retaining the mode and owner of the original command. The program
prints messages telling the user exactly what files it is replacing or creating
and where they are going.

If no options or directories (dirx ...) are given, install will search (using
find(1)) a set of default directories (/bin, /usr/bin, /etc, /lib, and
/usr/lib, in that order) for a file with the same name as file. When the
first occurrence is found, install issues a message saying that it is overwrit­ing
that file with file, and proceeds to do so. If the file is not found, the
program states this and exits without further action.

If one or more directories (dirx ...) are specified after file, those directories
will be searched before the directories specified in the default list.

The meanings of the options are:

- c dira  Installs a new command in the directory specified in
dira. Looks for file in dira and installs it there if it is
not found. If it is found, install issues a message say­
ing that the file already exists, and exits without
overwriting it. May be used alone or with the - s
option.

- f dirb  Forces file to be installed in given directory, whether
or not one already exists. If the file being installed
does not already exist, the mode and owner of the
new file will be set to 755 and bin, respectively. If
the file already exists, the mode and owner will be
that of the already existing file. May be used alone
or with the - o or - s options.

- i      Ignores default directory list, searching only through
the given directories (dirx ...). May be used alone or
with any other options other than - c and - f.

- n dirc If file is not found in any of the searched directories,
it it put in the directory specified in dirc. The mode
and owner of the new file will be set to 755 and bin,
respectively. May be used alone or with any other
options other than - c and - f.

- o      If file is found, this option saves the "found" file by
copying it to OLDfile in the directory in which it was
found. May be used alone or with any other options
other than - c.

- s      Suppresses printing of messages other than error
messages. May be used alone or with any other
options.

SEE ALSO
mk(8).
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.

Fields are normally separated by blank, tab or new-line. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

- a n In addition to the normal output, produce a line for each unpairable line in file n, where n is 1 or 2.
- e s Replace empty output fields by string s.
- j n m Join on the mth field of file n. If n is missing, use the mth field in each file.
- 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.
- t c Use character c as a separator (tab character). Every appearance of c in a line is significant.

SEE ALSO
awk(1), comm(1), sort(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 wildly incongruous.
NAME
kas — assembler for the KMC11 microprocessor

SYNOPSIS
kas [ name ] [ -o name1 ] [ -d name2 ]

DESCRIPTION
Kas is an assembler/debugger/loader for the KMC11 microprocessor. The optional argument name specifies the input file; default is standard input. The optional argument -o indicates that the next argument name1 will be the output of the assembler; default is a.out. The optional argument -d indicates that the assembler is to be used in debug mode and that the next argument name2 is the device file name of the microprocessor. No output file is created in debug mode.

Error diagnostics are written on the standard error output and contain the input file name and line number and a brief description of the error. C preprocessor control lines to change the file name and line number are recognized. This allows the use of the preprocessor to expand the input before assembly.

FILES
a.out output object
/dev/kmc? microprocessor device
/lib/cpp C preprocessor

SEE ALSO
kun(1), kmc(4).

Assembler for the DEC KMC11 Microprocessor by L. A. Wehr.
NAME
        kill — terminate a process

SYNOPSIS
        kill [ -signo ] processid ...

DESCRIPTION
        Kill sends signal 15 (terminate) to the specified processes. This will
normally kill processes that do not catch or ignore the signal. The process
number of each asynchronous process started with & is reported by the
Shell (unless more than one process is started in a pipeline, in which case
the number of the last process in the pipeline is reported). Process num-
bers can also be found by using ps(1).

The details of the kill are described in kill(2). For example, if process
number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-
user.

If a signal number preceded by - is given as first argument, that signal is
sent instead of terminate (see signal(2)). In particular “kill -9 ...” is a
sure kill.

SEE ALSO
        ps(1), sh(1), kill(2), signal(2).
NAME
kun — un-assembler for the KMCII/DMCII microprocessor

SYNOPSIS
kun [ name ] [ -o name1 ]

DESCRIPTION
Kun is a un-assembler for the KMCII/DMCII microprocessors. It produces an output listing, acceptable to the assembler kas(1), from the input object.

The optional argument name specifies the input object, default is standard input. The format of the input is either assembler output (first word magic 0410), or formatted dump (first word magic 0440), or raw dump (anything else). In the first two cases, the header is ignored.

The optional argument -o indicates that the next argument name1 is to contain the output listing, default is standard output.

The input object is first scanned to determine branch destinations. Labels will be inserted at these locations with format Lin: , where int is the octal value of the location in words. Immediate values of instructions are also printed in octal. Page breaks are noted by the labels P0:, ... , P3:.

SEE ALSO
kas(1), kmc(4).
NAME

ld — link editor

SYNOPSIS

ld [ -suxXrdnim ] [ -o name ] [ -t name ] [ -V num ] file ...

DESCRIPTION

Ld combines several object programs into one; resolves external references; and searches libraries (as created by ar(1)). In the simplest case several object files are given, and ld combines them, producing an object module which can be either executed or become the input for a further ld run. (In the latter case, the -r option must be given to preserve the relocation bits.) The output of ld is left on a.out. This file is made executable if no errors occurred during the load and the -r flag was not specified.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

The symbols _etext, _edata and _end (etext, edata and end in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several flag arguments which are written preceded by a -.

Except for -1, they should appear before the file names.

-s "Strip" the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by strip(1). This option is turned off if there are any undefined symbols.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-1 This option is an abbreviation for a library name. -1 alone stands for /lib/libc.a, which is the standard system library for C and assembly language programs. -lx stands for /lib/libx.a, where x is a string. If that does not exist, ld tries /usr/lib/libx.a A library is searched when its name is encountered, so the placement of a -1 is significant.

-x Do not preserve local (non-.globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

-X Save local symbols except for those whose names begin with L. This option is used by cc to discard internally generated labels while retaining symbols local to routines.

-r Generate relocation bits in the output file so that it can be the subject of another ld run. This flag also prevents final definitions from being given to common symbols, and suppresses the "undefined symbol" diagnostics.
-d Force definition of common storage even if the -r flag is present.

-n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text. On the VAX 11/780, this option is on by default; use -N to turn it off.

-i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and -n is that here the data starts at location 0. This option is meaningful only on the PDP-11; it does nothing on the VAX.

-m The names of all files and archive members used to create the output file are written to the standard output.

-o The name argument after -o is used as the name of the ld output file, instead of a.out.

-t The name argument is taken to be a symbol name, and any references to or definitions of that symbol are listed, along with their types. There can be up to 16 occurrences of -tname on the command line.

-V The num argument is taken as a decimal version number identifying the a.out that is produced. Num must be in the range 0-32767. The version stamp is stored in the a.out header; see a.out(5).

FILES
/lib/lib%.a libraries
/usr/lib/lib%.a more libraries
a.out output file

SEE ALSO
ar(1), as(1), cc(1), a.out(5).
NAME
lex — generate programs for simple lexical tasks

SYNOPSIS
lex [ -rchn ] [ file ] ...

DESCRIPTION
Lex generates programs to be used in simple lexical analysis of text.

The input files (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file lex.yy.c is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in yytext, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in [abx—z] to indicate a, b, x, y, and z; and the operators *, +, and ? mean respectively any non-negative number of, any positive number of, and either zero or one occurrences of, the previous character or character class. The character . is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation r{d,e} in a rule indicates between d and e instances of regular expression r. It has higher precedence than |, but lower than *, ?, +, and concatenation. The character † at the beginning of an expression permits a successful match only immediately after a new-line, and the character $ at the end of an expression requires a trailing new-line. The character / in an expression indicates trailing context; only the part of the expression up to the slash is returned in yytext, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within * symbols or preceded by \. Thus [a—zA—Z]+ matches a string of letters.

Three subroutines defined as macros are expected: input() to read a character; unput(c) to replace a character read; and output(c) to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named yylex(), and the library contains a main() which calls it. The action REJECT on the right side of the rule causes this match to be rejected and the next suitable match executed; the function yymore() accumulates additional characters into the same yytext; and the function yyless(p) pushes back the portion of the string matched beginning at p, which should be between yytext and yytext+yyleng. The macros input and output use files yyin and yyout to read from and write to, defaulted to stdin and stdout, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes %% it is copied into the external definition area of the lex.yy.c file. All rules should follow a %% as in YACC. Lines preceding %% which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with { }. Note that curly brackets do not imply parentheses; only string substitution is done.

EXAMPLE
D [0—9]
%%
if printf("IF statement\n");
[a—z]+ printf("tag, value %s\n",yytext);
0[D]+ printf("octal number %s\n",yytext);
{D}+ printf("decimal number %s\n",yytext);
"++" printf("unary op\n");
"+" printf("binary op\n");
"*/" { loop:
    while (input() != '\*');
    switch (input())
    {
        case '/': break;
        case '\*': unput('\*');
        default: go to loop;
    }
}

The external names generated by lex all begin with the prefix yy or YY.

The flags must appear before any files. The flag -r indicates RATFOR actions, -c indicates C actions and is the default, -t causes the lex.yy.c program to be written instead to standard output, -v provides a one-line summary of statistics of the machine generated, -n will not print out the - summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

- %p \text{n} number of positions is \text{n} (default 2000)
- %n \text{n} number of states is \text{n} (500)
- %t \text{n} number of parse tree nodes is \text{n} (1000)
- %a \text{n} number of transitions is \text{n} (3000)

The use of one or more of the above automatically implies the -v option, unless the -n option is used.

SEE ALSO
yacc(1).

LEX — Lexical Analyzer Generator by M. E. Lesk and E. Schmidt.

BUGS
The -r option is not yet fully operational.
NAME
  line — read one line

SYNOPSIS
  line

DESCRIPTION
  Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO
  sh(1), read(2).
NAME
link, unlink — exercise link and unlink system calls

SYNOPSIS
/etc/link file1 file2
/etc/unlink file

DESCRIPTION
Link and unlink perform their respective system calls on their arguments, abandoning all error checking. These commands may only be executed by the super-user, who (it is hoped) knows what he or she is doing.

SEE ALSO
rm(1), link(2), unlink(2).
NAME
lint — a C program checker

SYNOPSIS
lint [ -a ] file ...

DESCRIPTION
Lint attempts to detect features of the C program files which are likely to be
bugs, non-portable, or wasteful. It also checks type usage more strictly
than the compilers. Among the things which are currently detected are
unreachable statements, loops not entered at the top, automatic variables
declared and not used, and logical expressions whose value is constant.
Moreover, the usage of functions is checked to find functions which return
values in some places and not in others, functions called with varying num-
biers of arguments, and functions whose values are not used.

It is assumed that all the files are to be loaded together; they are checked
for mutual compatibility. By default, lint uses function definitions from the
standard lint library llib-le.in; function definitions from the portable lint
library llib-port.in are used when lint is invoked with the -p option.

Any number of lint options may be used, in any order. The following
options are used to suppress certain kinds of complaints:

- Suppress complaints about assignments of long values to variables
  that are not long.
- Suppress complaints about break statements that cannot be
  reached. (Programs produced by lex or yacc will often result in a
  large number of such complaints.)
- Suppress complaints about casts that have questionable portability.
- Do not apply heuristic tests that attempt to intuit bugs, improve
  style, and reduce waste.
- Suppress complaints about functions and external variables used
  and not defined, or defined and not used. (This option is suitable
  for running lint on a subset of files of a larger program.)
- Suppress complaints about unused arguments in functions.
- Do not report variables referred to by external declarations but
  never used.

The following arguments alter lint's behavior:
- Do not check compatibility against either the standard or the porta-
  ble lint library.
- Attempt to check portability to other dialects (IBM and GCOS) of C.

The -D, -U, and -I options of cc(1) are also recognized as separate
arguments.

Certain conventional comments in the C source will change the behavior of
lint:

/\*NOTREACHED*/
at appropriate points stops comments about unreachable
code.

/\*VARARGS*/
suppresses the usual checking for variable numbers of
arguments in the following function declaration. The data
types of the first n arguments are checked; a missing n is
taken to be 0.
*/ARGSUSED*/
  turns on the -v option for the next function.

*/LINTLIBRARY*/
  at the beginning of a file shuts off complaints about unused
  functions in this file.

`Lint` produces its first output on a per source file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

**FILES**

```
/usr/lib/lint[12]  programs
/usr/lib/llib-1c.ln  declarations for standard functions (binary format;  
                     source is in /usr/lib/l1ib-1c)
/usr/lib/llib-port.ln declarations for portable functions (binary format;  
                        source is in /usr/lib/l1ib-port)
/usr/tmp/*lint*     temporary
```

**SEE ALSO**

`cc(1)`.

**BUGS**

`Exit(2)` and other functions which do not return are not understood; this causes various lies.
NAME

login — sign on

DESCRIPTION

The login command is used at the beginning of each terminal session and allows you to identify yourself to the system. It can no longer be invoked explicitly, but is invoked by the system when a connection is first established, or after the previous user has logged out by sending an "end-of-file" (control-D) to his or her initial shell. (See How to Get Started at the beginning of this volume for instructions on how to dial up initially.)

Login asks for your user name, 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.

At some installations, an option may be invoked that will require you to enter a second "external" password. This will occur only for dial-up connections, and will be prompted by the message "External security". Both passwords are required for a successful login.

If password aging has been invoked by the super-user on your behalf, your password may have expired. In this case, you will be shunted into passwd(1) to change it, after which you may attempt to login again.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, you will be informed of the existence (if any) of mail, and the profiles (i.e., /etc/profile and $HOME/.profile) (if any) are executed (see profile(5)). Login initializes the user and group IDs and the working directory, then executes a command interpreter (usually sh(1)) according to specifications found in the /etc/passwd file. Argument 0 of the command interpreter is — followed by the last component of the interpreter's path name. The environment (see environ(7)) is initialized to:

```
HOME=your-login-directory
PATH=:/bin:/usr/bin
LOGNAME=your-login-name
```

FILES

/etc/utmp accounting
/usr/adm/wtmp accounting
/usr/mail/your-name mailbox for user your-name
/etc/motd message-of-the-day
/etc/passwd password file
/etc/profile system profile
$HOME/.profile personal profile

SEE ALSO

mail(1), newgrp(1), sh(1), passwd(1), su(1), passwd(5), profile(5), environ(7), getty(8).

DIAGNOSTICS

Login incorrect
    if the user name or the password is incorrect.

No shell, cannot open password file, no directory:
    consult a UNIX programming counselor.

Your password has expired. Choose a new one.
    if password aging is implemented.
NAME
	nologname — get login name

SYNOPSIS

   logname

DESCRIPTION

   Logname returns the contents of the environment variable $LOGNAME, which is set when a user logs into the system.

FILES

   /etc/profile

SEE ALSO

   env(1), login(1), logname(3X), environ(7).
NAME
lorder — find ordering relation for an object library

SYNOPSIS
lorder file ...

DESCRIPTION
The input is one or more object or library archive files (see ar(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by tsort(1) to find an ordering of a library suitable for one-pass access by ld(1).

This brash one-liner intends to build a new library from existing .o files.

ar cr library `lorder *.o | tsort`

FILES
*symref, *symdef temp files

SEE ALSO
ar(1), ld(1), tsort(1).

BUGS
Object files whose name do not end with .o, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.
NAME
lpr — line printer spooler

SYNOPSIS
lpr [ option ... ] [ name ... ]

DESCRIPTION
Lpr causes the named files to be queued for printing on a line printer. If no names appear, the standard input is assumed; thus lpr may be used as a filter.

The following options may be given (each as a separate argument and in any order) before any file name arguments:
- c Makes a copy of the file to be sent before returning to the user.
- r Removes the file after sending it.
- m When printing is complete, reports that fact by mail(1).
- n Does not report the completion of printing by mail(1). This is the default option.

FILES
/etc/passwd user's identification and accounting data.
/usr/lib/ldp line printer daemon.
/usr/spool/ldp/* spool area.

SEE ALSO
dpd(1C), dpr(1C), lpd(1C).
NAME
ls — list contents of directories

SYNOPSIS
ls [ -logtasdrucif ] names

DESCRIPTION
For each directory named, ls lists the contents of that directory; for each file named, ls repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents. There are several options:

-1 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 will contain the major and minor device numbers, rather than a size.
-o The same as -1, except that the group is not printed.
-g The same as -1, except that the owner is not printed.
-t Sort by time of last modification (latest first) instead of by name.
-a List all entries; in the absence of this option, entries whose names begin with a period (.) are not listed.
-s Give size in blocks (including indirect blocks) for each entry.
-d If argument is a directory, list only its name; often used with -1 to get the status of a directory.
-r Reverse the order of sort to get reverse alphabetic or oldest first, as appropriate.
-u Use time of last access instead of last modification for sorting (with the -t option) and/or printing (with the -1 option).
-c Use time of last modification of the inode (mode, etc.) instead of last modification of the file for sorting (-t) and/or printing (-1).
-i For each file, print the i-number in the first column of the report.
-f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off -1, -t, -s, and -r, and turns on -a; the order is the order in which entries appear in the directory.

The mode printed under the -1 option consists of 11 characters that are interpreted as follows:

The first character is:

- d if the entry is a directory;
- b if the entry is a block special file;
- c if the entry is a character special file;
- p if the entry is a fifo (a.k.a. "named pipe") special file;
- - if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.
The permissions are indicated as follows:

- r if the file is readable;
- w if the file is writable;
- x if the file is executable;
- — if the indicated permission is not granted.

The group-execute permission character is given as s if the file has set-group-ID mode; likewise, the user-execute permission character is given as s if the file has set-user-ID mode. The last character of the mode (normally x or —) is t if the 1000 (octal) bit of the mode is on; see chmod(1) for the meaning of this mode. The indications of set-ID and 1000 bit of the mode are capitalized if the corresponding execute permission is not set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES
/etc/passwd to get user IDs for ls -l and ls -o.
/etc/group to get group IDs for ls -l and ls -g.

SEE ALSO
chmod(1), find(1).
NAME
m4 — macro processor

SYNOPSIS
m4 [ options ] [ files ]

DESCRIPTION
M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:
- e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.
- s Enable line sync output for the C preprocessor ( # line ... )
- B int Change the size of the push-back and argument collection buffers from the default of 4,096.
- H int Change the size of the symbol table hash array from the default of 199. The size should be prime.
- S int Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
- T int Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any -D or -U flags:
- D name [=val]
  Defines name to val or to null in val’s absence.
- U name
  undefines name.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The ( must immediately follow the name of the macro. If a defined macro name is not followed by a (, it is deemed to have no arguments. Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $n in the replacement text, where n is a digit, is replaced by the n-
th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; $@5$ is replaced by the number of arguments; $@6$ is replaced by a list of all the arguments separated by commas; $@7$ is like $@5$, but each argument is quoted (with the current quotes).

**undefine** removes the definition of the macro named in its argument.

**defn** returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

**pushdef** like **define**, but saves any previous definition.

**popdef** removes current definition of its argument(s), exposing the previous one if any.

**ifdef** if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word **unix** is predefined on UNIX versions of **m4**.

**shift** returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

**changequote** change quote symbols to the first and second arguments. The symbols may be up to five characters long. Changequote without arguments restores the original values (i.e., ".").

**changecom** change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes newline. With two arguments, both markers are affected. Comment markers may be up to five characters long.

**divert** **m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The **divert** macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

**undivert** causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

**divnum** returns the value of the current output stream.

**dnl** reads and discards characters up to and including the next new-line.

**ifelse** has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

**incr** returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
returns the value of its argument decremented by 1.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is identical to include, except that it says nothing if the file is inaccessible.

syscmd executes the UNIX command given in the first argument. No value is returned.

sysval is the return code from the last call to syscmd.

maketemp fills in a string of XXXXX in its argument with the current process ID.

m4exit causes immediate exit from m4. Argument 1, if given, is the exit code; the default is 0.

m4wrap argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup(`)

erprint prints its argument on the diagnostic output file.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.

traceon with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff turns off trace globally and for any macros specified. Macros specifically traced by traceon can be untraced only by specific calls to traceoff.

SEE ALSO
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.
NAME
mail, rmail — send mail to users or read mail

SYNOPSIS
mail [ -rpq ] [ -f file ]
m]{persons
rmail persons

DESCRIPTION
Mail without arguments prints a user’s mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input to determine the disposition of the message:

<new-line> Go on to next message.
+ Same as <new-line>.
d Delete message and go on to next message.
p Print message again.
- Go back to previous message.
s [ files ] Save message in the named files (mbox is default).
w [ files ] Save message, without its header, in the named files (mbox is default).
m [ persons ] Mail the message to the named persons (yourself is default).
q Put undeleted mail back in the mailfile and stop.
EOT (control-d) Same as q.
x Put all mail back in the mailfile unchanged and stop.
!command Escape to the shell to do command.
* Print a command summary.

The optional arguments alter the printing of the mail:

-r causes messages to be printed in first-in, first-out order.
-p causes all mail to be printed without prompting for disposition.
-q causes mail to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.
-fjile causes mail to use file (e.g., mbox) instead of the default mailfile.

When persons are named, mail takes the standard input up to an end-of-file (or up to a line consisting of just a ) and adds it to each person’s mailfile. The message is preceded by the sender’s name and a postmark. Lines that look like postmarks in the message, (i.e., “From ...”) are preceded with a >. A person is usually a user name recognized by login(1). If a person being sent mail is not recognized, or if mail is interrupted during input, the dead.letter will be saved to allow editing and resending.

To denote a recipient on a remote system, prefix person by the system name and exclamation mark (see uucp(1)). Everything after the first exclamation mark in persons is interpreted by the remote system. In particular, if persons contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying albled as a recipient’s name causes the message to be sent to user blede on system a. System a will interpret that destination as a request to send the message to user ede on system b. This might be useful, for instance, if the sending system can access system a but not system b, and system a has access to system b.

The mailfile may be manipulated in two ways to alter the function of mail. The other permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than
the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to person

which will cause all mail sent to the owner of the mailfile to be forwarded to person. This is especially useful to forward all of a person’s mail to one machine in a multiple machine environment.

Rmail only permits the sending of mail; uucp(1C) uses rmail as a security precaution.

When a user logs in he is informed of the presence of mail, if any.

FILES
/etc/passwd    to identify sender and locate persons
/usr/mail/*    incoming mail for user *; mailfile
$HOME/mbox     saved mail
SMAIL          mailfile
/tmp/mail*     temporary file
/usr/mail/*.lock lock for mail directory
dead.letter   unmailable text

SEE ALSO
login(1), uucp(1C), write(1).

BUGS
Race 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
make — maintain, update, and regenerate groups of programs

SYNOPSIS
[-m] [-t] [-q] [-d] [names]

DESCRIPTION
The following is a brief description of all options and some special names:

-\f makefile  Description file name. Makefile is assumed to be the name of
a description file. A file name of - denotes the standard
input. The contents of makefile override the built-in rules if
they are present.

-p  Print out the complete set of macro definitions and target
descriptions.

-i  Ignore error codes returned by invoked commands. This
mode is entered if the fake target name .IGNORE appears in
the description file.

-k  Abandon work on the current entry, but continue on other
branches that do not depend on that entry.

-s  Silent mode. Do not print command lines before executing.
This mode is also entered if the fake target name .SILENT
appears in the description file.

-r  Do not use the built-in rules.

-n  No execute mode. Print commands, but do not execute
them. Even lines beginning with an @ are printed.

-b  Compatibility mode for old makefiles.

-e  Environment variables override assignments within makefiles.

-m  Print a memory map showing text, data, and stack. This
option is a no-operation on systems without the getu system
call.

-t  Touch the target files (causing them to be up-to-date) rather
than issue the usual commands.

-d  Debug mode. Print out detailed information on files and
times examined.

-q  Question. The make command returns a zero or non-zero
status code depending on whether the target file is or is not
up-to-date.

.DEFAULT If a file must be made but there are no explicit commands or
relevant built-in rules, the commands associated with the
name .DEFAULT are used if it exists.

.PRECIOUS Dependents of this target will not be removed when quit or
interrupt are hit.

.SILENT Same effect as the -s option.

.IGNORE Same effect as the -i option.

Make executes commands in makefile to update one or more target names.
Name is typically a program. If no -f option is present, makefile,
Makefile, s.makefile, and s. Makefile are tried in order. If makefile is -,
the standard input is taken. More than one -f makefile argument pair
may appear.
Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a ;, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or # begins a new dependency or macro definition. Shell commands may be continued across lines with the \< Backslash> \< New-line> sequence. Sharp (#) and new-line surround comments.

The following makefile says that pgm depends on two files a.o and b.o, and that they in turn depend on their corresponding source files (a.c and b.c) and a common file incl.h:

```
pgm: a.o b.o
    cc a.o b.o -o pgm
a.o: incl.h a.c
    cc -c a.c
b.o: incl.h b.c
    cc -c b.c
```

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the -s option is present, or the entry .SILENT: is in makefile, or unless the first character of the command is @. The -n option specifies printing without execution; however, if the command line has the string $(MAKE) in it, the line is always executed (see discussion of the MAKEFLAGS macro under Environment). The -t (touch) option updates the modified date of a file without executing any commands. Commands returning non-zero status normally terminate make. If the -i option is present, or the entry .IGNORE: appears in makefile, or if the line specifying the command begins with <tab><hyphen>, the error is ignored. If the -k option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The -b option allows old makefiles (those written for the old version of make) to run without errors. The difference between the old version of make and this version is that this version requires all dependency lines to have a (possibly null) command associated with them. The previous version of make assumed if no command was specified explicitly that the command was null.

Interrupt and quit cause the target to be deleted unless the target depends on the special name .PRECIOUS.

Environment

The environment is read by make. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The -e option causes the environment to override the macro assignments in a makefile.

The MAKEFLAGS environment variable is processed by make as containing any legal input option (except -f, -p, and -d) defined for the command line. Further, upon invocation, make “invents” the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, MAKEFLAGS always contains the current input options. This proves very useful for “super-makes”. In fact, as noted above, when the -n option is used, the command $(MAKE) is
executed anyway; hence, one can perform a `make -n` recursively on a whole software system to see what would have been executed. This is because the `-n` is put in MAKEFLAGS and passed to further invocations of `$MAKE`). This is one way of debugging all of the makefiles for a software project without actually doing anything.

**Macros**

Entries of the form `string1 = string2` are macro definitions. Subsequent appearances of `$string1[subst1 = subst2]` are replaced by `string2`. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional `:subst1 = subst2` is a substitute sequence. If it is specified, all non-overlapping occurrences of `subst1` in the named macro are replaced by `subst2`. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under Libraries.

**Internal Macros**

There are five internally maintained macros which are useful for writing rules for building targets.

- **$** The macro `$` stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.
- **$@** The `$@` macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.
- **$<** The `$<` macro is only evaluated for inference rules or the `.DEFAULT` rule. It is the module which is out of date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the `.c.o` rule, the `$<` macro would evaluate to the `.c` file. An example for making optimized `.o` files from `.c` files is:

  ```
  .c.o:
  cc -c -O $*.c
  or:
  .c.o:
  cc -c -O $<
  ```

- **$?** The `$?` macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.
- **$%** The `$%` macro is only evaluated when the target is an archive library member of the form `lib(file.o)`. In this case, `$@` evaluates to `lib` and `$%` evaluates to the library member, `file.o`.

Four of the five macros can have alternative forms. When an upper case `D` or `F` is appended to any of the four macros the meaning is changed to "directory part" for `D` and "file part" for `F`. Thus, `$(@D)` refers to the directory part of the string `$@`. If there is no directory part, The only macro excluded from this alternative form is `$?`. The reasons for this are debatable.

**Suffixes**

Certain names (for instance, those ending with `.o`) have inferable prerequisites such as `.c`, `.s`, etc. If no update commands for such a file appear in `makefile`, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, `make` has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:
The internal rules for make are contained in the source file rules.c for the make program. These rules can be locally modified. To print out the rules compiled into the make on any machine in a form suitable for recompilation, the following command is used:

```
makesh -fp -z > /dev/null < /dev/null
```

The only peculiarity in this output is the (null) string which printf(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see sccsfile(5)). Thus, the rule .c.o would transform an SCCS C source file into an object file (.o). Because the s. of the SCCS files is a prefix it is incompatible with make's suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e. c:) is the definition of how to build x from x.c. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for .SUFFIXES. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite.

The default list is:

```
.SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; .SUFFIXES: with no dependencies clears the list of suffixes.

**Inference Rules**

The first example can be done more briefly:

```
pgm: a.o b.o
c: a.o b.o -0 pgm
a.o b.o: incl.h
```

This is because make has a set of internal rules for building files. The user may add rules to this list by simply putting them in the makefile.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, CFLAGS, LFLAGS, and YFLAGS are used for compiler options to cc(1), lex(1), and yacc(1) respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix .o from a file with suffix .c is specified as an entry with .c.o: as the target and no dependents. Shell commands associated with the target define the rule for making a .o file from a .c file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

**Libraries**

If a target or dependency name contains parenthesis, it is assumed to be an archive library, the string within parenthesis referring to a member within the library. Thus lib(file.o) and $(LIB)(file.o) both refer to an archive library which contains file.o. (This assumes the LIB macro has been previously defined.) The expression $(LIB)(file1.o file2.o) is not legal. Rules pertaining to archive libraries have the form .XX.a where the XX is the
suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the \texttt{XX} to be different from the suffix of the archive member. Thus, one cannot have \texttt{lib(file.o)} depend upon \texttt{file.o} explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

\begin{verbatim}
lib: lib(file1.o) lib(file2.o) lib(file3.o)
   @echo lib is now up to date
.c.a:
   $(CC) -c $(CFLAGS) $<
   ar rv $(@) $*.o
   rm -f $*.o
\end{verbatim}

In fact, the \texttt{.c.a} rule listed above is built into \texttt{make} and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

\begin{verbatim}
lib: lib(file1.o) lib(file2.o) lib(file3.o)
   $(CC) -c $(CFLAGS) $(?..o=.c)
   ar rv lib$(?)
   rm $(?)
   @echo lib is now up to date
.c.a::;
\end{verbatim}

Here the substitution mode of the macro expansions is used. The \texttt{$?} list is defined to be the set of object file names (inside \texttt{lib}) whose C source files are out of date. The substitution mode translates the \texttt{.o} to \texttt{.c}. (Unfortunately, one cannot as yet transform to \texttt{.c}; however, this may become possible in the future.) Note also, the disabling of the \texttt{.c.a:} rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

### FILES

[\texttt{Mm}]akefile

s.[\texttt{Mm}]akefile

### SEE ALSO

\texttt{sh(1)}, \texttt{mk(8)}.

*Make—A Program for Maintaining Computer Programs* by S. I. Feldman.

*An Augmented Version of Make* by E. G. Bradford.

### BUGS

Some commands return non-zero status inappropriately; use \texttt{-i} to overcome the difficulty. Commands that are directly executed by the shell, notably \texttt{cd(1)}, are ineffectual across new-lines in \texttt{make}. The syntax (\texttt{lib(file1.o file2.o file3.o)}) is illegal. You cannot build \texttt{lib(file.o)} from \texttt{file.o}. The macro \texttt{$(a:.o=.c)$} doesn't work.
NAME

man — print entries in this manual

SYNOPSIS

man [ options ] [ section ] titles

DESCRIPTION

Man locates and prints the entry of this manual named title in the specified section. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The title is entered in lower case. The section number may not have a letter suffix. If no section is specified, the whole manual is searched for title and all occurrences of it are printed. Options and their meanings are:

- t Typeset the entry in the default format (8.5"×11").
- s Typeset the entry in the small format (6"×9").
- T4014 Display the typeset output on a Tektronix 4014 terminal using tc(1).
- Ttek Same as -T4014.
- Tst Print the typeset output on the MHCC STARE facility (see gccat(1C)).
- Tvp Print the typeset output on a Versatec printer using vpr(1); this option is not available at all UNIX sites.
- Tterm Format the entry using nroff(1) and print it on the standard output (usually, the terminal); term is the terminal type (see term(7) and the explanation below); for a list of recognized values of term, type help term2. The default value of term is 450.
- w Print on the standard output only the path names of the entries, relative to /usr/man, or to the current directory for -d option.
- d Search the current directory rather than /usr/man; requires the full file name (e.g., cu.1c, rather than just cu).
- 12 Indicates that the manual entry is to be produced in 12-pitch. May be used when STERM (see below) is set to one of 300, 300s, 450, and 1620. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 11 if this option is used.)
- c Causes man to invoke col(1); note that col(1) is invoked automatically by man unless term is one of 300, 300s, 450, 37, 4000A, 382, 4014, tek, 1620, and X.
- y Causes man to use the non-compacted version of the macros.

The above options are mutually exclusive, except that the -s option may be used in conjunction with the first four -T options above. Any other options are passed to troff(1), nroff(1), or the man(7) macro package.

When using nroff(1), man examines the environment variable STERM (see environ(7)) and attempts to select options to nroff(1), as well as filters, that adapt the output to the terminal being used. The -Tterm option overrides the value of STERM; in particular, one should use -T1p when sending the output of man to a line printer.

Section may be changed before each title.

As an example:

man man

would reproduce on the terminal this entry, as well as any other entries named man that may exist in other sections of the manual, e.g., man(7).
If the first line of the input for an entry consists solely of the string:

`\* x`

where `x` is any combination of the three characters c, e, and t, and where there is exactly one blank between the double quote (`*`) and `x`, then `man` will preprocess its input through the appropriate combination of `cw(1)`, `eqn(1)` or `neqn(1)`, and `tbl(1)`, respectively.

FILES

`/usr/man/man[1-8]/*`
`/usr/man/local/man[1-8]/*`

SEE ALSO

`cw(1)`, `eqn(1)`, `gcat(1c)`, `tbl(1)`, `tc(1)`, `troff(1)`, `environ(7)`, `man(7)`, `term(7)`.

BUGS

All entries are supposed to be reproducible either on a typesetter or on a terminal. However, on a terminal some information is necessarily lost.
NAME
   mesg — permit or deny messages

SYNOPSIS
   mesg [ n ] [ y ]

DESCRIPTION
   Mesg with argument n forbids messages via write(1) by revoking non-user write permission on the user’s terminal. Mesg with argument y reinstates permission. All by itself, mesg reports the current state without changing it.

FILES
   /dev/tty*

SEE ALSO
   write(1).

DIAGNOSTICS
   Exit status is 0 if messages are receivable, 1 if not, 2 on error.
NAME
  mkdir — make a directory

SYNOPSIS
  mkdir dirname ...

DESCRIPTION
  Mkdir creates specified directories in mode 777. Standard entries, .., for the
directory itself, and ..., for its parent, are made automatically.

  Mkdir requires write permission in the parent directory.

SEE ALSO
  rm(1).

DIAGNOSTICS
  Mkdir returns exit code 0 if all directories were successfully made; oth-
erwise, it prints a diagnostic and returns non-zero.
NAME
mkfs — construct a file system

SYNOPSIS
/etc/mkfs special blocks[:inodes] [gap blocks]
/etc/mkfs special proto [gap blocks]

DESCRIPTION
Mkfs constructs a file system by writing on the special file according to the
directions found in the remainder of the command line. If the second
argument is given as a string of digits, mkfs builds a file system with a
single empty directory on it. The size of the file system is the value of
blocks interpreted as a decimal number. The boot program is left uninitialize-
d. If the optional number of inodes is not given, the default is the num-
ber of blocks divided by 4.

If the second argument is a file name that can be opened, mkfs assumes it
to be a prototype file proto, and will take its directions from that file. The
prototype file contains tokens separated by spaces or new-lines. The first
token is the name of a file to be copied onto block zero as the bootstrap
program (see unixboot(8)). The second token is a number specifying the
size of the created file system. Typically it will be the number of blocks on
the device, perhaps diminished by space for swapping. The next token is
the i-list size in blocks (remember there are eight i-nodes per block). The
next set of tokens comprise the specification for the root file. File
specifications consist of tokens giving the mode, the user ID, the group ID,
and the initial contents of the file. The syntax of the contents field depends
on the mode.

The mode token for a file is a 6 character string. The first character
specifies the type of the file. (The characters -bca 1 specify regular, block
special, character special and directory files respectively.) The second
character of the type is either u or - to specify set-user-id mode or not.
The third is g or - for the set-group-id mode. The rest of the mode is a
three digit octal number giving the owner, group, and other read, write,
execute permissions (see chmod(1)).

Two decimal number tokens come after the mode; they specify the user
and group ID's of the owner of the file.

If the file is a regular file, the next token is a path name whence the con-
tents and size are copied. If the file is a block or character special file, two
decimal number tokens follow which give the major and minor device num-
bers. If the file is a directory, mkfs makes the entries . and .. and then
reads a list of names and (recursively) file specifications for the entries in
the directory. The scan is terminated with the token $.

A sample prototype specification follows:

    /stand/diskboot
    4872 110
    d -- 777 3 1
    usr d -- 777 3 1
         sh --- 755 3 1 /bin/sh
         ken d -- 755 6 1
         $ b0 b -- 644 3 1 0 0
c0 c -- 644 3 1 0 0 $
In both command syntaxes, the rotational \textit{gap} and the number of \textit{blocks} can be specified. For RP04 type drives, these numbers should be 7 and 418.

\textbf{SEE ALSO}

dir(5), fs(5), unixboot(8).

\textbf{BUGS}

If a prototype is used, it is not possible to initialize a file larger than 64K bytes, nor is there a way to specify links.
NAME
mknod — build special file

SYNOPSIS
/etc/mknod name [ c ] [ b ] major minor
/etc/mknod name p

DESCRIPTION
Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the name of the entry. In the first case, the second is b if the special file is block-type (disks, tape) or c if it is character-type (other devices). The last two arguments are numbers specifying the major device type and the minor device (e.g. unit, drive, or line number), which may be either decimal or octal.

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file conf.c.

Mknod can also be used to create fifo's (a.k.a named pipes) (second case in SYNOPSIS above).

SEE ALSO
mknod(2).
NAME
mm – print out documents formatted with the MM macros

SYNOPSIS
mm [ options ] [ files ]

DESCRIPTION
Mm can be used to type out documents using nroff(1) and the MM text-formatting macro package. It has options to specify preprocessing by tbl(1) and/or neqn(1) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for nroff(1) and MM arc generated, depending on the options selected.

Options for mm are given below. Any other arguments or flags (e.g., -rC3) are passed to nroff(1) or to MM, as appropriate. Such options can occur in any order, but they must appear before the files arguments. If no arguments are given, mm prints a list of its options.

-Tterm Specifies the type of output terminal; for a list of recognized values for term, type help term2. If this option is not used, mm will use the value of the shell variable STERM from the environment (see profile(5) and environ(7)) as the value of term, if STERM is set; otherwise, mm will use 450 as the value of term. If several terminal types are specified, the last one takes precedence.

-12 Indicates that the document is to be produced in 12-pitch. May be used when STERM is set to one of 300, 300s, 450, and 1620. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 12 if this option is used.)

-c Causes mm to invoke col(1); note that col(1) is invoked automatically by mm unless term is one of 300, 300s, 450, 37, 4000A, 382, 4014, tek, 1620, and X.

e Causes mm to invoke neqn(1).

-t Causes mm to invoke tbl(1).

-E Invokes the -e option of nroff(1).

-y Causes mm to use the non-compacted version of the macros (see mm(7)).

As an example (assuming that the shell variable STERM is set in the environment to 450), the two command lines below are equivalent:

```
mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3
```

Mm reads the standard input when - is specified instead of any file names. (Mentioning other files together with - leads to disaster.) This option allows mm to be used as a filter, e.g.:

```
cat dws | mm -
```

HINTS
1. Mm invokes nroff(1) with the -h flag. With this flag, nroff(1) assumes that the terminal has tabs set every 8 character positions.

2. Use the -olist option of nroff(1) to specify ranges of pages to be output. Note, however, that mm, if invoked with one or more of the -e, -t, and - options, together with the -olist option of nroff(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in list.

3. If you use the -s option of nroff(1) (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The -s option of nroff(1) does not work with the -c option of mm, or if mm automatically invokes col(1) (see -c
option above).

4. If you lie to \texttt{mm} about the kind of terminal its output will be printed on, you’ll get (often subtle) garbage; however, if you are redirecting output into a file, use the \texttt{--T37} option, and then use the appropriate terminal filter when you actually print that file.

\textbf{SEE ALSO}

\texttt{col(1), env(1), eqn(1), greek(1), mmt(1), nroff(1), tbl(1), profile(5), mm(7), term(7)}.

\textit{MM—Memorandum Macros} by D. W. Smith and J. R. Mashey.

\textit{Typing Documents with MM} by D. W. Smith and E. M. Piskorik.

\textbf{DIAGNOSTICS}

“\texttt{mm: no input file}” if none of the arguments is a readable file and \texttt{mm} is not used as a filter.
NAME
  mmchek - check usage of mm macros and eqn delimiters

SYNOPSIS
  mmchek [files]

DESCRIPTION
  Mmchek is a program for checking the contents of the named files for errors
  in the use of Memorandum Macros (see mm(1) and some eqn(1) constructions.
  Appropriate messages are produced. The program skips all directories,
  and if no file name is given, standard input is read.

SEE ALSO
  eqn(1), mm(1), mmt(1).
  MM—Memorandum Macros by D. W. Smith and J. R. Mashey.

DIAGNOSTICS
  Unreadable files cause the message "Cannot open file-name". The remaining
  output of the program is diagnostic of the source file.

BUGS
  This is an experimental version of mmchek. Mmchek may be fully supported
  in the future.
NAME
mmt, mvt — typeset documents, view graphs, and slides

SYNOPSIS
mmt [ options ] [ files ]
mvt [ options ] [ files ]

DESCRIPTION
These two commands are very similar to mm(1), except that they both typeset their input via troff(1), as opposed to formatting it via nroff(1); mmt uses the MM macro package, while mvt uses the Macro Package for View Graphs and Slides. These two commands have options to specify pre-processing by tbl(1) and/or eqn(1). The proper pipelines and the required arguments and flags for troff(1) and for the macro packages are generated, depending on the options selected.

Options are given below. Any other arguments or flags (e.g., -rC3) are passed to troff(1) or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the files arguments. If no arguments are given, these commands print a list of their options.

-e Causes these commands to invoke eqn(1).
-t Causes these commands to invoke tbl(1).
-Tst Directs the output to the MH STARE facility.
-Typ Directs the output to a Versatec printer via the vpr(1) spooler; this option is not available at all UNIX sites.
-T4014 Directs the output to a Tektronix 4014 terminal via the tc(1) filter.
-Ttek Same as -T4014.
-a Invokes the -a option of troff(1).
-y Causes mmt to use the non-compacted version of the macros (see mm(7)). No effect for mvt.

These commands read the standard input when - is specified instead of any file names.

Mvt is just a link to mmt.

HINT
Use the -olist option of troff(1) to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the -e, -t, and - options, together with the -olist option of troff(1) may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in list.

SEE ALSO
env(1), eqn(1), mm(1), tbl(1), tc(1), troff(1), profile(5), environ(7), mm(7), mv(7).

MM—Memorandum Macros by D. W. Smith and J. R. Mashey.
Typing Documents with MM by D. W. Smith and E. M. Piskorik.

DIAGNOSTICS
"m[mv]: no input file" if none of the arguments is a readable file and the command is not used as a filter.
NAME
mount, umount — mount and dismount file system

SYNOPSIS
/etc/mount [ special directory [ -r ] ]
/etc/umount special

DESCRIPTION
Mount announces to the system that a removable file system is present on the
device special. The directory must exist already; it becomes the name of the
root of the newly mounted file system.

These commands maintain a table of mounted devices. If invoked with no
arguments, mount prints the table.

The optional last argument indicates that the file is to be mounted read­
only. Physically write-protected and magnetic tape file systems must be
mounted in this way or errors will occur when access times are updated,
whether or not any explicit write is attempted.

Umount announces to the system that the removable file system previously
mounted on device special is to be removed.

FILES
/etc/mnttab  mount table

SEE ALSO
mount(2), mnttab(5).

DIAGNOSTICS
Mount issues a warning if the file system to be mounted is currently moun­
ted under another name.

Umount complains if the special file is not mounted or if it is busy. The file
system is busy if it contains an open file or some user's working directory.

BUGS
Some degree of validation is done on the file system, however it is gen­
erally unwise to mount garbage file systems.
NAME
mvdir — move a directory

SYNOPSIS
/etc/mvdir dirname  name

DESCRIPTION
Mvdir renames directories within a file system. Dirname must be a directory; name must not exist. Neither name may be a sub-set of the other (/x/y cannot be moved to /x/y/z, nor vice versa).

Only super-user can use mvdir.

SEE ALSO
mkdir(1).
NAME

ncheck — generate names from i-numbers

SYNOPSIS

ncheck [ -i numbers ] [ -a ] [ -s ] [ file-system ]

DESCRIPTION

Ncheck with no argument generates a path name vs. i-number list of all files on a set of default file systems. Names of directory files are followed by ./.. The -i option reduces the report to only those files whose i-numbers follow. The -a option allows printing of the names , and ..., which are ordinarily suppressed, suppressed. The -s option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

When the file system structure is improper, ?? denotes the “parent” of a parentless file and a path name beginning with ... denotes a loop.
NAME
newgrp — log in to a new group

SYNOPSIS
newgrp [ group ]

DESCRIPTION
Newgrp changes the group identification of its caller, analogously to login(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

Newgrp without an argument changes the group identification to the group in the password file; in effect it changes the group identification back to the caller’s original group.

A password is demanded if the group has a password and the user himself does not, or if the group has a password and the user is not listed in /etc/group as being a member of that group.

When most users log in, they are members of the group named other.

FILES
/etc/group
/etc/passwd

SEE ALSO
login(1), group(5).

BUGS
There is no convenient way to enter a password into /etc/group.
Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.
NAME
news - print news items

SYNOPSIS
news [ -a ] [ -n ] [ -s ] [ items ]

DESCRIPTION
News is used to keep the user informed of current events. By convention, these events are described by files in the directory /usr/news.

When invoked without arguments, news prints the contents of all current files in /usr/news, most recent first, with each preceded by an appropriate header. News stores the "currency" time as the modification date of a file named .news_time in the user's home directory (the identity of this directory is determined by the environment variable $HOME); only files more recent than this currency time are considered "current."

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 one's .profile file, or in the system's /etc/profile.

All other arguments are assumed to be specific news items that are to be printed.

If a delete is typed during the printing of a news item, printing stops and the next item is started. Another delete within one second of the first causes the program to terminate.

FILES
/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO
profile(5), environ(7).
NAME
nice — run a command at low priority

SYNOPSIS
nice [ -increment ] command [ arguments ]

DESCRIPTION
Nice executes command with a lower CPU scheduling priority. If the increment argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., --10.

SEE ALSO
nohup(1), nice(2).

DIAGNOSTICS
Nice returns the exit status of the subject command.

BUGS
An increment larger than 19 is equivalent to 19.
NAME
nl — line numbering filter

SYNOPSIS
[-ssep] [-wwidth] [-nformat] 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 num­
bering options are independently available for header, body, and footer
(e.g. no numbering of header and footer lines while numbering blank lines
only in the body).
The start of logical page sections are signaled by input lines containing
nothing but the following character(s):

<table>
<thead>
<tr>
<th>Line contents</th>
<th>Start of</th>
</tr>
</thead>
</table>
| \\
| header       |
| \\:          | body     |
| \\:          | footer   |

Unless signaled otherwise, nl assumes the text being read is in a single logi­
cal 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:

- btype Specifies which logical page body lines are to be numbered.
  Recognized types and their meaning are: a, number all lines; t, number
  lines with printable text only; n, no line numbering; pstring, number
  only lines that contain the regular expression specified in string. Default type for logical page body is t (text
  lines numbered).

- htype Same as -btype except for header. Default type for logical page
  header is n (no lines numbered).

- ftype Same as -btype except for footer. Default for logical page
  footer is n (no lines numbered).

- p Do not restart numbering at logical page delimiters.

- vstart# Start# is the initial value used to number logical page lines.
  Default is 1.

- iincr Incr is the increment value used to number logical page lines.
  Default is 1.

- ssep Sep is the character(s) used in separating the line number and
  the corresponding text line. Default sep is a tab.

- wwidth Width is the number of characters to be used for the line
  number. Default width is 6.

- nformat Format is the line numbering format. Recognized values are: lmn,
  left justified, leading zeroes supressed; rmn, right justified, leading
  zeroes supressed; rz, right justified, leading zeroes kept. Default
  format is rmn (right justified).
-1num

Num is the number of blank lines to be considered as one. For example, -12 results in only the second adjacent blank being numbered (if the appropriate -ha, -ba, and/or -fa option is set). Default is 1.

SEE ALSO

pr(1).

-2-
NAME

nm — print name list

SYNOPSIS

nm [ -gnoprsu ] [ file ... ]

DESCRIPTION

Nm prints the name list (symbol table) of each object file in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no file is given, the symbols in a.out are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters U (undefined), A (absolute), T (text segment symbol), D (data segment symbol), B (bss segment symbol), R (register symbol), F (file symbol), or C (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

- g  Print only global (external) symbols.
- n  Sort numerically rather than alphabetically.
- o  Prepend file or archive element name to each output line rather than only once. This option can be used to make piping to grep(1) more meaningful.
- p  Don’t sort; print in symbol-table order.
- r  Sort in reverse order.
- s  Sort according to the size of the external symbol (computed from the difference between the value of the symbol and the value of the symbol with the next highest value). This difference is the value printed. This flag turns on -g and -n and turns off -u and -p.
- u  Print only undefined symbols.

SEE ALSO

ar(1), a.out(5), ar(5).
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
re-directed by the user, it will be sent to nohup.out. If nohup.out is not
writable in the current directory, output is redirected to $HOME/nobup.out.

SEE ALSO
nice(1), signal(2).
NAME
od — octal dump

SYNOPSIS
od [ -bcdox ] [ file ] [ [ + ]offset[ . ]][ b ]

DESCRIPTION
Od dumps file in one or more formats as selected by the first argument. If
the first argument is missing, -o is default. The meanings of the format
options are:

-b Interpret bytes in octal.
-c Interpret bytes in ASCII. Certain non-graphic characters appear as C
escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
-d Interpret words in decimal.
-o Interpret words in octal.
-x Interpret words in hex.

The file argument specifies which file is to be dumped. If no file argument
is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to
commence. This argument is normally interpreted as octal bytes. If . is
appended, the offset is interpreted in decimal. If b is appended, the offset
is interpreted in blocks of 512 bytes. If the file argument is omitted, the
offset argument must be preceded by +.

Dumping continues until end-of-file.

SEE ALSO
adb(1).
NAME
rjesta.t - RJE status and enquiries

SYNOPSIS
rjestat [ - ] [ A ] [ B ] [ C ] [ U1 ] [ U2 ] [ U3 ]

DESCRIPTION
When invoked without the - argument, rjestat reports the current status of
RJE links to the specified host computers. When invoked with the -
argument, rjestat sets up an interactive status terminal. If no hosts are
cited explicitly, the specification defaults to all those for which a given
UNIX is configured. The "host" pseudonyms A, B, C, U1, U2, and U3 are
built into the RJE software. A, B, and C may be used to represent any IBM
host machine. Their actual destinations are immaterial to RJE. The pseudo-
onyms U1, U2, and U3 are built into RJE to represent any UNIVAC host.

To enter an enquiry via such a status terminal, you must first generate an
interrupt. This can be done by hitting the DEL key or the
BREAK/INTERRUPT key. Rjestat will respond by prompting for enquiries
directed to each host in turn. The line on which a prompt appears may be
completed to form a legitimate display command for that particular host. If
the line is terminated with a \, the prompt will be repeated, otherwise it will
advance to the next host. A carriage return alone indicates that no enquiry
is to be directed to a particular host. You should expect to wait at least 30
seconds for a response.

An interrupt will temporarily halt the display of responses. It can therefore
be used to inhibit roll-up on a CRT terminal. The display of responses will
resume after all prompts have been satisfied (perhaps by null completions).

To exit from the status terminal, generate a quit signal or type DEL fol-
lowed by EOT.

The interactive status enquiry capability is not supported for UNIVAC.

FILES
/dev/rje* DSQ-11's used by RJE
/usr/rje/sys PWB/UNIX system name
/usr/rje/lines configuration table

And, in the directory for each RJE subsystem:

log activity log
resp concatenated responses
status message of the day
xmit* files queued
*mesg enquiry slot
*init boot program

SEE ALSO
Guide to IBM Remote Job Entry for PWB/UNIX Users by A. L. Sabsevitz and
E. J. Finger.
Operator's Library: OS/VS2 Reference (JES2), IBM SRL # GC38-0210.
NAME
pack, pcat, unpack — compress and expand files

SYNOPSIS
pack [ - ] 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. 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 12 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 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what cat(1) does for ordinary files. The specified files are unpacked and written to the standard output. Thus to view a packed file named name.z use:

pcat name.z

or just:

pcat name

To make an unpacked copy, say nnn, of a packed file named name.z (without destroying name.z) use the command:

pcat name > nnn
Pcat returns the number of files it was unable to unpack. Failure may occur if:

- the file name (exclusive of the .z) has more than 12 characters;
- the file cannot be opened;
- the file does not appear to be the output of pack.

Unpack expands files created by pack. For each file name specified in the command, a search is made for a file called name.z (or just name, if name ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in pcat, as well as for the following:

- a file with the “unpacked” name already exists;
- if the unpacked file cannot be created.
NAME
passwd — change login password

SYNOPSIS
passwd name

DESCRIPTION
This command changes (or installs) a password associated with the login name.

The program prompts for the old password (if any) and then for the new one (twice). The caller must supply these. New passwords should be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. Only the first eight characters of the password are significant.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password. Only the super-user can create a null password.

The password file is not changed if the new password is the same as the old password, or if the password has not “aged” sufficiently; see passwd(5).

FILES
/etc/passwd

SEE ALSO
login(1), crypt(3C), passwd(5).
NAME
paste — merge same lines of several files or subsequent lines of one file

SYNOPSIS
paste file1 file2 ...
paste -d list file1 file2 ...
paste -s [ -d list ] file1 file2 ...

DESCRIPTION
In the first two forms, paste concatenates corresponding lines of the given input files file1, file2, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of cat(1) which concatenates vertically, i.e., one file after the other. In the last form above, paste subsumes 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 file name.

The meanings of the options are:

- d 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. This option allows replacing the tab character by one or more alternate characters (see below).

list One or more characters immediately following -d replace the default tab as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (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). Quoting may be necessary, if characters 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. Use tab for concatenation, unless a list is specified with -d option. Regardless of the list, the very last character of the file is forced to be a new-line.

- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES
ls | paste -d"* " list directory in one column
ls | paste - - - - list directory in four columns
paste -s -d"\t\n" file combination of pairs of lines into lines

SEE ALSO
grep(1), cut(1),
pr(1): pr -t -m... works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS
line too long Output lines are restricted to 511 characters.
too many files Except for -s option, no more than 12 input files may be specified.
NAME
pr — print files

SYNOPSIS
pr [ options ] [ files ]

DESCRIPTION
Pr prints the named files on the standard output. If file is -, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the -s option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until pr has completed printing.

Options may appear singly or be combined in any order. Their meanings are:

+k Begin printing with page k (default is 1).
-k Produce k-column output (default is 1). The options -e and -i are assumed for multi-column output.
-a Print multi-column output across the page.
-m Merge and print all files simultaneously, one per column (overrides the -k, and -a options).
-d Double-space the output.
-eck 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).
-ick 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).
-ack Provide k-digit line numbering (default for k is 5). The number occupies the first k + 1 character positions of each column of normal output or each line of -m output. If c (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for c is a tab).
-wk Set the width of a line to k character positions (default is 72 for equal-width multi-column output, no limit otherwise).
-ok Offset each line by k character positions (default is 0). The number of character positions per line is the sum of the width and offset.
-lk Set the length of a page to k lines (default is 66).
-h Use the next argument as the header to be printed instead of the file name.
-p Pause before beginning each page if the output is directed to a terminal (pr will ring the bell at the terminal and wait for a carriage
return).

- `f` Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

- `r` Print no diagnostic reports on failure to open files.

- `t` Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

- `sc` Separate columns by the single character `c` instead of by the appropriate number of spaces (default for `c` is a tab).

**EXAMPLES**

Print `file1` and `file2` as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write `file1` on `file2`, expanding tabs to columns 10, 19, 28, 37, ...:

```
pr -e9 -t <file1 >file2
```

**FILES**

```
/dev/tty*  to suspend messages
```

**SEE ALSO**

`cat(1)`.
NAME
prof — display profile data

SYNOPSIS
prof [ -v ] [ -a ] [ -l ] [ -low [ -high ] ] [ file ]

DESCRIPTION
Prof interprets the file mon.out produced by the monitor(3C) subroutine. Under default modes, the symbol table in the named object file (a.out default) is read and correlated with the mon.out profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the -a option is used, all symbols are reported rather than just external symbols. If the -l option is used, the output is listed by symbol value rather than decreasing percentage.

If the -v option is used, all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the tplot(1G) filters. The optional arguments low and high, by default 0 and 100, cause a selected percentage of the profile to be plotted with accordingly higher resolution.

In order for the number of calls to a routine to be tallied, the -p option of cc must have been given when the file containing the routine was compiled. This option also arranges for the mon.out file to be produced automatically.

FILES
mon.out for profile
a.out for namelist

SEE ALSO
cc(1), tplot(1G), profil(2), monitor(3C).

BUGS
Beware of quantization errors.
NAME  
prfld, prfstat, prfde, prfsnap, prfpr — operating system profiler

SYNOPSIS  
/etc/prfld [ namelist ]
/etc/prfstat [ on | off ]
/etc/prfde [ period | off_hour ]
/etc/prfsnap file
/etc/prfpr file [ cutoff | namelist ]

DESCRIPTION  
Prfld, prfstat, prfde, prfsnap, and prfpr form a system of programs to facilitate an activity study of the UNIX operating system.

Prfld is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from namelist.

Prfstat is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. Prfstat will also reveal the number of text addresses being measured.

Prfde and prfsnap perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. Prfde will store the counters into file every period minutes and will turn off at off_hour. Prfsnap collects data at the time of invocation only, appending the counter values to file.

Prfpr formats the data collected by prfde or prfsnap. Each text address is converted to the nearest text symbol (as found in namelist) and is printed if the percent activity for that range is greater than cutoff.

FILES  
/dev/prf interface to profile data and text addresses
/unix default for namelist file

SEE ALSO  
prf(4).
NAME
prs — print an SCCS file

SYNOPSIS
prs [-d[dataspec]] [-r[SID]] [-e] [-l] [-a] files

DESCRIPTION
Prs prints, on the standard output, parts or all of an SCCS file (see sccsfile(5)) in a user supplied format. If a directory is named, prs behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.), and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to prs, which may appear in any order, consist of keyletter arguments, and file names.

All the described keyletter arguments apply independently to each named file:

- d[dataspec] Used to specify the output data specification. The dataspec is a string consisting of SCCS file data keywords (see DATA KEYWORDS) interspersed with optional user supplied text.

- r[SID] Used to specify the SCCS IDentification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

- e Requests information for all deltas created earlier than and including the delta designated via the - r keyletter.

- l Requests information for all deltas created later than and including the delta designated via the - r keyletter.

- a Requests printing of information for both removed, i.e., delta type = R, (see rmdel(1)) and existing, i.e., delta type = D, deltas. If the - a keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS
Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see sccsfile(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a dataspec.

The information printed by prs consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the dataspec. The format of a data keyword value is either Simple (S), in which keyword substitution is direct, or Multi-line (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line is specified by \n.
### TABLE 1. SCCS Files Data Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Data Item</th>
<th>File Section</th>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Dt:</td>
<td>Delta information</td>
<td>Delta Table</td>
<td>See below*</td>
<td>S</td>
</tr>
<tr>
<td>:DL:</td>
<td>Delta line statistics</td>
<td>* :Li:/:Ld:/:Lu:</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Li:</td>
<td>Lines inserted by Delta</td>
<td>*</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Ld:</td>
<td>Lines deleted by Delta</td>
<td>*</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Lu:</td>
<td>Lines unchanged by Delta</td>
<td>*</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:DT:</td>
<td>Delta type</td>
<td>*</td>
<td>D or R</td>
<td>S</td>
</tr>
<tr>
<td>:Li:</td>
<td>SCCS ID string (SID)</td>
<td>* :R:/:Li:/:Bi:/:S:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:R:</td>
<td>Release number</td>
<td>*</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:L:</td>
<td>Level number</td>
<td>*</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:B:</td>
<td>Branch number</td>
<td>*</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:S:</td>
<td>Sequence number</td>
<td>*</td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:D:</td>
<td>Date Delta created</td>
<td>* :Dy:/:Dm:/:Dd:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:Dy:</td>
<td>Year Delta created</td>
<td>*</td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Dm:</td>
<td>Month Delta created</td>
<td>*</td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Dd:</td>
<td>Day Delta created</td>
<td>*</td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:T:</td>
<td>Time Delta created</td>
<td>* :Th:/:Tm:/:Ts:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:Th:</td>
<td>Hour Delta created</td>
<td>*</td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Tm:</td>
<td>Minutes Delta created</td>
<td>*</td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Ts:</td>
<td>Seconds Delta created</td>
<td>*</td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:P:</td>
<td>Programmer who created Delta</td>
<td>*</td>
<td>logname</td>
<td>S</td>
</tr>
<tr>
<td>:DS:</td>
<td>Delta sequence number</td>
<td>*</td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:DP:</td>
<td>Predecessor Delta seq-no.</td>
<td>*</td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Di:</td>
<td>Seq-no. of deltas incl., excl., ignored</td>
<td>* :Dn:/:Dx:/:Dg:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:Dn:</td>
<td>Deltas included (seq #)</td>
<td>* :DS:/:DS:...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:Dx:</td>
<td>Deltas excluded (seq #)</td>
<td>* :DS:/:DS:...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:Dg:</td>
<td>Deltas ignored (seq #)</td>
<td>* :DS:/:DS:...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:MR:</td>
<td>MR numbers for delta</td>
<td>*</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:UN:</td>
<td>User names</td>
<td>User Names</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:FL:</td>
<td>Flag list</td>
<td>Flags</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:Y:</td>
<td>Module type flag</td>
<td>*</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:MF:</td>
<td>MR validation flag</td>
<td>*</td>
<td>yes or no</td>
<td>S</td>
</tr>
<tr>
<td>:MP:</td>
<td>MR validation pgm name</td>
<td>*</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:KF:</td>
<td>Keyword error/warning flag</td>
<td>*</td>
<td>yes or no</td>
<td>S</td>
</tr>
<tr>
<td>:BF:</td>
<td>Branch flag</td>
<td>*</td>
<td>yes or no</td>
<td>S</td>
</tr>
<tr>
<td>:J:</td>
<td>Joint edit flag</td>
<td>*</td>
<td>yes or no</td>
<td>S</td>
</tr>
<tr>
<td>:LK:</td>
<td>Locked releases</td>
<td>* :R:...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>:Q:</td>
<td>User defined keyword</td>
<td>*</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:M:</td>
<td>Module name</td>
<td>*</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:FB:</td>
<td>Floor boundary</td>
<td>* :R:</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>:CB:</td>
<td>Ceiling boundary</td>
<td>* :R:</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>:Ds:</td>
<td>Default SID</td>
<td>* :l:</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>:ND:</td>
<td>Null delta flag</td>
<td>*</td>
<td>yes or no</td>
<td>S</td>
</tr>
<tr>
<td>:FD:</td>
<td>File descriptive text</td>
<td>Comments</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:BD:</td>
<td>Body</td>
<td>Body</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:GB:</td>
<td>Gotten body</td>
<td>*</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:W:</td>
<td>A form of what (1) string</td>
<td>N/A</td>
<td>:Z:/:M::I:</td>
<td>S</td>
</tr>
<tr>
<td>:A:</td>
<td>A form of what (1) string</td>
<td>N/A</td>
<td>:Z:/:Y:/:M:/:l:/:Z:</td>
<td>S</td>
</tr>
<tr>
<td>:Z:</td>
<td>what (1) string delimiter</td>
<td>N/A</td>
<td>@(#)</td>
<td>S</td>
</tr>
<tr>
<td>:F:</td>
<td>SCCS file name</td>
<td>N/A</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:PN:</td>
<td>SCCS file path name</td>
<td>N/A</td>
<td>text</td>
<td>S</td>
</tr>
</tbody>
</table>

EXAMPLES

prs -d "Users and/or user IDs for :F: are:
\nUN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:

xyz
131
abc

prs -d "Newest delta for pgm :M: :I: Created :D: By :P: -r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a special case:

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/000000/000000
MRs:
bl78-12345
bl79-54321
COMMENTS:

this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument
allowed to be used with the special case is the -a keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(5).


DIAGNOSTICS

Use help(1) for explanations.
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. Otherwise, the information that is displayed is controlled by the following options:

-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. (Normally, a short listing containing only process ID, terminal ("tty") identifier, cumulative execution time, and the command name is printed.) See below for meaning of columns in a full listing.
-l Generate a long listing. See below.
-c corefile Use the file corefile in place of /dev/mem.
-s swapdev Use the file swapdev in place of /dev/swap. This is useful when examining a corefile; a swapdev of /dev/null will cause the user block to be zeroed out.
-n namelist The argument will be taken as the name of an alternate namelist (/unix is the default).
-t tlist Restrict listing to data about the processes associated with the terminals given in tlist, where tlist can be in one of two forms: a list of terminal identifiers separated from one another by a comma, or a list of terminal identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.
-p plist Restrict listing to data about processes whose process ID numbers are given in plist, where plist is in the same format as tlist.
-u ulist Restrict listing to data about processes whose user ID numbers or login names are given in ulist, where ulist is in the same format as tlist. In the listing, the numerical user ID will be printed unless the -f option is used, in which case the login name will be printed.
-g glist Restrict listing to data about processes whose process groups are given in glist, where glist is a list of process group leaders and is in the same format as tlist.

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 only determine what information is provided for a process; they do not determine which processes will be listed.

F (1) Flags (octal and additive) associated with the process:

01 in core;
02 system process;
04 locked in core (e.g., for physical I/O);
10 being swapped;
20 being traced by another process.
The state of the process:
- 0 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; it is possible to kill a process if you know this datum.
PPID (f,l) The process ID of the parent process.
C (f,l) Processor utilization for scheduling.
STIME (f) Starting time of the process.
PRI (l) The priority of the process; higher numbers mean lower priority.
NI (l) Nice value; used in priority computation.
ADDR (l) The memory address of the process, if resident; otherwise, the disk address.
SZ (l) The size in blocks of the core image of the process.
WCHAN (l) The event for which the process is waiting or sleeping; if blank, the process is running.
TTY (all) The controlling terminal for the process.
TIME (all) The cumulative execution time for the process.
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 memory or the swap area. Failing this, the command name, as it would appear without the -f option, is printed in square brackets.

FILES
/ unix system namelist
/ dev/mem memory
/ dev searched to find swap device and terminal ("tty") names.

SEE ALSO
kill(1), nice(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 are irrelevant.
NAME
ptx — permuted index

SYNOPSIS
ptx [ options ] [ input [ output ]]

DESCRIPTION
Ptx generates a permuted index to file input on file output (standard input
and output default). It has three phases: the first does the permutation,
generating one line for each keyword in an input line. The keyword is rota-
ted to the front. The permuted file is then sorted. Finally, the sorted lines
are rotated so the keyword comes at the middle of each line. Ptx produces
output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx is assumed to be an nroff or troff(1) macro provided by the user.
The before keyword and keyword and after fields incorporate as much of the
line as will fit around the keyword when it is printed. Tail and head, at
least one of which is always the empty string, are wrapped-around pieces
small enough to fit in the unused space at the opposite end of the line.

The following options can be applied:

-f Fold upper and lower case letters for sorting.
-t Prepare the output for the phototypesetter.
-w n Use the next argument, n, as the length of the output line.
The default line length is 72 characters for nroff and 100 for troff.
-g n Use the next argument, n, as the number of characters that ptx
will reserve in its calculations for each gap among the four
parts of the line as finally printed. The default gap is 3 charac-
ters.
-o only Use as keywords only the words given in the only file.
-i ignore Do not use as keywords any words given in the ignore file. If
the -i and -o options are missing, use /usr/lib/eign as the
ignore file.
-b break Use the characters in the break file to separate words. Tab,
new-line, and space characters are always used as break charac-
ters.
-r Take any leading non-blank characters of each input line to be
a reference identifier (as to a page or chapter), separate from
the text of the line. Attach that identifier as a 5th field on each
output line.

The index for this manual was generated using ptx.

FILES
/bin/sort
/usr/lib/eign

BUGS
Line length counts do not account for overstriking or proportional spacing.
Lines that contain tildes (`) are botched, because ptx uses that character
internally.
NAME
pwck, grpck — password/group file checkers

SYNOPSIS
pwck [file]
grpck [file]

DESCRIPTION
Pwck scans the password file and notes any inconsistencies. The checks
include validation of the number of fields, login name, user ID, group ID,
and whether the login directory and optional program name exist. The cri-
teria for determining a valid login name are taken from Setting Up UNIX.
The default password file is /etc/passwd.

Grpck verifies all entries in the group file. This verification includes a check
of the number of fields, group name, group ID, and whether all login
names appear in the password file. The default group file is /etc/group.

FILES
/etc/group
/etc/passwd

SEE ALSO
group(5), passwd(5).
Setting Up UNIX.

DIAGNOSTICS
Group entries in /etc/group with no login names are flagged.
NAME
pwd — working directory name

SYNOPSIS
pwd

DESCRIPTION
Pwd prints the path name of the working (current) directory.

SEE ALSO
cd(1).

DIAGNOSTICS
"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX programming counselor.
NAME
ratfor — rational Fortran dialect

SYNOPSIS
ratfor [ options ] [ files ]

DESCRIPTION
_Ratfor_ converts a rational dialect of Fortran into ordinary irrational Fortran.
_Ratfor_ provides control flow constructs essentially identical to those in C:

statement grouping:

```
{ statement; statement; statement }
```

decision-making:

```
if (condition) statement [ else statement ]
switch (integer value) {
    case integer: statement
    ...[
    [ default: ] statement
}
```

loops:

```
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [ until (condition) ]
break
next
```

and some syntactic sugar to make programs easier to read and write:

free form input:

```
multiple statements/line; automatic continuation
```

comments:

```
# this is a comment.
```

translation of relationals:

```
>, >=, etc., become .GT., .GE., etc.
```

return expression to caller from function:

```
return (expression)
```

define:

```
declare name replacement
```

include:

```
include file
```

The option -h causes quoted strings to be turned into 27H constructs.
The -C option copies comments to the output and attempts to format it
neatly. Normally, continuation lines are marked with a & in column 1; the
option -6x makes the continuation character x and places it in column 6.

_Ratfor_ is best used with _f77_(1).

SEE ALSO
efl(1), f77(1).
NAME
 reform - reformat text file

SYNOPSIS
 reform [tabspec1 [tabspec2]] [+bn] [+en] [+f] [+in] [+mn] [+pn] [+s] [+tn]

DESCRIPTION
 Reform reads each line of the standard input file, reformats it, and then
 writes it to the standard output. Various combinations of reformatting
 operations can be selected, of which the most common involve reaar-
 rangement of tab characters. It is often used to trim trailing blanks, trun-
 cate lines to a specified length, or prepend blanks to lines.
 Reform first scans its arguments, which may be given in any order. It then
 processes its input file, performing the following actions upon each line, in
 the order given:
 - A line is read from the standard input.
 - If +s is given, all characters up to the first tab are stripped off and
   saved for later addition to the end of the line. Presumably, these
   characters comprise an “SCCS SID” produced by get(1).
 - The line is expanded into a tableless form, by replacing tabs with blanks
   according to the input tab specification tabspec1.
 - If +pn is given, n blanks are prepended to the line.
 - If +tn is given, the line is truncated to a length of n characters.
 - All trailing blanks are now removed.
 - If +en is included, the line is extended out with blanks to the length of
   n characters.
 - If +s is given, the previously-saved “SCCS SID” is added to the end of
   the line.
 - If +bn is given, the n characters at the beginning of the line are conver-
   ted to blanks, if and only if all of them are either digits or blanks.
 - If +mn is included, the line is moved left, i.e., n characters are remo-
   ved from the beginning of the line.
 - The line is now contracted by replacing some blanks with tab characters
   according to the list of tabs indicated by the output tab specification
   tabspec2, and is written to the standard output file. Option +i controls
   the method of contraction (see below).

The various arguments accepted by reform are as follows:

tabspec1 describes the tab stops assumed for the input file. This tab
specification may take on any of the forms described in tabs(1).
In addition, the operand – – indicates that the tab specification
is to be found in the first line read from the standard input. If
no legal tab specification is found there, – 8 is assumed. If
tabspec1 is omitted entirely, – – is assumed.

tabspec2 describes the tabs assumed for the output file. It is interpreted
in the same way as tabspec1, except that omission of tabspec2
causes the value of tabspec1 to be used for tabspec2.

The remaining arguments are all optional and may be used in any com-
bination, although only a few combinations make much sense. Specifying
an argument causes an action to be performed, as opposed to the usual
default of not performing the action. Some options include numeric
values, which also have default values. Option actions are applied to each line in the order described above. Any line length mentioned applies to the length of a line just before the execution of the option described, and the terminating new-line is never counted in the line length.

+bn causes the first $n$ characters of a line to be converted to blanks, if and only if those characters include only blanks and digits. If $n$ is omitted, the default value is 6, which is useful in deleting sequence numbers from COBOL programs.

+en causes each line shorter than $n$ characters to be extended out with blanks to that length. Omitting $n$ implies a default value of 72. This option is useful for those rare cases in which sequence numbers need to be added to an existing unnumbered file. The use of $s$ in editor regular expressions is more convenient if all lines have equal length, so that the user can issue editor commands such as:

```
s/$00*01000/
```

+f causes a format line to be written to the standard output, preceding any other lines written. See fspec(5) for details regarding format specifications. The format line is taken from tabspec2, i.e., the line normally appears as follows:

```
<:t-tabspec2 d:>
```

If tabspec2 is of the form --file-name (i.e., an indirect reference to a tab specification in the first line of the named file), then that tab specification line is written to the standard output.

+in controls the technique used to compress interior blanks into tabs. Unless this option is specified, any sequence of 1 or more blanks may be converted to a single tab character if that sequence occurs just before a tab stop. This causes no problems for blanks that occur before the first nonblank character in a line, and it is always possible to convert the result back to an equivalent tabless form. However, occasionally an interior blank (one occurring after the first nonblank) is converted to a tab when this is not intended. For instance, this might occur in any program written in a language utilizing blanks as delimiters. Any single blank might be converted to a tab if it occurred just before a tab stop. Insertion or deletion of characters preceding such a tab may cause it to be interpreted in an unexpected way at a later time. If the +i option is used, no string of blanks may be converted to a tab unless there are $n$ or more contiguous blanks. The default value is 2. Note that leading blanks are always converted to tabs when possible. It is recommended that conversion of programs from non-UNIX to UNIX systems use this option.

+mn causes each line to be moved lift $n$ characters, with a default value of 6. This can be useful for crunching COBOL programs.

+pn causes $n$ blanks to be prepended (default of 6 if $n$ is omitted). This option is effectively the inverse of +mn, and is often useful for adjusting the position of nroff(1) output for terminals lacking both forms tractor positioning and a settable left margin.

+s is used with the -m option of get(1). The -m option causes get to prepend to each generated line the appropriate SCCS SID,
followed by a tab. The +s option causes *reform* to remove the SID from the front of the line, save it, then add it later to the end of the line. Because +e72 is implied by this option, the effect is to produce 80-character card images with SCCS SID in columns 73-80. Up to 8 characters of the SID are shown; if it is longer, the eighth character is replaced by • and any characters to the right of it are discarded.

The following illustrate typical uses of *reform*. The terms PWB and OBJECT below refer to UNIX and non-UNIX computer systems, respectively. Each arrow indicates the direction of conversion. The character ? indicates an arbitrary tab specification; see *tabs*(1) for descriptions of legal specifications.

**OBJECT — — > PWB** (i.e., manipulation of RJE output):

Note that files transferred by RJE from OBJECT to PWB materialize with format -8.

reform -8 -c +t +b +i <oldfile >newfile (into COBOL)
reform -8 -c3 +t +m +i <oldfile >newfile (into COBOL, crunched)

NOTE: -c3 is the preferred format COBOL; it uses the least disk space of the COBOL formats.

**PWB — — > OBJECT** (i.e., preparation of files for RJE submission):

reform ?-8 <oldfile >newfile (from arbitrary format into -8)

get -p -m sccsfile | reform +s | send ...

**PWB ONLY** (i.e., no involvement with other systems):

pr file | reform ? -0 <oldfile (print on terminal without hardware tabs)
reform ? -0 <oldfile >newfile (convert file to tableless format)

**DIAGNOSTICS**

All diagnostics are fatal, and the offending line is displayed following the message.

"line too long" a line exceeds 512 characters (in tab less form).

"not SCCS -m" a line does not have at least one tab when +s flag is used.

Any of the diagnostics of *tabs*(1) can also appear.

**EXIT CODES**

0 — normal
1 — any error

**SEE ALSO**

get(1), nroff(1), send(1C), tabs(1), fspec(5).

**BUGS**

*Reform* is aware of the meanings of backspaces and escape sequences, so that it can be used as a postprocessor for *nroff*. However, be warned that the +e, +m, and +t options only count characters, not positions. Anyone using these options on output containing backspaces or halfline motions will probably obtain unexpected results.
NAME
   regcmp — regular expression compile

SYNOPSIS
   regcmp [ - ] files

DESCRIPTION
   Regcmp, in most cases, precludes the need for calling regcmp (see
   regex(3X)) from C programs. This saves on both execution time and pro-
   gram size. The command regcmp compiles the regular expressions in file
   and places the output in file.i. If the - option is used, the output will be
   placed in file.c. The format of entries in file is a name (C variable) fol-
   lowed by one or more blanks followed by a regular expression enclosed in
   double quotes. The output of regcmp is C source code. Compiled regular
   expressions are represented as extern char vectors. File.i files may thus be
   included into C programs, or file.c files may be compiled and later loaded.
   In the C program which uses the regcmp output, regex(abc, line) will apply
   the regular expression named abc to line. Diagnostics are self-explanatory.

EXAMPLES
   name   "([A-Za-z][A-Za-z0-9][*])$0"
   telno  "\{[0,1]\{2-9\}[01][1-9]\{0,1\}\{0,1\} *"
           "([2-9][0-9][2])$1{ -}{0,1}"
           "([0-9][4])$2"

In the C program that uses the regcmp output,
   regex(telno, line, area, exch, rest)

will apply the regular expression named telno to line.

SEE ALSO
   regex(3X).
NAME
restor — incremental file system restore

SYNOPSIS
restor key [ arguments ]

DESCRIPTION
Restor is used to read magnetic tapes dumped with the dump command. The key specifies what is to be done. Key is one of the characters rRxt, optionally combined with f.

f Use the first argument as the name of the tape instead of the default.

r or R The tape is read and loaded into the file system specified in argument. This should not be done lightly (see below). If the key is R, restor asks which tape of a multi-volume set to start on. This allows restor to be interrupted and then restarted (an fsck must be done before the restart).

x Each file on the tape named by an argument is extracted. The file name has all "mount" prefixes removed; for example, if /usr is a mounted file system, /usr/bin/lpr is named /bin/lpr on the tape. The extracted file is placed in a file with a numeric name supplied by restor (actually the inode number). In order to keep the amount of tape read to a minimum, the following procedure is recommended:

1. Mount volume 1 of the set of dump tapes.
2. Type the restor command.
3. Restor will announce whether or not it found the files, give the numeric name that it will assign to the file, and rewind the tape.
4. It then asks you to "mount the desired tape volume". Type the number of the volume you choose. On a multi-volume dump the recommended procedure is to mount the last through the first volumes, in that order. Restor checks to see if any of the requested files are on the mounted tape (or a later tape—thus the reverse order) and doesn't read through the tape if no files are. If you are working with a single-volume dump or if the number of files being restored is large, respond to the query with 1 and restor will read the tapes in sequential order.

t Print the date the tape was written and the date the file system was dumped from.

The r option should only be used to restore a complete dump tape onto a clear file system, or to restore an incremental dump tape onto a file system so created. Thus:

/etc/mkfs /dev/rp0 40600
restor r /dev/rp0

is a typical sequence to restore a complete dump. Another restor can be done to get an incremental dump in on top of this.

A dump followed by a mkfs and a restor is used to change the size of a file system.

FILES
default tape unit varies with installation
RESTOR (1M) (Obsolescent) RESTOR (1M)

rst*

SEE ALSO
dump(1M), fsck(1M), mkfs(1M).

DIAGNOSTICS
There are various diagnostics involved with reading the tape and writing the
disk. There are also diagnostics if the i-list or the free list of the file system
is not large enough to hold the dump.

If the dump extends over more than one tape, it may ask you to change
tapes. Reply with a new-line when the next tape has been mounted.

BUGS
There is redundant information on the tape that could be used in case of
tape reading problems. Unfortunately, restor doesn't use it.
NAME
rjestat - RJE status report and interactive status console

SYNOPSIS
rjestat [ host ]... [ -shost ] [ -chost cmd ]...

DESCRIPTION
Rjestat provides a method of determining the status of an RJE link and of simulating an IBM remote console (with UNIX features added). When invoked with no arguments, rjestat reports the current status of all the RJE links connected to to the UNIX system. The options are:

host     Print the status of the line to host. Host is the pseudonym for a particular IBM system. It can be any name that corresponds to one in the first column of the RJE configuration file.

-shost   After all the arguments have been processed, start an interactive status console to host.

-chost cmd
Interpret cmd as if it were entered in status console mode to host. See below for the proper format of cmd.

In status console mode, rjestat prompts with the host pseudonym followed by : whenever it is ready to accept a command. Commands are terminated with a new-line. A line that begins with ! is sent to the UNIX shell for execution. A line that begins with the letter q terminates rjestat. All other input lines are assumed to have the form:

ibmcmd [ redirect ]

Ibmcmd is any IBM JES or HASP command. Only the super-user or rje login can send commands other than display or inquiry commands. Redirect is a pipeline or a redirection to a file (e.g., "> file" or " | grep ..." ). The IBM response is written to the pipeline or file. If redirect is not present, the response is written to the standard output of rjestat.

An interrupt signal (DEL or BREAK) will cancel the command in progress and cause rjestat to return to the command input mode.

EXAMPLE
The following command reports the status of all the card readers attached to host A, remote 5. JES2 is assumed.

rjestat -cA 'Sdu,rmt5 | grep RD'

DIAGNOSTICS
The message "RJE error: ..." indicates that rjestat found an inconsistency in the RJE system. This may be transient but should be reported to the site administrator.

FILES
/usr/rje/lines  RJE configuration file
resp         host response file that exists in the RJE subsystem directory (e.g., /usr/rjel).

SEE ALSO
send(1C), rje(8).
Operator's Library: OS/VS2 Reference (JES2), IBM SRL #GC38-0210.
NAME
rm, rmdir — remove files or directories

SYNOPSIS
rm [ -fri ] file ...
rm -r dir ...

DESCRIPTION
Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with y the file is deleted, otherwise the file remains. No questions are asked when the -f option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument -r has been used. In that case, rm recursively deletes the entire contents of the specified directory, and the directory itself.

If the -i (interactive) option is in effect, rm asks whether to delete each file, and, under -r, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO
unlink(2).

DIAGNOSTICS
 Generally self-explanatory. It is forbidden to remove the file .. merely to avoid the antisocial consequences of inadvertently doing something like:

        rm -r ..
NAME
    rmdel — remove a delta from an SCCS file

SYNOPSIS
    rmdel -rSID files

DESCRIPTION
    Rmdel removes the delta specified by the SID from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must not be that of a version being edited for the purpose of making a delta (i.e., if a p-file (see get(1)) exists for the named SCCS file, the specified must not appear in any entry of the p-file).

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

    The exact permissions necessary to remove a delta are documented in the Source Code Control System User's Guide. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES
    x-file  (see delta(1))
    z-file  (see delta(1))

SEE ALSO
    delta(1), get(1), help(1), prs(1), sccsfile(5).

DIAGNOSTICS
    Use help(1) for explanations.
NAME
rsh — restricted shell (command interpreter)

SYNOPSIS
rsh [ flags ] [ name [ argl ... ] ]

DESCRIPTION
Rsh is a restricted version of the standard command interpreter sh(1). It is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of rsh are identical to those of sh, except that the following are disallowed:

cd
setting the value of SPATH
command names containing /
> and >>

When invoked with the name -rsh, rsh reads the user’s .profile (from $HOME/.profile). It acts as the standard sh while doing this, except that an interrupt causes an immediate exit, instead of causing a return to command level. The restrictions above are enforced after .profile is interpreted.

When a command to be executed is found to be a shell procedure, rsh invokes sh to execute it. Thus, it is possible to provide to the end user shell procedures that have access to the full power of the standard shell, while restricting him to 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, then leaving the user in an appropriate directory (probably not the login directory).

Rsh is actually just a link to sh and any flags arguments are the same as for sh(1).

The system administrator often sets up a directory of commands that can be safely invoked by rsh. Some systems also provide a restricted editor red.

SEE ALSO
sh(1), profile(5).
NAME
  runacct - run daily accounting

SYNOPSIS
  runacct [mmdd [state]]

DESCRIPTION
  Runacct is the main daily accounting shell procedure. It is normally initiated via cron(1M). Runacct processes connect, fee, disk, and process accounting files. It also prepares summary files for prdaily or billing purposes.

  Runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into active. When an error is detected, a message is written to /dev/console, mail (see mail(1)) is sent to root and adm, and runacct terminates. Runacct uses a series of lock files to protect against re-invocation. The files lock and lock1 are used to prevent simultaneous invocation, and lastdate is used to prevent more than one invocation per day.

  Runacct breaks its processing into separate, restartable states using statefile to remember the last state completed. It accomplishes this by writing the state name into statefile. Runacct then looks in statefile to see what it has done and to determine what to process next. States are executed in the following order:

  SETUP     Move active accounting files into working files.
  WTMPFIX   Verify integrity of wtmp file, correcting date changes if necessary.
  CONNECT1  Produce connect session records in ctmp.h format.
  CONNECT2  Convert ctmp.h records into tacct.h format.
  PROCESS   Convert process accounting records into tacct.h format.
  MERGE     Merge the connect and process accounting records.
  FEES      Convert output of chargefee into tacct.h format and merge with connect and process accounting records.
  DISK      Merge disk accounting records with connect, process, and fee accounting records.
  MERGETACCT Merge the daily total accounting records in daytacct with the summary total accounting records in /usr/adm/acct/sum/tacct.
  CMS       Produce command summaries.
  USEREXIT  Any installation-dependent accounting programs can be included here.
  CLEANUP   Cleanup temporary files and exit.

To restart runacct after a failure, first check the active file for diagnostics, then fix up any corrupted data files such as pacct or wtmp. The lock files and lastdate file must be removed before runacct can be restarted. The argument mmdd is necessary if runacct is being restarted, and specifies the month and day for which runacct will rerun the accounting. Entry point for processing is based on the contents of statefile; to override this, include the desired state on the command line to designate where processing should begin.
EXAMPLES
To start `runacct`.
    `nohup runacct 2> /usr/adm/acct/nite/fd2log &`
To restart `runacct`.
    `nohup runacct 0601 2> > /usr/adm/acct/nite/fd2log &`
To restart `runacct` at a specific state.
    `nohup runacct 0601 MERGE 2> > /usr/adm/acct/nite/fd2log &`

FILES
`/usr/lib/acct/runacct`
`/usr/adm/wtmp`
`/usr/adm/acct[1-9]`
`/usr/src/cmd/acct/tacct.h`
`/usr/src/cmd/acct/ctmp.h`
`/usr/adm/acct/nite/active`
`/usr/adm/acct/nite/dayacct`
`/usr/adm/acct/nite/lock`
`/usr/adm/acct/nite/lock 1`
`/usr/adm/acct/nite/lastdate`
`/usr/adm/acct/nite/statefile`
`/usr/adm/acct/nite/ptacct[1-9].mmd d`

SEE ALSO
`acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), acct(2), acct(5), utmp(5).`

The UNIX Accounting System by H. S. McCreary.

DIAGNOSTICS
Self explanatory.

BUGS
Normally it is not a good idea to restart `runacct` in the SETUP state. Run SETUP manually and restart via:

    `runacct mmd d WTMPFIX`

If `runacct` failed in the PROCESS state, remove the last `ptacct` file because it will not be complete.
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 behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 specifies the SID for the new delta to be created.

Field 3 contains the logname of the user who will make the delta (i.e. executed a get for editing).

Field 4 contains the date that get -e was executed.

Field 5 contains the time that get -e was executed.

SEE ALSO
delta(1), get(1), unget(1).

DIAGNOSTICS
Use help(1) for explanations.
NAME
sag — system activity graph

SYNOPSIS
sag [-s time ] [-e time ] [-T term ] [-uirwcohdpaf ] [ file ]

DESCRIPTION
Sag displays, in a graphical form, the system activity of the UNIX operating
system during a specified time interval. File is the file that contains the
daily system activity information, default is /usr/adm/sa/sadd, where dd is
today's day of the month. Sag has the following options:

- s time Begin graph at time specified as hh:mm. Default is 08:00.
- e time End graph at time specified as hh:mm. Default is 18:00.
- T term Translate output to a form suitable for terminal term. If this
  option is not used, the environment variable STERM (see
  environ(7)) is used. Refer to tplot(1G) for available types of
  terminals.
- u Plot CPU utilization, showing proportion of user, system and
  idle time (default option).
- i Plot percent of time the CPU was idle and waiting on block
  I/O, waiting on swap in or swap out, or waiting on physical
  I/O.
- r Plot logical reads/minute and block reads/minute.
- w Plot logical writes/minute and block writes/minute.
- c Plot buffer cache hit ratios for reads and for writes.
- o Plot block transfer rate between system buffers and devices,
  showing reads/minute, writes/minute, and the sum of reads
  and writes/minute.
- h Plot bytes read/second by system call read(2) and bytes
  written/second by system call write(2).
- d Plot the sum of reads and writes/minute for each of the first
  three RP06 type disk drives.
- p Plot process switches/second, process preemptions/second and
  system calls/second.
- a Plot process swapins/minute and process swapouts/minute.
- f Plot file access activities: iget/second, namei/second, and
  directory blocks read/second.

FILES
/usr/adm/sa/sadd daily data file, where dd are digits representing the
day of the month.

SEE ALSO
graph(1G), tplot(1G), sar(8).

NOTES
Plotted data points are extracted from the system activity file,
/usr/adm/sa/sadd, which is written under the control of cron(1M), nor-
mally every 20 minutes between 8:00 and 18:00 on weekdays, and hourly at
other times.
In the event of a system outage, the system activity counters are reset to
zero when the system is rebooted. This discontinuity is shown by a gap in
the plotted data.

DIAGNOSTICS
"terminal type not known" if STERM is not set and the -T option is not
specified.
NAME
scc - C compiler for stand-alone programs

SYNOPSIS
scc [ +[ lib ] ] [ option ] ... [ file ] ...

DESCRIPTION
Scc prepares the named files for stand-alone execution. The option and file
arguments may be anything that can legally be used with the cc command;
it should be noted, though, that the -p (profiling) option, as well as any
object module that contains system calls, will cause the executable not to
run.

Scc defines the compiler constant, STANDALONE, so that sections of C
programs may be compiled conditionally for when the executable will be
run stand-alone.

The first argument specifies an auxiliary library that defines the device
configuration of the PDP-11 computer for which the stand-alone executable
is being prepared. Lib may be one of the following:

A RP04/05/06 disk and TU16 magnetic tape, or equivalent
B RK11/RK05 disk, RP11/RP03 disk, and TM11/TU16 magnetic tape,
or equivalent

If no +lib argument is specified, +A is assumed. If the + argument is
specified alone, no configuration library is loaded unless the user supplies
his own.

FILES
/lib/crt20.o execution start-off
/usr/lib/lib2.a stand-alone library
/usr/lib/lib2A.a +A configuration library (PDP-11 only)
/usr/lib/lib2B.a +B configuration library (PDP-11 only)

SEE ALSO
cc(1), ld(1), a.out(5).
A Stand-alone Input/Output Library, by S. R. Eisen.
NAME
sccsdiff — compare two versions of an SCCS file

SYNOPSIS
sccsdiff -rSID1 -rSID2 [-p] [-sn] files

DESCRIPTION
Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

- rSID? SID1 and SID2 specify the deltas of an SCCS file that are to be compared. Versions are passed to bdiff(1) in the order given.

-p pipe output for each file through pr(1).

-sn n is the file segment size that bdiff will pass to diff(1). This is useful when diff fails due to a high system load.

FILES
/tmp/get?????? Temporary files

SEE ALSO
bdiff(1), get(1), help(1), pr(1).

DIAGNOSTICS
"file: No differences" If the two versions are the same.
Use help(1) for explanations.
NAME
sdb — symbolic debugger

SYNOPSIS
sdb [ objfil [ corfil [ directory ] ] ]

DESCRIPTION
Sdb is a symbolic debugger which can be used with C and F77 programs. It may be used to examine their files and to provide a controlled environment for their execution.

Objfil is an executable program file which has been compiled with the –g (debug) option. The default for objfil is a.out. Corfil is assumed to be a core image file produced after executing objfil; the default for corfil is core. The core file need not be present.

It is useful to know that at any time there is a current line and current file. If corfil exists then they are initially set to the line and file containing the source statement at which the process terminated or stopped. Otherwise, they are set to the first line in main(). The current line and file may be changed with the source file examination commands.

Names of variables are written just as they are in C or F77. Variables local to a procedure may be accessed using the form procedure:variable. If no procedure name is given, the procedure containing the current line is used by default. It is also possible to refer to structure members as variable.member, pointers to structure members as variable—>member and array elements as variable[number]. Combinations of these forms may also be used.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal or hexadecimal.

Line numbers in the source program are referred to as file-name:number or procedure:number. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

The commands for examining data in the program are:

- t  Print a stack trace of the terminated or stopped program.
- T  Print the top line of the stack trace.

variable/Im
Print the value of variable according to length l and format m. If l and m are omitted, sdb chooses a length and format suitable for the variable’s type as declared in the program. The length specifiers are:

- b  one byte
- h  two bytes (half word)
- l  four bytes (long word)

number
string length for formats s and a

Legal values for m are:
- c  character
- d  decimal
- u  decimal, unsigned
- o  octal
- x  hexadecimal
f 32 bit single precision floating point

32

g 64 bit double precision floating point

64

s Assume variable is a string pointer and print characters starting at the address pointed to by the variable.

s

a Print characters starting at the variable’s address.

a

p pointer to procedure

The length specifiers are only effective with the formats d, u, o and x. If one of these formats is specified and l is omitted, the length defaults to the word length of the host machine; 4 for the VAX-11/780. If a numeric length specifier is used for the s or a command then that many characters are printed. Otherwise successive characters are printed until either a null byte is reached or 128 characters are printed. The last variable may be redisplayed with the command ./.

The sh(1) metacharacters * and ? may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, both variables local to the current procedure and global (common for F77) variables are matched, while if a procedure name is specified then only variables local to that procedure and matched. To match only global variables (or blank common for F77), the form :pattern is used. The name of a common block may be specified instead of a procedure name for F77 programs.

variable = lm

Print the address of variable or linenumber, or the value of number in the format specified by lm. If no format is given, then lx is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

variable!value

Set variable to the given value. The value may be a number, character constant or a variable. If the variable is of type float or double, the value may also be a floating constant.

The commands for examining source files are:

e procedure

Set the current file to the file containing procedure or to file-name. Set the current line to the first line in the named procedure or file. If no procedure or file name is given, the current procedure and file names are reported.

/regular expression/

Search forward from the current line for a line containing a string matching regular expression as in ed(1). The trailing / may be elided.

?regular expression?

Search backward from the current line for a line containing a string matching regular expression as in ed(1). The trailing ? may be elided.

p Print the current line.

z Print the current line followed by the next 9 lines. Set the current line to the last line printed.

ccontrol-D

Scroll. Print the next 10 lines. Set the current line to the last line printed.
Window. Print the 10 lines around the current line.

Set the current line to the given line number. Print the new current line.

Advance the current line by count lines. Print the new current line.

Retreat the current line by count lines. Print the new current line.

The commands for controlling the execution of the source program are:

Run the program with the given arguments. The r command with no arguments reuses the previous arguments to the program while the R command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output respectively. If count is given, it specifies the number of breakpoints to be ignored.

Continue after a breakpoint or interrupt. If count is given, it specifies the number of breakpoints to be ignored. C continues with the signal which caused the program to stop and c ignores it. If a linenumber is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

Continue after a breakpoint with execution resumed at the given line. If count is given, it specifies the number of breakpoints to be ignored.

Single step. Run the program through count lines. If no count is given then the program is run for one line.

Single step, but step through subroutine calls.

Kill the debugged program.

Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format m. If no format is given, it defaults to d.

Set a breakpoint at the given line. If a procedure name without a line number is given (e.g. "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled with the debug flag. If no linenumber is given, a breakpoint is placed at the current line. If no commands are given then execution stops just before the breakpoint and control is returned to sdb. Otherwise the commands are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons.
B Print a list of the currently active breakpoints.

`linenumber d`
Delete a breakpoint at the given line. If no `linenumber` is given then the breakpoints are deleted interactively: Each breakpoint location is printed and a line is read from the standard input. If the line begins with a y or d then the breakpoint is deleted.

D Delete all breakpoints.

I Print the last executed line.

`linenumber a`
Announce. If `linenumber` is of the form `proc:number`, the command effectively does a `linenumber b l`. If `linenumber` is of the form `proc:`, the command effectively does a `proc: b T`.

Miscellaneous commands:

`!command`
The command is interpreted by `sh(1)`.

`new-line`
If the previous command printed a source line then advance the current line by 1 line and print the new current line. If the previous command displayed a core location then display the next core location.

`* string`
Print the given string.

q Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

V Print the version number.

X Print a list of procedures and files being debugged.

Y Toggle debug output.

FILES

a.out

core

SEE ALSO

adb(1), a.out(5), core(5).

DIAGNOSTICS

Error reports are either identical to those of adb(1) or are self-explanatory.

BUGS

If a procedure is called when the program is not stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

Arrays must be of one dimension and of zero origin to be correctly addressed by sdb.

The default type for printing F77 parameters is incorrect. Their address is printed instead of their value.

Tracebacks containing F77 subprograms with multiple entry points may print too many arguments in the wrong order, but their values are correct.
NAME
sdiff — side-by-side difference program

SYNOPSIS
sdiff [ options ... ] file1 file2

DESCRIPTION
Sdiff uses the output of diff(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in file1, a > in the gutter if the line only exists in file2, and a | for lines that are different.

For example:

\[
x | y \\
a  | a \\
b < \\
c < \\
d > c
\]

The following options exist:

-w n Use the next argument, n, as the width of the output line. The default line length is 130 characters.
-l Only print the left side of any lines that are identical.
-s Do not print identical lines.
-o output Use the next argument, output, as the name of a third file that is created as a user controlled merging of file1 and file2. Identical lines of file1 and file2 are copied to output. Sets of differences, as produced by diff(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, sdiff prompts the user with a % and waits for one of the following user-typed commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>append the left column to the output file</td>
</tr>
<tr>
<td>r</td>
<td>append the right column to the output file</td>
</tr>
<tr>
<td>s</td>
<td>turn on silent mode; do not print identical lines</td>
</tr>
<tr>
<td>v</td>
<td>turn off silent mode</td>
</tr>
<tr>
<td>e l</td>
<td>call the editor with the left column</td>
</tr>
<tr>
<td>e r</td>
<td>call the editor with the right column</td>
</tr>
<tr>
<td>e b</td>
<td>call the editor with the concatenation of left and right</td>
</tr>
<tr>
<td>e</td>
<td>call the editor with a zero length file</td>
</tr>
<tr>
<td>q</td>
<td>exit from the program</td>
</tr>
</tbody>
</table>

On exit from the editor, the resulting file is concatenated on the end of the output file.

SEE ALSO
diff(1), ed(1).
NAME

sed - stream 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.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function ! (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The text argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The rfile or wfile argument must terminate the command line and must be preceded by exactly one blank. Each wfile is created before processing begins. There can be at most 10 distinct wfile arguments.
Append. Place text on the output before reading the next input line.

Branch to the : command bearing the label. If label is empty, branch to the end of the script.

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.

Delete the pattern space. Start the next cycle.

Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

Replace the contents of the pattern space by the contents of the hold space.

Append the contents of the hold space to the pattern space.

Replace the contents of the hold space by the contents of the pattern space.

Append the contents of the pattern space to the hold space.

Insert. Place text on the standard output.

List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

Copy the pattern initial segment of the pattern space through the first new-line to the standard output.

Quit. Branch to the end of the script. Do not start a new cycle.

Read the contents of rfile. Place them on the output before reading the next input line.

Substitute the replacement string for instances of the regular expression in the pattern space. Any character may be used instead of /. For a fuller description see ed(1). Flags is zero or more of:

- **g** Global. Substitute for all nonoverlapping instances of the regular expression rather than just the first one.
- **p** Print the pattern space if a replacement was made.
- **w wfile** Write. Append the pattern space to wfile if a replacement was made.

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.

Write. Append the pattern space to wfile.

Exchange the contents of the pattern and hold spaces.

Transform. Replace all occurrences of characters in string1 with the corresponding character in string2. The lengths of string1 and string2 must be equal.
(2) ! function
Don't. Apply the function (or group, if function is {) only to
lines not selected by the address(es).

(0) : label This command does nothing; it bears a label for b and t com-
mands 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.

SEE ALSO
awk(1), ed(1), grep(1).
SED—A Non-interactive Text Editor by L. E. McMahon.
NAME
send, gath — gather files and/or submit RJE jobs

SYNOPSIS

\texttt{gath} \ [-ih] \ \texttt{file} . . .
\texttt{send} \ \texttt{argument} . . .

DESCRIPTION

\textit{Gath}
\texttt{Gath} concatenates the named files and writes them to the standard output. Tabs are expanded into spaces according to the format specification for each file (see \texttt{fspec(5)}). The size limit and margin parameters of a format specification are also respected. Non-graphic characters other than tabs are identified by a diagnostic message and excised. The output of \texttt{gath} contains no tabs unless the \texttt{-h} flag is set, in which case the output is written with standard tabs (every eighth column).

Any line of any of the files which begins with \texttt{'} is interpreted by \texttt{gath} as a control line. A line beginning \texttt{"\~"} (tilde, space) specifies a sequence of files to be included at that point. A line beginning \texttt{'}! specifies a UNIX command; that command is executed, and its output replaces the \texttt{'}! line in the \texttt{gath} output.

Setting the \texttt{-i} flag prevents control lines from being interpreted and causes them to be output literally.

A file name of \texttt{-} at any point refers to standard input, and a control line consisting of \texttt{'} is a logical EOF. Keywords may be defined by specifying a replacement string which is to be substituted for each occurrence of the keyword. Input may be collected directly from the terminal, with several alternatives for prompting. In fact, all of the special arguments and flags recognized by the \texttt{send} command are also recognized and treated identically by \texttt{gath}. Several of them only make sense in the context of submitting an RJE job.

\textit{Send}
\texttt{Send} is a command-level interface to the RJE subsystems. It allows the user to collect input from various sources in order to create a run stream consisting of card images, and submit this run stream for transmission to a host computer.

Possible sources of input to \texttt{send} are: ordinary files, standard input, the terminal, and the output of a command or shell file. Each source of input is treated as a virtual file, and no distinction is made based upon its origin. Typical input is an ASCII text file of the sort that is created by the editor \texttt{ed(1)}. An optional format specification appearing in the first line of a file (see \texttt{fspec(5)}) determines the settings according to which tabs are expanded into spaces. In addition, lines that begin with \texttt{'} are normally interpreted as commands controlling the execution of \texttt{send}. They may be used to set or reset flags, to define keyword substitutions, and to open new sources of input in the midst of the current source. Other text lines are translated one-for-one into card images of the run stream.

The run stream that results from this collection is treated as one job by the RJE subsystems. \texttt{Send} prints the card count of the run stream, and the queuer that is invoked prints the name of the temporary file that holds the job while it is awaiting transmission. The initial card of a job submitted to an IBM host must have a \texttt{//} in the first column. The initial card of a job submitted to a UNIVAC host must begin with a \texttt{"@RUN"} or \texttt{"run"}, etc. Any cards preceding these will be excised. If a host computer is not
specified before the first card of the runstream is ready to be sent, send will
select a reasonable default. In the case of an IBM job, all cards beginning
with /*$ will be excised from the runstream, because they are HASP com-
mand cards.

The arguments that send accepts are described below. An argument is
interpreted according to the first pattern that it matches. Preceding a
character with \ causes it to loose any special meaning it might otherwise
have when matching against an argument pattern.

. Close the current source.
- Open standard input as a new source.
+ Open the terminal as a new source.
:spec: Establish a default format specification for included sources,
e.g., :m6t-12:
:message Print message on the terminal.
-prompt Open standard input and, if it is a terminal, print prompt.
+prompt Open the terminal and print prompt.
-flags Set the specified flags, which are described below.
+flags Reset the specified flags.
=flags Restore the specified flags to their state at the pre-
vious level.
!command Execute the specified UNIX command via the one-
line shell, with input redirected to /dev/null as a
default. Open the standard output of the com-
mand as a new source.
Sline Collect contiguous arguments of this form and
write them as consecutive lines to a temporary
file; then have the file executed by the shell. Open the standard output of the shell as a new
source.
@directory The current directory for the send process is
changed to directory. The original directory will be
restored at the end of the current source.
~comment Ignore this argument.
?:keyword Prompt for a definition of keyword from the ter-
minal unless keyword has an existing definition.
?keyword=xx Define the keyword as a two digit hexadecimal
character code unless it already has a non null
replacement.
?keyword=string Define the keyword in terms of a replacement
string unless it already has a non null repla-
cement.
=keyword Prompt for a definition of keyword from the ter-
minal.
keyword=xx Define keyword as a two-digit hexadecimal charac-
ter code.
keyword = string
Define keyword in terms of a replacement string.

host
The host machine that the job should be submitted to. It can be any name that corresponds to one in the first column of the RJE configuration file (/usr/rje/lines).

file-name
Open the specified file as a new source of input.

When commands are executed via $ or ! the shell environment (see environ(7)) will contain the values of all send keywords that begin with $ and have the syntax of a shell variable.

The flags recognized by send are described in terms of the special processing that occurs when they are set:

- `-l` List card images on standard output. EBCDIC characters are translated back to ASCII.
- `-q` Do not output card images.
- `-f` Do not fold lower case to upper.
- `-t` Trace progress on diagnostic output, by announcing the opening of input sources.
- `-k` Ignore the keywords that are active at the previous level and erase any keyword definitions that have been made at the current level.
- `-r` Process included sources in raw mode; pack arbitrary 8-bit bytes one per column (80 columns per card) until an EOF.
- `-i` Do not interpret control lines in included sources; treat them as text.
- `-s` Make keyword substitutions before detecting and interpreting control lines.
- `-y` Suppress error diagnostics and submit job anyway.
- `-g` Gather mode, qualifying `-l` flag; list text lines before converting them to card images.
- `-h` Write listing with standard tabs.
- `-p` Prompt with * when taking input from the terminal.
- `-m` When input returns to the terminal from a lower level, repeat the prompt, if any.
- `-a` Make `-k` flag propagate to included sources, thereby protecting them from keyword substitutions.
- `-c` List control lines on diagnostic output.
- `-d` Extend the current set of keyword definitions by adding those active at the end of included sources.
- `-x` This flag guarantees that the job will be transmitted in the order of submission (relative to other jobs sent with this flag).

Control lines are input lines that begin with `-`. In the default mode `-ir`, they are interpreted as commands to send. Normally they are detected immediately and read literally. The `-s` flag forces keyword substitutions to be made before control lines are intercepted and interpreted. This can lead to unexpected results if a control line uses a keyword which is defined within an immediately preceding `$` sequence. Arguments appearing in control lines are handled exactly like the
command arguments to send, except that they are processed at a nested level of input.

The two possible formats for a control line are: ""argument"" and "" argument ...". In the first case, where the ' is not followed by a space, the remainder of the line is taken as a single argument to send. In the second case, the line is parsed to obtain a sequence of arguments delimited by spaces. In this case the quotes ' and " may be employed to pass embedded spaces.

The interpretation of the argument . is chosen so that an input line consisting of .. is treated as a logical EOF. The following example illustrates some of the above conventions:

```
send
  argument ...
  .
```

This sequence of three lines is equivalent to the command synopsis at the beginning of this description. In fact, the — is not even required. By convention, the send command reads standard input if no other input source is specified. Send may therefore be employed as a filter with side-effects.

The execution of the send command is controlled at each instant by a current environment, which includes the format specification for the input source, a default format specification for included sources, the settings of the mode flags, and the active set of keyword definitions. This environment can be altered dynamically. When a control line opens a new source of input, the current environment is pushed onto a stack, to be restored when input resumes from the old source. The initial format specification for the new source is taken from the first line of the file. If none is provided, the established default is used or, in its absence, standard tabs. The initial mode settings and active keywords are copied from the old environment. Changes made while processing the new source will not affect the environment of the old source, with one exception: if -d mode is set in the old environment, the old keyword context will be augmented by those definitions that are active at the end of the new source.

When send first begins execution, all mode flags are reset, and the values of the shell environment variables become the initial values for keywords of the same name with a $ prefixed.

The initial reset state for all mode flags is the + state. In general, special processing associated with a mode N is invoked by flag -N and is revoked by flag +N. Most mode settings have an immediate effect on the processing of the current source. Exceptions to this are the -r and -i flags, which apply only to included source, causing it to be processed in an uninterpreted manner.

A keyword is an arbitrary 8-bit ASCII string for which a replacement has been defined. The replacement may be another string, or (for IBM RJE only) the hexadecimal code for a single 8-bit byte. At any instant, a given set of keyword definitions is active. Input text lines are scanned, in one pass from left to right, and longest matches are attempted between substrings of the line and the active set of keywords. Characters that do not match are output, subject to folding and the standard translation. Keywords are replaced by the specified hexadecimal code or replacement string, which is then output character by character. The expansion of tabs and length checking, according to the format
specification of an input source, are delayed until substitutions have been made in a line.

All of the keywords definitions made in the current source may be deleted by setting the \(-k\) flag. It then becomes possible to reuse them. Setting the \(-k\) flag also causes keyword definitions active at the previous source level to be ignored. Setting the \(+k\) flag causes keywords at the previous level to be ignored but does not delete the definitions made at the current level. The \(-k\) argument reactivates the definitions of the previous level.

When keywords are redefined, the previous definition at the same level of source input is lost, however the definition at the previous level is only hidden, to be reactivated upon return to that level unless a \(-d\) flag causes the current definition to be retained.

Conditional prompts for keywords, \(?:A./p\) which have already been defined at some higher level to be null or have a replacement will simply cause the definitions to be copied down to the current level; new definitions will not be solicited.

Keyword substitution is an elementary macro facility that is easily explained and that appears useful enough to warrant its inclusion in the \texttt{send} command. More complex replacements are the function of a general macro processor (\texttt{m4}(1), perhaps). To reduce the overhead of string comparison, it is recommended that keywords be chosen so that their initial characters are unusual. For example, let them all be upper case.

\texttt{Send} performs two types of error checking on input text lines. Firstly, only ASCII graphics and tabs are permitted in input text. Secondly, the length of a text line, after substitutions have been made, may not exceed 80 bytes for IBM, or 132 bytes for UNIVAC. The length of each line may be additionally constrained by a size parameter in the format specification for an input source. Diagnostic output provides the location of each erroneous line, by line number and input source, a description of the error, and the card image that results. Other routine errors that are announced are the inability to open or write files, and abnormal exits from the shell. Normally, the occurrence of any error causes \texttt{send}, before invoking the queuer, to prompt for positive affirmation that the suspect run stream should be submitted.

For IBM hosts, \texttt{send} is required to translate 8-bit ASCII characters into their EBCDIC equivalents. The conversion for 8-bit ASCII characters in the octal range 040-176 is based on the character set described in "Appendix H" of \textit{IBM System/370 Principles of Operation} (IBM SRL GA22-7000). Each 8-bit ASCII character in the range 040-377 possesses an EBCDIC equivalent into which it is mapped, with five exceptions: \textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde

For UNIVAC hosts, on the other hand, operate in ASCII code, and any translations between ASCII and field-data are made, in accordance with the UNIVAC standard, by the host computer.

Additional control over the translation process is afforded by the \(-f\) flag and hexadecimal character codes. As a default, \texttt{send} folds lowercase letters into upper case. For UNIVAC RJE it does more: the entire ASCII range 0140-0176 is folded into 0100-0136, so that \textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciitilde\textasciиторм
example, becomes @. In either case, setting the \texttt{-f} flag inhibits any folding. Non-standard character codes are obtained as a special case of keyword substitution.

**SEE ALSO**

m4(1), orjestat(1C), rjestat(1C), sh(1), fspec(5), ascii(7), hasp(8), rje(8), uvac(8).


**BUGS**

Standard input is read in blocks, and unused bytes are returned via \texttt{lseek(2)}. If standard input is a pipe, multiple arguments of the form \texttt{-} and \texttt{-:prompt} should not be used, nor should the logical EOF (\texttt{^\textendash}).
NAME
setmnt — establish mnttab table

SYNOPSIS
/etc/setmnt

DESCRIPTION
Setmnt creates the /etc/mnttab table (see mnttab(5)), which is needed for both the mount(1M) and umount(1M) commands. Setmnt reads standard input and creates a mnttab entry for each line. Input lines have the format:

filesys node

where filesys is the name of the file system's special file (e.g., "rp??") and node is the root name of that file system. Thus filesys and node become the first two strings in the mnttab(5) entry.

FILES
/etc/mnttab

SEE ALSO
mnttab(5).

BUGS
Evil things will happen if filesys or node are longer than 10 characters. Setmnt silently enforces an upper limit on the maximum number of mnttab entries.
NAME
sh — shell, the standard command programming language

SYNOPSIS
sh [ -ceikrstuvx ] [ args ]

DESCRIPTION
Sh is a command programming language that executes commands read from a terminal or a file. See Invocation below for the meaning of arguments to the shell.

Commands.
A simple-command is a sequence of non-blank words separated by blanks (a blank is a tab or a space). 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.

A list is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does not wait for that pipeline to finish). The symbol && (||) causes the list following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a list, instead of semicolons, to delimit commands.

A command is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for name [ in word . . . ] do list done
Each time a for command is executed, name is set to the next word taken from the in word list. If in word . . . is omitted, then the for command executes the do list once for each positional parameter that is set (see Parameter Substitution below). Execution ends when there are no more words in the list.

case word in [ pattern [ | pattern ] . . . ] list ; ; ] . . . esac
A case command executes the list associated with the first pattern that matches word. The form of the patterns is the 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
A while command repeatedly executes the while list and, if the exit status of the last command in the list is zero, executes the do list;
otherwise the loop terminates. If no commands in the do list are executed, then the while command returns a zero exit status; until may be used in place of while to negate the loop termination test.

(list)  
Execute list in a sub-shell.

{list;}  
list is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments.
A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution.
The standard output from a command enclosed in a pair of grave accents (```) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution.
The character $ is used to introduce substitutable parameters. Positional parameters may be assigned values by set. Variables may be set by writing:

name = value [ name = value ] ...

Pattern-matching is not performed on value.

$[parameter]
A parameter is a sequence of letters, digits, or underscores (a name), a digit, or any of the characters *, @, #, ?, -, $, and !.
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. A name must begin with a letter or underscore. If parameter is a digit then it is a positional parameter. If parameter is * or @, then 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 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 the message "parameter null or not set" is printed.

$[parameter: +word]
If parameter is set and is non-null then substitute word; otherwise substitute nothing.

In the above, word is not evaluated unless it is to be used as the substituted string, so that, in the following example, pwd is executed only if d is not set or is null:

    echo ${d: = "pwd" }

If the colon (:) is omitted from the above expressions, then the shell only checks whether parameter is set or not.
The following parameters are automatically set by the shell:

# The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the set command.
? The decimal value returned by the last synchronously executed command.
$ The process number of this shell.
! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the cd command.
PATH The search path for commands (see Execution below).
MAIL If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.
PS1 Primary prompt string, by default "$".
PS2 Secondary prompt string, by default "">".
IFS Internal field separators, normally space, tab, and new-line.

The shell gives default values to PATH, PS1, PS2, and IFS, while HOME and MAIL are not set at all by the shell (although HOME is set by login(1)).

Blank Interpretation.

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in IFS) and split into distinct arguments where such characters are found. Explicit null arguments ("" or "") are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

File Name Generation.

Following substitution, each command word is scanned for the characters *, ?, and [. If one of these characters appears then the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. The character . at the start of a file name 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.

Quoting.

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | < > new-line space tab

A character may be quoted (i.e., made to stand for itself) by preceding it with a \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (""), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, ", " and $. "$s" is equivalent to "$1 $2 ...", whereas "$s@" is equivalent to "$1" "$2" ....

Prompting.

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a new-line is typed and further input is
needed to complete a command, then the secondary prompt (i.e., the value of PS2) is issued.

Input/Output.
Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a command and are not passed on to the invoked command; substitution occurs before word or digit is used:

<word Use file word as standard input (file descriptor 0).

>word Use file word as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length.

>>word Use file word as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[ - ]word The shell input is read up to a line that is the same as word, or to an end-of-file. The resulting document becomes the standard input. If any character of word is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unesaged) \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 >.

<& The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by & then the default standard input for the command is the empty file /dev/null. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Environment.
The environment (see environ(7)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the export command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in export commands.

The environment for any simple-command may be augmented by prefixing it with one or more assignments to parameters. Thus:

TERM=450 cmd args
and
(export TERM; TERM=450; cmd args)
are equivalent (as far as the above execution of `cmd` is concerned).
If the `-k` flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints `a=b c` and then `c`:

```
echo a=b c
set -k
echo a=b c
```

**Signals.**
The `INTERRUPT` and `QUIT` signals for an invoked command are ignored if the command is followed by `&`; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the `trap` command below).

**Execution.**
Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via `exec(2)`.

The shell parameter `PATH` defines the search path for the directory containing the command. Alternative directory names are separated by a colon (`:`). The default path is `:/bin:/usr/bin` (specifying the current directory, `/bin`, and `/usr/bin`, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a `/` then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an `a.out` file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

**Special Commands.**
The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

- `:` No effect; the command does nothing. A zero exit code is returned.
- `file` Read and execute commands from `file` and return. The search path specified by `PATH` is used to find the directory containing `file`.
- `break [ n ]` Exit from the enclosing `for` or `while` loop, if any. If `n` is specified then `break n` levels.
- `continue [ n ]` Resume the next iteration of the enclosing `for` or `while` loop. If `n` is specified then resume at the `n`-th enclosing loop.
- `cd [ arg ]` Change the current directory to `arg`. The shell parameter `HOME` is the default `arg`.
- `eval [ arg ... ]` The arguments are read as input to the shell and the resulting command(s) executed.
- `exec [ arg ... ]` The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.
- `exit [ n ]` Causes a shell to exit with the exit status specified by `n`. If `n` is
omitted then the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

**export [ name ... ]**
The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given, then a list of all names that are exported in this shell is printed.

**newgrp [ arg ... ]**
Equivalent to `exec newgrp arg ....`

**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 names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given, then a list of all readonly names is printed.

**set [ -ekntun [ arg ... ] ]**
- **e** If the shell is non-interactive then exit immediately if a command exits with a non-zero exit status.
- **k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **n** Read commands but do not execute them.
- **t** Exit after reading and executing one command.
- **u** Treat unset variables as an error when substituting.
- **v** Print shell input lines as they are read.
- **x** Print commands and their arguments as they are executed.
- **-** Do not change any of the flags; useful in setting $1 to .

Using + rather than − causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $- . The remaining arguments are positional parameters and are assigned, in order, to $1, $2, .... If no arguments are given then the values of all names are printed.

**shift**
The positional parameters from $2 ... are renamed $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 ] ...**  
arg is a command to be read and executed when the shell receives signal(s) n. (Note that arg is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If arg is absent then all trap(s) n are reset to their original values. If arg is the null string then this signal is ignored by the shell and by the commands it invokes. If n is 0 then the command arg is executed on exit from the shell. The `trap` command with no arguments prints a list of commands associated with each signal number.

**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.

**wait**  
Wait for all child processes to terminate report the termination status. If \( n \) is not given then all currently active child processes are waited for. The return code from this command is always zero.

**Invocation.**

If the shell is invoked through `exec(2)` and the first character of argument zero is `−`, commands are initially read from `/etc/profile` and then from `$HOME/.profile`, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as `/bin/sh`. The flags below are interpreted by the shell on invocation only; Note that unless the `−c` or `−s` flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

`−c string`  
If the `−c` flag is present then commands are read from `string`.

`−s`  
If the `−s` flag is present or if no arguments remain then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

`−i`  
If the `−i` flag is present or if the shell input and output are attached to a terminal, then this shell is *interactive*. In this case `TERMENATE` is ignored (so that `kill 0` does not kill an interactive shell) and `INTERRUPT` is caught and ignored (so that `wait` is interruptible). In all cases, `QUIT` is ignored by the shell.

`−r`  
If the `−r` flag is present the shell is a restricted shell (see `rsh(1)`).

The remaining flags and arguments are described under the `set` command above.

**EXIT STATUS**

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 then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the `exit` command above).

**FILES**

/`etc/profile`

`$HOME/.profile`

/tmp/sh*

/dev/null

**SEE ALSO**


**BUGS**

The command `readonly` (without arguments) produces the same output as the command `export`.

If `<<` is used to provide standard input to an asynchronous process invoked by `&`, the shell gets mixed up about naming the input document; a garbage file `/tmp/sh*` is created and the shell complains about not being able to find that file by another name.
NAME
shutdown — terminate all processing

SYNOPSIS
/etc/shutdown

DESCRIPTION
Shutdown is part of the UNIX operation procedures. Its primary function is to terminate all currently running processes in an orderly and cautious manner. The procedure is designed to interact with the operator (i.e., the person who invoked shutdown). Shutdown may instruct the operator to perform some specific tasks, or to supply certain responses before execution can resume. Shutdown goes through the following steps:

- All users logged on the system are notified to log off the system by a broadcasted message. The operator may display his/her own message at this time. Otherwise, the standard file save message is displayed.

- If the operator wishes to run the file-save procedure, shutdown unmounts all file systems.

- All file systems' super blocks are updated before the system is to be stopped (see sync(1M)). This must be done before re-booting the system, to insure file system integrity. The most common error diagnostic that will occur is device busy. This diagnostic happens when a particular file system could not be unmounted. See umount(1M).

SEE ALSO
sync(1M), umount(1M).
NAME
size — size of an object file

SYNOPSIS
size [ object ... ]

DESCRIPTION
Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. If no file is specified, a.out is used.

SEE ALSO
a.out(5).
NAME
sleep — suspend execution for an interval

SYNOPSIS
sleep time

DESCRIPTION
Sleep suspends execution for time seconds. It is used to execute a command after a certain amount of time as in:

(sleep 105; command)&

or to execute a command every so often, as in:

while true
do
  command
  sleep 37
done

SEE ALSO
alarm(2), sleep(3C).

BUGS
Time must be less than 65536 seconds.
NAME
sno — SNOBOL interpreter

SYNOPSIS
sno [ files ]

DESCRIPTION
Sno is a SNOBOL compiler and interpreter (with slight differences). Sno obtains input from the concatenation of the named files and the standard input. All input through a statement containing the label end is considered program and is compiled. The rest is available to syspit.

Sno differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

a •• b unanchored search for b.
a ••• b = x c unanchored assignment

There is no back referencing.

x = "abc" is an unanchored search for abc.

Function declaration is done at compile time by the use of the (non-unique) label define. Execution of a function call begins at the statement following the define. Functions cannot be defined at run time, and the use of the name define is preempted. There is no provision for automatic variables other than parameters. Examples:

define f( )
define f(a, b, c)

All labels except define (even end) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on end cannot merely name a label.

If start is a label in the program, program execution will start there. If not, execution begins with the first executable statement; define is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and • must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable sysppt is not available.

SEE ALSO
awk(1).
NAME
sort — sort and/or merge files

SYNOPSIS
sort [ -cmubdfinrtx ] [ +pos1 [ -pos2 ] ] ... [ -o output ]
[ names ]

DESCRIPTION
Sort sorts lines of all the named files together and writes the result on the
standard output. The name - means the standard input. If no input files
are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by
bytes in machine collating sequence. The ordering is affected globally by
the following options, one or more of which may appear.

b  Ignore leading blanks (spaces and tabs) in field comparisons.
d  "Dictionary" order: only letters, digits and blanks are significant in
   comparisons.
f  Fold upper case letters onto lower 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. Option n implies option b.
r  Reverse the sense of comparisons.
tx  "Tab character" separating fields is x.

The notation +pos1 -pos2 restricts a sort key to a field beginning at pos1
and ending just before pos2. Pos1 and pos2 each have the form m.n,
onitionally followed by one or more of the flags bdfinr, where m tells a
number of fields to skip from the beginning of the line and n tells a num-
ber of characters to skip further. If any flags are present they override all
the global ordering options for this key. If the b option is in effect n is
counted from the first non-blank in the field; b is attached independently to
pos2. A missing .n means .0; a missing -pos2 means the end of the line.
Under the -tx option, fields are strings separated by x; otherwise fields are
non-empty non-blank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all
erlier keys compare equal. Lines that otherwise compare equal are
ordered with all bytes significant.

These option arguments are also understood:
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  Suppress all but one in each set of equal lines. Ignored bytes and
   bytes outside keys do not participate in this comparison.
o  The next argument 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.

EXAMPLES
Print in alphabetical order all the unique spellings in a list of words (capit-
alyzed words differ from uncapitalized):
sort -u +0f +0 list
Print the password file (passwd(5)) sorted by user ID (the third colon-separated field):
    sort -t: +2n /etc/passwd
Print the first instance of each month in an already sorted file of (month-day) entries (the options -um with just one input file make the choice of a unique representative from a set of equal lines predictable):
    sort -um +0 -1 dates

FILES
/usr/tmp/stm???

SEE ALSO
    comm(1), join(1), uniq(1).

DIAGNOSTICS
    Comments and exits with non-zero status for various trouble conditions and for disorder discovered under option -c.

BUGS
    Very long lines are silently truncated.
SPELL(1)

NAME
spell, spellin, spellout — find spelling errors

SYNOPSIS
spell [ options ] [ files ]
/usr/lib/spell/spellin [ list ]
/usr/lib/spell/spellout [ -d ] list

DESCRIPTION
Spell collects words from the named files and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most troff(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, speciality, travelled, etc., this option insists upon -ise in words like standardise, Fowler and the OED to the contrary notwithstanding.

Under the -x option, every plausible stem is printed with = for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy—y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by spell (both expect a list of words, one per line, from the standard input): spellin adds the words on the standard input to the preexisting list and places a new list on the standard output. If no list is specified, the new list is created from scratch. Spellout looks up each word read from the standard input, and prints on the standard output those that are missing from (or, with the -d option, present in) the hash list.

FILES
D_SPELL=/usr/lib/spell/hlist[ab] hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop hashed stop list
H_SPELL=/usr/lib/spell/spellhist history file
/tmp/spell.$$ temporary
/usr/lib/spell/spellprog program

SEE ALSO
deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1), typo(1).

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 dictionary that is added to the hashed list via spellin.

British spelling was done by an American.
NAME
spline — interpolate smooth curve

SYNOPSIS
spline [ options ]

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., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by graph.

The following options are recognized, each as a separate argument:

- Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.
- k The constant k used in the boundary value computation:
  \( y_0 = ky_1, \quad y_n = ky_{n-1} \)
  is set by the next argument (default \( k = 0 \)).
- n Space output points so that approximately \( n \) intervals occur between the lower and upper \( x \) limits (default \( n = 100 \)).
- p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.
- x Next 1 (or 2) arguments are lower (and upper) \( x \) limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

SEE ALSO
graph.

DIAGNOSTICS
When data is not strictly monotone in \( x \), spline reproduces the input without interpolating extra points.

BUGS
A limit of 1,000 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), as many as necessary, onto a set of output files. The name of the first output file is name with aa appended, and so on lexicographically. If no output name is given, x is default.

If no input file is given, or if is given in its stead, then the standard input file is used.

SEE ALSO
bfs(1), csplit(1).
NAME
  st — synchronous terminal control

SYNOPSIS
  /etc/stload
  /etc/stcntrl [ on | off ]

DESCRIPTION
  The stload command file is used to load the synchronous terminal prototype
  script, /etc/proto, into the designated KMC11-B microprocessor, and start
  execution of the script. As supplied, stload uses /dev/kmc0; it may need
  local modification if another KMC11-B is being used.

  The stcntrl command is used to activate and deactivate the synchronous ter-
  minal driver.

  The /etc/rc file should contain the following multi-user entries:

  /etc/stload
  /etc/stcntrl on

  while /etc/shutdown should have:

  /etc/stcntrl off

FILES
  /etc/stproto  synchronous terminal prototype script
  /dev/kmc?  KMC11-B microprocessor
  /dev/vpm?  virtual protocol machine
  /dev/st0  synchronous terminal control channel
  /dev/st?  synchronous terminal user channels

SEE ALSO
  kmc(4), st(4), trace(4), vpm(4).

BUGS
  The stcntrl.c file assumes that /dev/vpm0 is the vpm device being used for
  the first (and usually only) synchronous terminal controller. If some other
  vpm device is being used, the stcntrl.c file must be modified and rebuilt.
NAME
stat — statistical network useful with graphical commands

SYNOPSIS
node-name [options] [files]

DESCRIPTION
Stat is a collection of command level functions (nodes) that can be interconnected using sh(1) to form a statistical network. The nodes reside in /usr/bin/graf (see graphics(1G)). Data is passed through the network as sequences of numbers (vectors), where a number is of the form:

[sign](digits)(.digits)[e[sign]digits]
evaluated in the usual way. Brackets and parentheses surround fields. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Any character input to a node that is not part of a number is taken as a delimiter.

Stat nodes are divided into four classes.

Transformers, which map input vector elements into output vector elements;

Summarizers, which calculate statistics of a vector;

Translators, which convert among formats; and

Generators, which are sources of definable vectors.

Below is a list of synopses for stat nodes. Most nodes accept options indicated by a leading minus (−). In general, an option is specified by a character followed by a value, such as c=5. This is interpreted as c := 5 (c is assigned 5). The following keys are used to designate the expected type of the value:

c characters,
i integer,
f floating point or integer,
file file name, and
string string of characters, surrounded by quotes to include a Shell argument delimiter.

Options without keys are flags. All nodes except generators accept files as input, hence it is not indicated in the synopses.

Transformers:

abs [−ci] — absolute value
columns (similarly for −c options that follow)
af [−ci t v ] — arithmetic function
titled output, verbose
ceil [−ci] — round up to next integer
cusum [−ci] — cumulative sum
exp [−ci] — exponential
floor [−ci] — round down to next integer
gamma [−ci] — gamma
list [−ci dstring] — list vector elements
delimiter(s)
log \ [-ci \ h\ f] \ - \ logarithm
base

mod \ [-ci \ m\ f] \ - \ modulus
modulus

pair \ [-ci \ F\ file \ x\ i] \ - \ pair \ elements
File \ containing \ base \ vector, \ x \ group \ size

power \ [-ci \ p\ f] \ - \ raise \ to \ a \ power
power

root \ [-ci \ r\ f] \ - \ take \ a \ root
root

round \ [-ci \ p\ i \ s\ i ] \ - \ round \ to \ nearest \ integer, .5 \ rounds \ to \ 1
places \ after \ decimal \ point, \ significant \ digits

siline \ [-ci \ i/ f/ n/ i s/ f] \ - \ generate \ a \ line \ given \ slope \ and \ intercept
intercept, \ number \ of \ positive \ integers, \ slope

sin \ [-ci] \ - \ sine

subset \ [-af/h f/ci \ F\ file \ i\ i /f/ n\ p\ p/ si \ t\ i] \ - \ generate \ a \ subset
above, \ below, \ File \ with \ master \ vector, \ interval, \ leave,
master \ contains \ element \ numbers \ to \ leave, \ master \ con­
tains \ element \ numbers \ to \ pick, \ pick, \ start, \ terminate

Summarizers:

bucket \ [-ai \ ci \ F\ file \ h\ f /i /f/ n/ i] \ - \ break \ into \ buckets
average \ size, \ File \ containing \ bucket \ boundaries, \ high, \ interval, \ low, \ number

cor \ [-F\ file] \ - \ correlation \ coefficient
File \ containing \ base \ vector

hilo \ [-h/ l/ o/ o\ x/ o/ y] \ - \ find \ high \ and \ low \ values
high \ only, \ low \ only, \ option \ form, \ option \ form \ with \ x
prepended, \ option \ form \ with \ y \ prepended

lreg \ [-F\ file /i/ o/ s] \ - \ linear \ regression
File \ containing \ base \ vector, \ intercept \ only, \ option \ form \ for
siline, \ slope \ only

mean \ [-f/ n/ i p/ f] \ - \ (trimmed) \ arithmetic \ mean
fraction, \ number, \ percent

point \ [-f/ n/ i p/ f s] \ - \ point \ from \ empirical \ cumulative \ density
function
fraction, \ number, \ percent, \ sorted \ input

prod \ - \ internal \ product

qsort \ [-c]\ - \ quick \ sort

rank \ - \ vector \ rank

total \ - \ sum \ total

var \ - \ variance

Translators:

bar \ [-a/ b/ f/ g/ r/ i/ wi/ x/ f/ xa/ y/ f/ ya/ yl/ yh/ f] \ - \ build \ a \ bar \ chart
suppress \ axes, \ bold, \ suppress \ frame, \ suppress \ grid, \ region,
width \ in \ percent, \ x \ origin, \ suppress \ x-axis \ label, \ y \ origin,
suppress \ y-axis \ label, \ y-axis \ lower \ bound, \ y-axis \ high
bound
hist \[ [-a \ b \ f \ g \ ri \ xf \ xa \ yf \ ya \ ylf \ yhf ] \] — build a histogram
suppress axes, bold, suppress frame, suppress grid, region,
x origin, suppress x-axis label, y origin, suppress y-axis
label, y-axis lower bound, y-axis high bound

label \[ [-b \ c \ Ffile \ h \ p \ ri \ xu \ y yr ] \] — label the axis of a GPS
file
bar chart input, retain case, label File, histogram input,
plot input, rotation, x-axis, upper x-axis, y-axis, right y-axis

pie \[ [-b \ o \ p \ pni \ ppi \ ri \ v \ xi \ yi ] \] — build a pie chart
bold, values outside pie, value as percentage(:=100), value
as percentage(:=i), draw percent of pie, region, no values,
x origin, y origin
Unlike other nodes, input is lines of the form
\[
[<i \ e \ f \ cc>] \text{ value [label]}
\]
ignore (don't draw) slice, explode slice, fill slice,
color slice \(c=(\text{ black, red, green, blue})\)

plot \[ [-a \ b \ cstring \ d \ f \ Ffile \ g \ m \ ri \ xf \ xa \ xif \ xhf \ xlf \ xni \ xt \]
\[ yf \ ya \ ylf \ yhf \ yli \ ylt \] — plot a graph
suppress axes, bold, plotting characters, disconnected,
suppress frame, File containing x vector, suppress grid,
mark points, region, x origin, suppress x-axis label, x
interval, x high bound, x low bound, number of ticks on
x-axis, suppress x-axis title, y origin, suppress y-axis label,
y interval, y high bound, y low bound, number of ticks on
y-axis, suppress y-axis title

title \[ [-b \ c \ Istring \ vstring \ ustring ] \] — title a vector or a GPS
title bold, retain case, lower title, upper title, vector title

Generators:

gas \[ [-ci \ if \ ni \ sf \ tf ] \] — generate additive sequence
interval, number, start, terminate

prime \[ [-ci \ hi \ li \ ni ] \] — generate prime numbers
high, low, number

rand \[ [-ci \ hf \ lf \ mf \ ni \ si ] \] — generate random sequence
high, low, multiplier, number, seed

RESTRICTIONS
Some nodes have a limit on the size of the input vector.

SEE ALSO
graphics(1G), gps(5).
NAME
strip — remove symbols and relocation bits

SYNOPSIS
strip name ...

DESCRIPTION
Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. This is useful to save space after a program has been debugged.

The effect of strip is the same as use of the -s option of ld.

If name is an archive file, strip will remove the local symbols from any a.out format files it finds in the archive. Certain libraries, such as those residing in /lib, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

FILES
/tmp/stm* temporary file

SEE ALSO
ld(1).
NAME
stty — set the options for a terminal

SYNOPSIS
stty [ -a ] [ -g ] [ options ]

DESCRIPTION
Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the -a option, it reports all of the option settings; with the -g option, it reports current settings in a form that can be used as an argument to another stty command. Detailed information about the modes listed in the first five groups below may be found in tty(4). Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes
parenb (-parenb) enable (disable) parity generation and detection.
parodd (-parodd) select odd (even) parity.
cs5 cs6 cs7 cs8 select character size (see tty(4)).
0 hang up phone line immediately.
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible (these are the speeds supported by the DH-11 interface).
hupcl (-hupcl) hang up (do not hang up) DATA-PHONE* 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.
clocal (-clocal) assume a line without (with) modem control.

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 tty(4)).
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.
igner (-igner) ignore (do not ignore) CR on input.
icrnl (-icrnl) map (do not map) CR to NL on input.
iucle (-iucle) 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 an ASCII DC3 and started by sending an ASCII DC1.
ixany (-ixany) allow any character (only DC1) to restart output.
ixoff (-ixoff) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes
opost (-opost) post-process output (do not post-process output; ignore all other output modes).
olcuc (-olcuc) map (do not map) lower-case alphabetics to upper case on output.
onlcr (-onlcr) map (do not map) NL to CR-NL on output.
ocrnl (-ocrnl) map (do not map) CR to NL on output.
onocr (-onocr)  do not (do) output CRs at column zero.
onlret (-onlret) on the terminal NL performs (does not perform) the CR function.
ofill (-ofill) use fill characters (use timing) for delays.
ofdel (-ofdel) fill characters are DELs (NULs).
cr0 cr1 cr2 cr3 select style of delay for carriage returns (see tty(4)).
nl0 nl1 select style of delay for line-feeds (see tty(4)).
tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see tty(4)).
bs0 bs1 select style of delay for backspaces (see tty(4)).
ff0 ff1 select style of delay for form-feeds (see tty(4)).
vt0 vt1 select style of delay for vertical tabs (see tty(4)).

Local Modes

isig (-isig) enable (disable) the checking of characters against the special control characters INTR and QUIT.
icanon (-icanon) enable (disable) canonical input (ERASE and KILL processing).
xcase (-xcase) canonical (unprocessed) 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-backspace-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does not keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (-echok) echo (do not echo) NL after KILL character. the same as echok (-echok); obsolete.
echof (-echof) echo (do not echo) NL.

Control Assignments

control-character c set control-character to c, where control-character is erase, kill, intr, quit, eof, eol, min, or time (min and time are used with -icanon; see tty(4)). If c is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "^d" is a CTRL-d); "?" is interpreted as DEL and "-" is interpreted as undefined. set line discipline to i (0 < i < 127).

line i

Combination Modes
even or parity enable parenb and cs7.
odd parity, -evenp, or -oddp enable parenb, cs7, and parodd.
disable parenb, and set cs8.

raw (-raw or cooked) enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).
nl (-nl) unset (set) icrnl, onlcr. In addition -nl unsets inler, igncr, ocrl, and onlret.

lcase (-lcase) set (unset) xcase, icue, and olcuc.
LCASE (-LCASE) same as lcase (-lcase).
tabs (-tabs or tab3) preserve (expand to spaces) tabs when printing.
ek reset ERASE and KILL characters back to normal # and @.
sane resets all modes to some reasonable values.
set all modes suitable for the terminal type `term`, where `term` is one of tty33, tty37, vt05, tn300, ti700, or tek.

SEE ALSO
tabs(1), ioctl(2), tty(4).
NAME

su — become super-user or another user

SYNOPSIS

su [ - ] [ name [ arg ... ] ]

DESCRIPTION

Su allows one to become another user without logging off. The default
user name is root (i.e., super-user).

To use su, the appropriate password must be supplied (unless one is
already super-user). If the password is correct, su will execute a new shell
with the user ID set to that of the specified user. To restore normal user ID
privileges, type an EOF to the new shell.

Any additional arguments are passed to the shell, permitting the super-user
to run shell procedures with restricted privileges (an arg of the form -c
string executes string via the shell). When additional arguments are passed,
/bin/sh is always used. When no additional arguments are passed, su uses
the shell specified in the password file.

An initial - flag causes the environment to be changed to the one that
would be expected if the user actually logged in again. This is done by
invoking the shell with an arg0 of -su causing the .profile in the home
directory of the new user ID to be executed. Otherwise, the environment is
passed along with the possible exception of $PATH, which is set to
/bin:/etc:/usr/bin for root. Note that the .profile can check arg0 for -sh
or -su to determine how it was invoked.

FILES

/etc/passwd system's password file
$HOME/.profile user's profile

SEE ALSO

env(1), login(1), sh(1), environ(7).
NAME
   sum — sum and count blocks in 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 check-
   sum.

SEE ALSO
   wc(1).

DIAGNOSTICS
   "Read error" is indistinguishable from end of file on most devices; check
   the block count.
NAME
 sync — update the super block

SYNOPSIS
 sync

DESCRIPTION
 Sync executes the sync system primitive. If the system is to be stopped, 
 sync must be called to insure file system integrity. See sync(2) for details.

SEE ALSO
 sync(2).
NAME
sysdef — system definition

SYNOPSIS
/etc/sysdef [opsys [master]]

DESCRIPTION
Sysdef analyzes the named operating system file and extracts configuration information. This includes all hardware devices, their addresses, interrupt vectors and unit count, as well as system devices and all tunable parameters.

The output of sysdef can be used directly by config(1M) to regenerate the appropriate low.s (univec.c on the VAX-11/780) and conf.c configuration files.

FILES
/unix         default operating system file
/etc/master   default table for hardware specifications

DIAGNOSTICS
"unknown device interrupts at vector xxx" if information regarding the device cannot be found in the master table.

SEE ALSO
config(1M), master(5).

BUGS
As yet, sysdef knows nothing of devices that are not interrupt driven. Because information regarding config aliases is not preserved by the system, device names returned might not be accurate.
NAME

tabs — set tabs on a terminal

SYNOPSIS

tabs [ tabspec ] [ +nm ] [ -Type ]

DESCRIPTION

Tabs sets the tab stops on the user’s terminal according to the tab specification tabspec, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the left margin on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for tabspec: "canned,“ repetitive, arbitrary, and file. If no tabspec is given, the default value is -8, i.e., UNIX "standard" tabs. The lowest column number is 1. Note that for tabs, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code
gives the name of one of a set of "canned" 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:

  <:<t--c2 m6 s66 d:>

- c3
  1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
  COBOL compact format (columns 1-6 omitted), with more tabs than -c2. This is the recommended format for COBOL. The appropriate format specification is:

  <:<t--c3 m6 s66 d:>

- f
  1,7,11,15,19,23
  FORTRAN

- p
  1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
  PL/I

- s
  1,10,55
  SNOBOL

- u
  1,12,20,44
  UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

- n
  A repetitive specification requests tabs at columns 1+n, 1+2n, etc. Note that such a setting leaves a left margin of n columns on TermiNet terminals only. Of particular importance is the value
-8: this represents the UNIX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the nroff(1) -h option for high-speed output. Another special case is the value -0, implying no tabs at all.

\[ n1,n2,\ldots \]

The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

\[ \text{--file} \]

If the name of a file is given, tabs reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as -8. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the pr(1) command:

\[
\text{tabs -- file; pr file}
\]

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

\[ \text{--Ttype Tabs} \]

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(7). If no --T flag is supplied, tabs searches for the $TERM value in the environment (see environ(7)). If no type can be found, tabs tries a sequence that will work for many terminals.

\[ +m \]

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.

**DIAGNOSTICS**

\[ \text{illegal tabs} \]

when arbitrary tabs are ordered incorrectly.

\[ \text{illegal increment} \]

when a zero or missing increment is found in an arbitrary specification.

\[ \text{unknown tab code} \]

when a "canned" code cannot be found.

\[ \text{can't open} \]

if --file option used, and file can't be opened.

\[ \text{file indirection} \]

if --file option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

**SEE ALSO**

nroff(1), environ(7), term(7).

**BUGS**

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin. It is generally impossible to usefully change the left margin without also setting tabs. Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.
NAME
tail — deliver the last part of a file

SYNOPSIS
tail [ ±[number][lbe] [ -f ] ] [ file ]

DESCRIPTION
Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +number from the beginning, or −number from the end of the input (if number is null, the value 10 is assumed). Number is counted in units of lines, blocks, or characters, according to the appended option l, b, or c. When no units are specified, counting is by lines.

With the −f ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

tail −f fred

will print 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.

SEE ALSO
dd(1).

BUGS
Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.
NAME
tar — tape file archiver

SYNOPSIS
tar [ key ] [ files ]

DESCRIPTION
Tar saves and restores files on magnetic tape. Its actions are controlled by
the key argument. The key is a string of characters containing at most one
function letter and possibly one or more function modifiers. Other
arguments to the command are files (or directory names) specifying which
files are to be dumped or restored. In all cases, appearance of a directory
name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the end of the tape. The c function
implies this function.

x 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. The owner,
modification time, and mode are restored (if possible). 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 names of the specified files are listed each time that they
occur on the tape. If no files argument is given, all the names on
the tape are listed.

u The named files are added to the tape if they are not already there,
or have been modified since last written on that tape.

c Create a new tape; writing begins at the beginning of the tape,
instead of after the last file. This command implies the r function.

The following characters may be used in addition to the letter that selects
the desired function:

0,...,7 This modifier selects the drive on which the tape is mounted. The
default is 1.

v Normally, tar does its work silently. The v (verbose) option
causes it to type the name of each file it treats, preceded by the
function letter. With the t function, v gives more information
about the tape entries than just the name.

w causes tar to print the action to be taken, followed by the name of
the file, and then wait for the user’s confirmation. If a word
beginning with y is given, the action is performed. Any other
input means “no”.

f causes tar to use the next argument as the name of the archive
instead of /dev/mt?. 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 pipel-
line. Tar can also be used to move hierarchies with the command:

cd fromdir; tar cf - . | (cd todir; tar xf -)

b causes tar to use the next argument as the blocking factor for tape
records. The default is 1, the maximum is 20. This option should
only be used with raw magnetic tape archives (see f above). The
block size is determined automatically when reading tapes (key let-
ters x and t).

l 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 tells `tar` to not restore the modification times. The modification time of the file will be the time of extraction.

**FILES**

`/dev/mt?`

`/tmp/tar*`

**DIAGNOSTICS**

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

**BUGS**

There is no way to ask for the n-th occurrence of a file.
Tape errors are handled ungracefully.
The `u` option can be slow.
The `b` option should not be used with archives that are going to be updated.
The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the `b` option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on file-name length is 100 characters.
NAME
tbl — format tables for nroff or troff

SYNOPSIS
tbl [ -TX ] [ files ]

DESCRIPTION
_Tbl_ is a preprocessor that formats tables for _nroff(1)_ or _troff(1)._ The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and are re-formatted by _tbl_. (The .TS and .TE command lines are not altered by _tbl_).

.TS is followed by global options. The available global options are:

- **center** center the table (default is left-adjust);
- **expand** make the table as wide as the current line length;
- **box** enclose the table in a box;
- **doublebox** enclose the table in a double box;
- **allbox** enclose each item of the table in a box;
- **tab (x)** use the character _x_ instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi-colon (;).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes all remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc. The available key-letters are:

- **c** center item within the column;
- **r** right-adjust item within the column;
- **l** left-adjust item within the column;
- **n** numerically adjust item in the column: units positions of numbers are aligned vertically;
- **s** span previous item on the left into this column;
- **a** center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
- **-** span down previous entry in this column;
- **=** replace this entry with a horizontal line;
- **=** replace this entry with a double horizontal line.

The characters _B_ and _I_ stand for the bold and italic fonts, respectively; the character | indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by .TE. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only _ _ or _=_, a single or double line, respectively, is drawn across the table at that point; if a _single item_ in a data line consists of only _ _ or _=_, then that item is replaced by a single or double line.

Full details of all these and other features of _tbl_ are given in the reference manual cited below.

The -TX option forces _tbl_ to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).
If no file names are given as arguments, `tbl` reads the standard input, so it may be used as a filter. When it is used with `eqn(1)` or `neqn(1), tbl` should come first to minimize the volume of data passed through pipes.

**EXAMPLE**

If we let `←` represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cI | cI s
^ | c c
l | n n .
Household Population
-
Town—Households
—Number—Size
=
Bedminster—789—3.26
Bernards Twp.—3087—3.74
Bernardsville—2018—3.30
Bound Brook—3425—3.04
Bridgewater—7897—3.81
Far Hills—240—3.19
.TE
```

yields:

<table>
<thead>
<tr>
<th>Household Population</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Town</strong></td>
</tr>
<tr>
<td><strong>Number</strong></td>
</tr>
<tr>
<td>Bedminster</td>
</tr>
<tr>
<td>Bernards Twp.</td>
</tr>
<tr>
<td>Bernardsville</td>
</tr>
<tr>
<td>Bound Brook</td>
</tr>
<tr>
<td>Bridgewater</td>
</tr>
<tr>
<td>Far Hills</td>
</tr>
</tbody>
</table>

**SEE ALSO**

*TBL—A Program to Format Tables* by M. E. Lesk

eqn(1), mm(1), mmt(1), troff(1), mm(7), mv(7).

**BUGS**

See **BUGS** under `troff(1)`. 
NAME
tc — phototypesetter simulator

SYNOPSIS

tc [ -t ] [ -sn ] [ -pl ] [ file ]

DESCRIPTION
Tc interprets its input (standard input default) as device codes for a Wang
Laboratories, Inc. C/A/T phototypesetter. The standard output of tc is
intended for a Tektronix 4014 terminal with ASCII and APL character sets.
The sixteen typesetter sizes are mapped into the 4014's four sizes; the
entire TROFF character set is drawn using the 4014's character generator,
with overstruck combinations where necessary. Typical usage is:

troff -t files | tc

At the end of each page, tc waits for a new-line (empty line) from the key­
board before continuing on to the next page. In this wait state, the com­
mand e will suppress the screen erase before the next page; sn will cause
the next n pages to be skipped; and !cmd will send cmd to the shell.

The command line options are:

- t  Don't wait between pages (for directing output into a file).
- sn  Skip the first n pages.
- pl  Set page length to l; l may include the scale factors p (points), i
     (inches), c (centimeters), and P (picas); default is picas.

SEE ALSO
4014(1), sh(1), tplot(1G), troff(1).

BUGS
Font distinctions are lost.
NAME
tee — pipe fitting

SYNOPSIS
    tee [ −i ] [ −a ] [ file ] ...

DESCRIPTION
    Tee transcribes the standard input to the standard output and makes copies in the files. The −i option ignores interrupts; the −a option causes the output to be appended to the files rather than overwriting them.
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 fol­
    lowing primitives are used to construct expr:
    
    -r file      true if file exists and is readable.
    -w file      true if file exists and is writable.
    -x file      true if file exists and is executable.
    -f file      true if file exists and is a regular file.
    -d file      true if file exists and is a directory.
    -c file      true if file exists and is a character special file.
    -b file      true if file exists and is a block special file.
    -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 sl        true if the length of string sl is zero.
    -n sl        true if the length of the string sl is non-zero.
    sl = s2      true if strings sl and s2 are identical.
    sl != s2     true if strings sl and s2 are not identical.
    sl          true if sl is not the null string.
    nl -eq n2    true if the integers nl and n2 are algebraically equal. Any of
    the comparisons -ne, -gt, -ge, -lt, and -le may be used
    in place of -eq.

    These primaries may be combined with the following operators:
    
    !             unary negation operator.
    -a            binary and operator.
    -o            binary or operator (-a has higher precedence than -o).
    ( expr )      parentheses for grouping.

    Notice that all the operators and flags are separate arguments to test.
    Notice also that parentheses are meaningful to the shell and, therefore,
    must be escaped.

SEE ALSO
    find(1), sh(1).

WARNING
    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
  time — time a command

SYNOPSIS
  time command

DESCRIPTION
  The given command is executed; after it is complete, time prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

  The execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

  The times are printed on standard error.

SEE ALSO
  timex(1), times(2).
NAME
timex — time a command and generate a system activity report

SYNOPSIS
timex command

DESCRIPTION
The given command is executed; after its execution, timex prints the elapsed time, the time spent executing command, and the time spent in the system, as time(1) does. It also reports system activity that occurred during command execution, including CPU utilization, I/O activity, system switching and swapping, and file system access. All system activity is reported, not just that due to command.

The output of timex is written on standard error.

SEE ALSO
time(1), sar(8).
NAME
toc — graphical table of contents routines

SYNOPSIS
dtoc [directory]
ttoc mm-file
vtoc [-cbnimsvn] [ITOC file]

DESCRIPTION
All of the commands listed below reside in /usr/bin/graf (see graphics(1G)).

dtoc
Dtoc makes a textual table of contents, TTOC, of all subdirectories beginning at directory (directory defaults to .). The list has one entry per directory. The entry fields from left to right are level number, directory name, and the number of ordinary readable files in the directory. Dtoc is useful in making a visual display of all or parts of a file system. The following will make a visual display of all the readable directories under /:

dtoc / | vtoc | td

vtoc
Output is the table of contents generated by the .TC macro of mm(1) translated to TTOC format. The input is assumed to be a mm file that uses the .H family of macros for section headers. If no file is given, the standard input is assumed.

v toe
Vtoc produces a GPS describing a hierarchy chart from a TTOC. The output drawing consists of boxes containing text connected in a tree structure. If no file is given, the standard input is assumed. Each TTOC entry describes one box and has the form:

id [line-weight,line-style] "text" [mark]

where:

id is an alternating sequence of numbers and dots. The id specifies the position of the entry in the hierarchy. The id 0. is the root of the tree.

line-weight is either:
  a, normal-weight; or
  m, medium-weight; or
  b, bold-weight.

line-style is either:
  so, solid-line;
  do, dotted-line;
  dd, dot-dash line;
  da, dashed-line; or
  ld, long-dashed

text is a character string surrounded by quotes. The characters between the quotes become the contents of the box. To include a quote within a box it must be escaped (\").

mark is a character string (surrounded by quotes if it contains spaces), with included dots being escaped. The string is put above the top right corner of the box. To include either a quote or a dot within a mark it must be escaped.

Entry example: 1.1 b,da "ABC" DEF
Entries may span more than one line by escaping the new-line
Comments are surrounded by the /*,*/ pair. They may appear anywhere in a TTOC.

Options:
- **c**: Use text as entered, (default is all upper case).
- **hn**: Horizontal interbox space is \( n \% \) of box width.
- **i**: Suppress the box id.
- **m**: Suppress the box mark.
- **s**: Do not compact boxes horizontally.
- **vn**: Vertical interbox space is \( n \% \) of box height.

**SEE ALSO**
- graphics(1G), gps(5).
NAME
touch — update access and modification times of a file

SYNOPSIS
touch [ -ame ] [ mmddhhmm[yy] ] files

DESCRIPTION
Touch causes the access and modification times of each argument to be updated. If no time is specified (see date(1)) the current time is used. The -a and -m options cause touch to update only the access or modification times respectively (default is -am). The -c option silently prevents touch from creating the file if it did not previously exist.

The return code from touch is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO
date(1), utime(2).
NAME

tp — manipulate tape archive

SYNOPSIS

    tp [ key ] [ name ... ]

DESCRIPTION

Tp saves and restores files on DECTape or other magnetic tape. Its actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) sub-directories of that directory.

The function portion of the key is specified by one of the following letters:

r  The named files are written on the tape. If files with the same names already exist, they are replaced. "Same" is determined by string comparison, so ./abc can never be the same as /usr/sbo/abc even if /usr/sbo is the current directory. If no file argument is given, . is the default.

u  Updates the tape. u is like r, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. u is the default command if none is given.

d  Deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magnetic tapes.

x  Extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t  Lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

m  Specifies magnetic tape as opposed to DECTape.

0,...,7  This modifier selects the drive on which the tape is mounted. For DECTape, x is default; for magnetic tape 0 is the default.

v  Normally tp does its work silently. The v (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the t function, v gives more information about the tape entries than just the name.

c  Means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with r and u. This option is assumed with magnetic tape since it is impossible to selectively overwrite magnetic tape.

i  Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f  Use the first named file, rather than a tape, as the archive. This option is known to work only with x.

w  Causes tp to pause before treating each file, type the indicative letter and the file name (as with v) and await the user's response. Response y means "yes", so the file is treated. Null
response means "no", and the file does not take part in whatever is being done. Response x means "exit"; the tp command terminates immediately. In the x function, files previously asked about have been extracted already. With r, u, and d no change has been made to the tape.

FILES
/dev/tap?
/dev/mt?

SEE ALSO
ar(1), cpio(1), tar(1).

DIAGNOSTICS
Several; the non-obvious one is "Phase error", which means the file changed after it was selected for dumping but before it was dumped.

BUGS
A single file with several links to it is treated like several files.
Binary-coded control information makes magnetic tapes written by tp difficult to carry to other machines; tar(1) avoids the problem.
Tp does not copy zero-length files to tape.
NAME
tplot — graphics filters

SYNOPSIS
tplot [ -Tterminal [ -e raster ] ]

DESCRIPTION
These commands read plotting instructions (see plot(5)) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular terminal. If no terminal is specified, the environment parameter STERM (see environ(7)) is used. Known terminals are:

300 DASI 300.
300S DASI 300s.
450 DASI 450.
4014 Tektronix 4014.
ver Versatec D1200A. This version of plot places a scan-converted image in /usr/tmp/raster$$ and sends the result directly to the plotter device, rather than to the standard output. The -e option causes a previously scan-converted file raster to be sent to the plotter.

FILES
/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

SEE ALSO
plot(3X), plot(5), term(7).
NAME
  tr — translate characters

SYNOPSIS
  tr [ -cds ] [ string1 [ string2 ] ]

DESCRIPTION
  Tr copies the standard input to the standard output with substitution or
deletion of selected characters. Input characters found in string1 are mapped
into the corresponding characters of string2. Any combination of the
options -cds may be used:

  -c Complements the set of characters in string1 with respect to the
    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 that are in
    string2 to single characters.

The following abbreviation conventions may be used to introduce ranges of
characters or repeated characters into the strings:

  [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 con-
    sidered octal; otherwise, n is taken to be decimal. A zero or missing
    n is taken to be huge; this facility is useful for padding
    string2.

The escape character \ may be used as in the shell to remove special mean-
ning from any character in a string. In addition, \ followed by 1, 2, or 3
octal digits stands for the character whose ASCII code is given by those
digits.

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 newline.

  tr -cs "[A-Z][a-z]" "[\012]" <file1 >file2

SEE ALSO
  ed(1), sh(1), ascii(7).

BUGS
  Won't handle ASCII NUL in string1 or string2; always deletes NUL from
  input.
NAME
  troff, nroff — typeset or format text

SYNOPSIS
  nroff [ options ] [ files ]
  troff [ options ] [ files ]

DESCRIPTION
  Nroff formats text contained in files (standard input by default) for printing
  on typewriter-like devices and line printers; similarly, troff formats text for
  a Wang Laboratories, Inc., C/A/T phototypesetter. Their capabilities are
described in the NROFF/TROFF User's Manual cited below.

  An argument consisting of a minus (-) is taken to be a file name
  corresponding to the standard input. The options, which may appear in any
  order, but must appear before the files, are:

  -olist      Print only pages whose page numbers appear in the list of num-
              bers and ranges, separated by commas. A range N–M means
              pages N through M; an initial −N means from the beginning to
              page N; and a final −N means from N to the end. (See BUGS
              below.)

  -nN         Number first generated page N.

  -sN         Stop every N pages. Nroff will halt after every N pages (default
              N=1) to allow paper loading or changing, and will resume upon
              receipt of a line-feed or new-line (new-lines do not work in
              pipelines, e.g., with mm(1)). This option does not work if the
              output of nroff is piped through col(1). Troff will stop the
              phototypesetter every N pages, produce a trailer to allow changing
              cassettes, and resume when the typesetter's start button is
              pressed. When nroff (troff) halts between pages, an ASCII BEL
              (in troff, the message page stop) is sent to the terminal.

  -raN        Set register a (which must have a one-character name) to N.

  -i          Read standard input after files are exhausted.

  -q          Invoke the simultaneous input-output mode of the .rd request.

  -z          Print only messages generated by .tm (terminal message)
              requests.

  -mname      Prepend to the input files the non-compacted (ASCII text) macro
              file /usr/lib/tmac/tmac.name.

  -cname      Prepend to the input files the compacted macro files
              /usr/lib/macros/cmp.[nt].[dt].name and
              /usr/lib/macros/ucmp.[ot].name.

  -kname      Compact the macros used in this invocation of nroff/troff, placing
              the output in files [dt].name in the current directory (see the
              May 1979 Addendum to the NROFF/TROFF User's Manual for
details of compacting macro files).

Nroff only:

  -Tname      Prepare output for specified terminal. Known names are 37 for
              the (default) TELETYPE® Model 37 terminal, tn300 for the GE
              TermiNet 300 (or any terminal without half-line capability),
              300s for the DASI 300s, 300 for the DASI 300, 450 for the DASI
              450, Ip for a (generic) ASCII line printer, 382 for the DTC-382,
              4000A for the Trendata 4000A, 832 for the Anderson Jacobson
              832, X for a (generic) EBCDIC printer, and 2631 for the Hewlett
              Packard 2631 line printer.

  -e          Produce equally-spaced words in adjusted lines, using the full
              resolution of the particular terminal.
Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

Set the emboldening factor (number of character overstrikes) for the third font position (bold) to \text{n}, or to zero if \text{n} is missing.

**Troff only:**
- \text{t} Direct output to the standard output instead of the phototypesetter.
- \text{f} Refrain from feeding out paper and stopping phototypesetter at the end of the run.
- \text{w} Wait until phototypesetter is available, if it is currently busy.
- \text{b} Report whether the phototypesetter is busy or available. No text processing is done.
- \text{a} Send a printable ASCII approximation of the results to the standard output.
- \text{pN} Print all characters in point size \text{N} while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
- \text{g} Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output (see \text{gcat(1C)}). This option is not compatible with the \text{-s} option; furthermore, when this option is invoked, all .fp (font position) requests (if any) in the \text{troff} input must come before the first break, and no .t1 requests may come before the first break.
- \text{-Tname} Use font-width tables for device \text{name} (the font tables are found in \text{/usr/lib/font/name/*}). Currently, no \text{names} are supported.

**FILES**
- \text{/usr/lib/suftab} suffix hyphenation tables
- \text{/tmp/tas#} temporary file
- \text{/usr/lib/tmac/tmac.*} standard macro files and pointers
- \text{/usr/lib/macros/*} standard macro files
- \text{/usr/lib/term/*} terminal driving tables for \text{nroff}
- \text{/usr/lib/font/*} font width tables for \text{troff}

**SEE ALSO**
A TROFF Tutorial by B. W. Kernighan.
eqn(1), tbl(1), mm(7).
col(1), greek(1), mm(1) (nroff only).
gcat(1C), mmt(1), tc(1), mv(7) (troff only).

**BUGS**
Nroff/troff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that nroff/troff generates may be off by one day from your idea of what the date is.
When nroff/troff is used with the \text{-olist} option inside a pipeline (e.g., with one or more of \text{cw(1), eqn(1), and tbl(1)}), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in \text{list}. 
**NAME**

true, false — provide truth values

**SYNOPSIS**

ture

false

**DESCRIPTION**

*True* does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true do
    command
done
```

**SEE ALSO**

*sh*(1).

**DIAGNOSTICS**

*True* has exit status zero, *false* nonzero.
NAME
tsort — topological sort

SYNOPSIS
tsord [ file ]

DESCRIPTION
Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input file. If no file is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO
lorder(1).

DIAGNOSTICS
Odd data: there is an odd number of fields in the input file.

BUGS
Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.
NAME
	tty — get the terminal’s name

SYNOPSIS

tty [ -s ]

DESCRIPTION
	Tty prints the path name of the user’s terminal. The -s option inhibits
	printing, allowing one to test just the exit code.

EXIT CODES

0 if standard input is a terminal,
1 otherwise.

DIAGNOSTICS

“not a tty” if the standard input is not a terminal and -s is not specified.
NAME

typo — find possible typographical errors

SYNOPSIS

typo [ -n ] [ files ]

DESCRIPTION

Typo hunts through a document for unusual words, typographic errors, and hapax legomena and prints them on the standard output.

The words used in the document are printed out in decreasing order of peculiarity along with an index of peculiarity. An index of 10 or more is considered peculiar. Printing of certain very common English words is suppressed.

The statistics for judging words are taken from the document itself, with some help from known statistics of English. The -n option suppresses the help from English and should be used if the document is written in, for example, Urdu.

Troff(1) control lines are ignored. Quote marks, vertical bars, hyphens, and ampersands within words are equivalent to spaces. Words hyphenated across lines are put back together.

FILES

/tmp/ttmp??
/usr/lib/salt
/usr/lib/w2006

SEE ALSO

spell(1).
NAME
umask — set file-creation mode mask

SYNOPSIS
umask [ ooo ]

DESCRIPTION
The user file-creation mode mask is set to ooo. The octal three digits refer to read/write/execute permissions for owner, group, and others, respectively (see chmod(2) and umask(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see creat(2)). For example, umask 022 removes group and others write permission (files normally created with mode 777 become mode 755; files created 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.

SEE ALSO
chmod(1), sh(1), chmod(2), creat(2), umask(2).
NAME
uname — print name of current UNIX

SYNOPSIS
uname [ -snrv ]

DESCRIPTION
Uname prints the current system name of UNIX on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by `uname(2)` to be printed:

- `-s` print the system name (default).
- `-n` print the nodename (the nodename may be a name that the system is known by to a communications network).
- `-r` print the operating system release.
- `-v` print the operating system version.
- `-a` print all the above information.

SEE ALSO
uname(2).
NAME
unget — undo a previous get of an SCCS file

SYNOPSIS
unget [-rSID] [-s] [-n] files

DESCRIPTION
Unget undoes the effect of a get -e done prior to creating the intended new delta. If a directory is named, unget behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of - is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

- rSID Uniquely identifies which delta is no longer intended. (This would have been specified by get as the "new delta"). The use of this keyletter is necessary only if two or more outstanding gets for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified SID is ambiguous, or if it is necessary and omitted on the command line.

- s Suppresses the printout, on the standard output, of the intended delta's SID.

- n Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO
delta(1), get(1), sact(1).

DIAGNOSTICS
Use help(1) for explanations.
NAME
uniq — report repeated lines in a file

SYNOPSIS
uniq [-ude [ +n ] [ -n ] ] [ input [ output ] ]

DESCRIPTION
Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Input and output should always be different. Note that repeated lines must be adjacent in order to be found; see sort(1). If the -u flag is used, just the lines that are not repeated in the original file are output. The -d option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the -u and -d mode outputs.

The -e option supersedes -u and -d and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The n arguments specify skipping an initial portion of each line in the comparison:
- n The first n fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.
+ n The first n characters are ignored. Fields are skipped before characters.

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 only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi ratio of circumference to diameter,
c speed of light,
e charge on an electron,
g acceleration of gravity,
force same as g,
mole Avogadro's number,
water pressure head per unit height of water,
au astronomical unit.

Pound is not recognized as a unit of mass; lb is. Compound names are run together, (e.g. light year). 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
NAME
uuclean — uucp spool directory clean-up

SYNOPSIS
uuclean [ options ] ...

DESCRIPTION
Uuclean will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

- adirectory
  Clean directory instead of the spool directory.

- ppre
  Scan for files with pre as the file prefix. Up to 10 -p arguments may be specified. A -p without any pre following will cause all files older than the specified time to be deleted.

- ntime
  Files whose age is more than time hours will be deleted if the prefix test is satisfied. (default time is 72 hours)

- m
  Send mail to the owner of the file when it is deleted.

This program will typically be started by cron(1M).

FILES
/usr/lib/uucp directory with commands used by uuclean internally
/usr/spool/uucp spool directory

SEE ALSO
uucp(1C), uux(1C).
NAME
uucp, uulog, uuname — unix to unix copy

SYNOPSIS
uucp [ option ] ... source-file ... destination-file
uulog [ option ] ...
uname

DESCRIPTION
Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where system-name is taken from a list of system names which uucp knows about. Shell metacharacters ?*[] appearing in path-name will be expanded on the appropriate system.

Path names may be one of:
(1) a full path name;
(2) a path name preceded by ~user where user is a login name on the specified system and is replaced by that user's login directory;
(3) a path name preceded by ~/user where user is a login name on the specified system and is replaced by that user's directory under PUB-DIR;
(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see chmod(2)).

The following options are interpreted by uucp:

-d Make all necessary directories for the file copy (default).
-f Do not make intermediate directories for the file copy.
-c Use the source file when copying out rather than copying the file to the spool directory (default).
-C Copy the source file to the spool directory.
-m Send mail to the requester when the copy is complete.
-nuser Notify user on the remote system that a file was sent.
-esys Send the uucp command to system sys to be executed there.
   (Note — this will only be successful if the remote machine allows the uucp command to be executed by /usr/lib/uucp/uuxqt.)

Uulog maintains a summary log of uucp and uux(1C) transactions in the file /usr/spool/uucp/LOGFILE by gathering information from partial log files named /usr/spool/uucp/LOG.*. These files will only be created if the LOGFILE is being used by another process.) It removes the partial log files.

The options cause uulog to print logging information:

-sys Print information about work involving system sys.
—user Print information about work done for the specified user.
Ununame lists the uucp names of known systems. The —I option returns
the local system name.

FILES
/usr/spool/uucp spool directory
/usr/spool/uucppublic public directory for receiving and sending (PUB-DIR)
/usr/lib/uucp/* other data and program files

SEE ALSO
mail(1), uux(1C).
Uucp Implementation Description by D. A. Nowitz.

WARNING
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 "nuucp or just ").

BUGS
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.)
NAME
uustat — uucp status inquiry and job control

SYNOPSIS
uustat [ option ] ...

DESCRIPTION
Uustat will display the status of, or cancel, previously specified uucp commands, or provide general status on uucp connections to other systems. The following options are recognized:

- mmch Report the status of accessibility of machine mch. If mch is specified as all, then the status of all machines known to the local uucp are provided.
- kjobn Kill the uucp request whose job number is jobn. The killed uucp request must belong to the person issuing the uustat command unless he is the super-user.
- chour Remove the status entries which are older than hour hours. This administrative option can only be initiated by the user uucp or the super-user.
- uuser Report the status of all uucp requests issued by user.
- ssys Report the status of all uucp requests which communicate with remote system sys.
- ohour Report the status of all uucp requests which are older than hour hours.
- yhour Report the status of all uucp requests which are younger than hour hours.
- jall Report the status of all the uucp requests.
- v Report the uucp status verbosely. If this option is not specified, a status code is printed with each uucp request.

When no options are given, uustat outputs the status of all uucp requests issued by the current user. Note that only one of the options -j, -m, -k, -c, or the rest of other options may be specified.

For example, the command

uustat -uhdc -smhtsa -y72 -v

will print the verbose status of all uucp requests that were issued by user hdc to communicate with system mhtsa within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status

where the status may be either an octal number or a verbose description. The octal code corresponds to the following description:

<table>
<thead>
<tr>
<th>OCTAL STATUS</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000001</td>
<td>the copy failed, but the reason cannot be determined</td>
</tr>
<tr>
<td>000002</td>
<td>permission to access local file is denied</td>
</tr>
<tr>
<td>000004</td>
<td>permission to access remote file is denied</td>
</tr>
<tr>
<td>000100</td>
<td>bad uucp command is generated</td>
</tr>
<tr>
<td>000200</td>
<td>remote system cannot create temporary file</td>
</tr>
<tr>
<td>000400</td>
<td>cannot copy to remote directory</td>
</tr>
<tr>
<td>001000</td>
<td>cannot copy to local directory</td>
</tr>
<tr>
<td>002000</td>
<td>local system cannot create temporary file</td>
</tr>
<tr>
<td>004000</td>
<td>cannot execute uucp</td>
</tr>
<tr>
<td>010000</td>
<td>copy succeeded</td>
</tr>
<tr>
<td>020000</td>
<td>copy finished, job deleted</td>
</tr>
<tr>
<td>040000</td>
<td>job is queued</td>
</tr>
</tbody>
</table>
The meanings of the machine accessibility status are:

```
  system-name time status
```

where `time` is the latest status time and `status` is a self-explanatory description of the machine status.

### FILES

```
/usr/spool/uucp             spool directory
/usr/lib/uucp/L_stat        system status file
/usr/lib/uucp/R_stat        request status file
```

### SEE ALSO

- `uucp(1C)`

*Uustat – A UUCP Status Inquiry Program*, by H. Che.
NAME
uusub — monitor uucp network

SYNOPSIS
uusub [ options ]

DESCRIPTION
Uusub defines a uucp subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

-asy s Add sys to the subnetwork.
-dsy s Delete sys from the subnetwork.
- Report the statistics on connections.
-r Report the statistics on traffic amount.
-f Flush the connection statistics.
-uhr Gather the traffic statistics over the past hr hours.
-csys Exercise the connection to the system sys. If sys is specified as all, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

sys #call #ok time #dev #login #nack #other

where sys is the remote system name, #call is the number of times the local system tries to call sys since the last flush was done, #ok is the number of successful connections, time is the latest successful connect time, #dev is the number of unsuccessful connections because of no available device (e.g. ACU), #login is the number of unsuccessful connections because of login failure, #nack is the number of unsuccessful connections because of no response (e.g. line busy, system down), and #other is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

where sfile is the number of files sent and sbyte is the number of bytes sent over the period of time indicated in the latest uusub command with the -uhr option. Similarly, rfile and rbyte are the numbers of files and bytes received.

The command:

uusub -c all -u 24

is typically started by cron(1M) once a day.

FILES
/usr/spool/uucp/SYSLOG system log file
/usr/lib/uucp/L_sub connection statistics
/usr/lib/uucp/R_sub traffic statistics

SEE ALSO
uucp(1C), uustat(1C).
NAME
uuto, uupick — public UNIX-to-UNIX file copy

SYNOPSIS
uuto [ options ] source-files destination
uupick [ -s system ]

DESCRIPTION
Uuto sends source-files to destination. Uuto uses the uucp(1C) 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 uname(1C)). Logname 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. 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).
a [ dir ] Same as m except moving all the files sent from system.
p Print the content of the file.
q Stop.
EOT (control-d) Same as q.
!command Escape to the shell to do command.
* Print a command summary.

Uupick invoked with the -s system option will only search the PUBDIR for files sent from system.

FILES
PUBDIR/usr/spool/uucppublic public directory

SEE ALSO
mail(1), uuclean(1M), uucp(1C), uulog(1C), uname(1C), uustat(1C), uux(1C).
NAME
uux - unix to unix command execution

SYNOPSIS
uux [ - ] command-string

DESCRIPTION
Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from uux. Many sites will permit 'little more than the receipt of mail (see mail(1)) via uux.

The *command-string* is made up of one or more arguments that look like a Shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

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.

The - option will cause the standard input to the uux command to be the standard input to the *command-string*. For example, the command

```
uux "!diff usg!/usr/dan/fl pwba!/a4/dan/fl > !f1.diff"
```

will get the fl files from the "usg" and "pwba" machines, execute a *diff* command and put the results in *f1.diff* in the local 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 which are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!uucp b!/usr/file (c!/usr/file)
```

will send a *uucp* command to system "a" to get /usr/file from system "b" and send it to system "c".

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

FILES
/usr/lib/uucp/spool spool directory
/usr/lib/uucp/* other data and programs

SEE ALSO
uuclean(1M), uucp(1C).
Uucp Implementation Description by D. A. Nowitz

BUGS
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.
NAME
val — validate SCCS file

SYNOPSIS
val
val [-s] [-rSID] [-mname] [-ytype] files

DESCRIPTION
Val determines if the specified file is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to val may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

Val has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

-s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

-rSID The argument value SID (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the SID is ambiguous (e.g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e.g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the SID is valid and not ambiguous, a check is made to determine if it actually exists.

-mname The argument value name is compared with the SCCS %M% keyword in file.

--ytype The argument value type is compared with the SCCS %Y% keyword in file.

The 8-bit code returned by val is a disjunction of the possible errors, i.e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = can’t open file or file not SCCS;
bit 4 = SID is invalid or ambiguous;
bit 5 = SID does not exist;
bit 6 = %Y%, -y mismatch;
bit 7 = %M%, -m mismatch;

Note that val can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned — a logical OR of the codes generated for each command line and file processed.
SEE ALSO
   admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS
   Use help(1) for explanations.

BUGS
   Val can process up to 50 files on a single command line. Any number above 50 will produce a core dump.
NAME
vc – version control

SYNOPSIS
vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

DESCRIPTION
The vc command copies lines from the standard input to the standard output under control of its arguments and control statements encountered in the standard input. In the process of performing the copy operation, user declared keywords may be replaced by their string value when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, except as modified by the -t keyletter (see below). The default control character is colon (:), except as modified by the -c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with ed(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The -a keyletter (see below) forces replacement of keywords in all text lines and not just in vc statements. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter arguments
- a Forces replacement of keywords surrounded by control characters with their assigned value in all text lines and not just in vc statements.
- t All characters from the beginning of a line up to and including the first tab character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the tab are discarded.
- cchar Specifies a control character to be used in place of :.
- s Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements
:dcl keyword[, ..., keyword]
Used to declare keywords. All keywords must be declared.

:asg keyword=value
Used to assign values to keywords. An asg statement overrides the assignment for the corresponding keyword on the vc command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

:if condition


: end
Used to skip lines of the standard input. If the condition is true all lines between the if statement and the matching end statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening if statements and matching end statements are recognized solely for the purpose of maintaining the proper if-end matching.

The syntax of a condition is:

<cond> ::= [ "not" ] <or>
<or> ::= <and> | <and> "!" <or>
<and> ::= <exp> | <exp> "&" <and>
<exp> ::= "(" <or> ")" | <value> <op> <value>
<op> ::= "=" | "!=" | "<" | ">" |
<value> ::= <arbitrary ASCII string> | <numeric string>

The available operators and their meanings are:

=         equal
!=        not equal
&         and
|         or
>         greater than
<         less than
( )       used for logical groupings
not        may only occur immediately after the if, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g.: 012 > 12 is false). All other operators take strings as arguments (e.g.: 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

== != > < all of equal precedence
&
|

Parentheses may be used to alter the order of precedence. Values must be separated from operators or parentheses by at least one blank or tab.

::text
Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the -a keyletter.

:on
:off
Turn on or off keyword replacement on all lines.

:ctl char
Change the control character to char.

:msg message
Prints the given message on the diagnostic output.
:err message
  Prints the given message followed by:
    ERROR: err statement on line ... (915)
  on the diagnostic output. Vc halts execution, and returns an exit code
  of 1.

DIAGNOSTICS
  Use help(1) for explanations.

EXIT CODES
  0 — normal
  1 — any error
NAME

vlx - VAX-11/780 LSI console floppy interface

SYNOPSIS

vlx key [ files ]

DESCRIPTION

VLX is used to maintain the console floppy. The floppy is in DEC RT-II format. Hence, a file name is restricted to a 1- to 6-character alphanumeric name optionally followed by a . character separator and a 1- to 3-character alphanumeric extension. Upper and lower cases are mapped together. Only the last component of a path name is used.

Key is one character from the set drtx, optionally concatenated with one or both of vf. The meanings of the key characters are:

d Delete the named files from the floppy.

r Replace the named files on the floppy.

t Print a table of contents of the floppy. If no names are given, all files are tabled. If names are given, only those files are tabled.

x Extract the named files from the floppy. If no names are given, all files are extracted.

v Verbose. When used with t, it gives a long listing of all information about the files. When used with x, it precedes each file with a name.

f Use the next name as the floppy file name, instead of the default /dev/conflop.

FILES

/dev/conflop console floppy

SEE ALSO

vaxops(8).

BUGS

Dependent on knowledge and correctness of DEC software.
NAME
volcopy, labelit — copy file systems with label checking

SYNOPSIS
/etc/volcopy [-bpi bits-per-inch] [-feetsize] fsname special1 volname1
special2 volname2

/etc/labelit special [ fsname volume [ -n ] ]

DESCRIPTION
Volcopy makes a literal copy of the file system using a blocksize matched to
the device (10 blocks for 800/1600 bpi tape; 88 blocks for everything else).
Using volcopy, a 2400 foot/1600 bpi tape will hold a 65K file system. The
optional flag arguments are used only with tapes (-bpi -- bits-per-inch;
-feet -- size of reel in feet). The program requests the information if it is
not given on the command line. If the file system is too large to fit on one
reel, volcopy will prompt for additional reels. Labels of all reels are
checked. Tapes may be mounted alternately on two drives.

The fsname argument represents the mounted name (e.g.: root, ul, etc.) of
the filesystem being copied.

The special should be the physical disk section or tape (e.g.: /dev/rrp15,
/dev/rmt0, etc.).

The volname is the physical volume name (e.g.: pk3, t0122, etc.) and
should match the external label sticker. Such label names are limited to
five or fewer characters.

Special1 and volname1 are the device and volume from which the copy of
the file system is being extracted. Special2 and volname2 are the target dev-
vice and volume.

Fsname and volname are recorded in the last 12 characters of the superblock
(char fsname[6], volname[6];).

Labelit can be used to provide initial labels for unmounted disk or tape file
systems. With the optional arguments omitted, labelit prints current label
values. The -n option provides for initial labeling of new tapes only (this
destroys previous contents).

FILES
/etc/log/filesave a record of file systems/volumes copied

SEE ALSO
fs(5).

BUGS
Only device names beginning /dev/rmt are treated as tapes.
NAME

vpmc — compiler for the virtual protocol machine

SYNOPSIS

vpmc [-m] [-r] [-c] [-x] [-s sfile] [-l lfile]
[-i ifile] [-o ofile] file

DESCRIPTION

vpmc is the compiler for a language that is used to describe communications link protocols. The output of vpmc is a load module for the virtual protocol machine (VPM), which is a software construct for implementing communications link protocols (e.g., BISYNC) on the DEC KMC11 microprocessor. VPM is implemented by an interpreter in the KMC11 which cooperates with a driver in the UNIX host computer to transfer data over a communications link in accordance with a specified link protocol. UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal UNIX open, read, write, and close operations. The VPM program in the KMC11 provides error control and flow control using the conventions specified in the protocol.

The language accepted by vpmc is essentially a subset of C; the implementation of vpmc uses the RATFOR preprocessor (ratfor(1)) as a front end; this leads to a few minor differences, mostly syntactic.

There are two versions of the interpreter. The appropriate version for a particular application is selected by means of the -i option. The BISYNC version (-i bisync) supports half-duplex, character-oriented protocols such as the various forms of BISYNC. The HDLC version (-i bdlc) supports full-duplex, bit-oriented protocols such as HDLC. The communications primitives used with the BISYNC version are character-oriented and blocking; the primitives used with the HDLC version are frame-oriented and non-blocking.

Options

The meanings of the command-line options are:

- m Use m4(1) instead of cpp as the macro preprocessor.
- r Produce RATFOR output on the standard output and suppress the remaining compiler phases.
- c Compile only (suppress the assembly and linking phases).
- x Retain the intermediate files used for communication between passes.
- s sfile Save the generated VPM assembly language on file sfile.
- l lfile Produce a VPM assembly-language listing on file lfile.
- i ifile Use the interpreter version specified by ifile (default bisync).
- o ofile Write the executable object file on file ofile (default a.out).

These options may be given in any order.

Programs

Input to vpmc consists of a (possibly null) sequence of array declarations, followed by one or more function definitions. The first defined function is invoked (on command from the UNIX VPM driver) to begin program execution.

Functions

A function definition has the following form:

    function name() 
    statement_list 
end
Function arguments (formal parameters) are not allowed. The effect of a function call with arguments can be obtained by invoking the function via a macro that first assigns the value of each argument to a global variable reserved for that purpose. See EXAMPLES below.

A statement_list is a (possibly null) sequence of labeled statements. A labeled_statement is a statement preceded by a (possibly null) sequence of labels. A label is either a name followed by a colon (:) or a decimal integer optionally followed by a colon.

The statements that make up a statement list must be separated by semicolons (;). (A semicolon at the end of a line can usually be omitted; refer to the description of RATFOR for details.) Null statements are allowed.

Statement Syntax
The following types of statements are allowed:

- expression
  - lvalue = expression
  - lvalue += expression
  - lvalue -= expression
  - lvalue |= expression
  - lvalue &= expression
  - lvalue ^= expression
  - lvalue <<= expression
  - lvalue >>= expression
  - if(expression) statement
  - if(expression) statement else statement
  - while(expression) statement
  - for(statement; expression; statement) statement
  - repeat statement
  - repeat statement until expression
  - break
  - next
  - switch(expression) {case_list}
  - return(expression)
  - return
  - goto name
  - goto decimal_constant
  - {statement_list}

repeat is equivalent to the do keyword in C; next is equivalent to continue.

A case_list is a sequence of statement lists, each of which is preceded by a label of the form:

- case constant:

The label for the last statement_list in a case_list may be of the form:

- default:

Unlike C, RATFOR supplies an automatic break preceding each new case label.

Expression Syntax
A primary_expression (abbreviated primary) is an lvalue or a constant. An lvalue is one of the following:

- name
  - name [constant]

A unary_expression (abbreviated unary) is one of the following:
The following types of expressions are allowed:

- primary
- name()
- system_call
- ++lvalue
- --lvalue
- (expression)
- !unary
- ~unary

Note that the right operand of a binary operator can only be a constant, a name, or a name with a constant subscript.

System Calls

A VPM program interacts with a communications device and a driver in the host computer by means of system calls (primitives).

The following primitives are available only in the BISYNC version of the interpreter:

- **crc16(primary)**
  
The value of the primary expression is combined with the cyclic redundancy check-sum at the location passed by a previous **crcloc** system call. The CRC-16 polynomial \((x^{16}+x^{15}+x^2+1)\) is used for the check-sum calculation.

- **crcloc(name)**
  
The two-byte array starting at the location specified by *name* is cleared. The address of the array is recorded as the location to be updated by subsequent **crc16** system calls.

- **get(lvalue)**
  
  Get a byte from the current transmit buffer. The next available byte, if any, is copied into the location specified by *lvalue*. The returned value is zero if a byte was obtained, otherwise it is non-zero.

- **getbuf(name)**
  
  Get (open) a receive buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by *name*. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a receive buffer has previously been opened via a **getbuf** call but has not yet been closed via a call to **rtnrbuf**, that buffer is reinitialized and
remains the current buffer.

getxbuf(name)
Get (open) a transmit buffer. The returned value is zero if a buffer is available, otherwise it is non-zero. If a buffer is obtained, the buffer parameters are copied into the array specified by name. The array should be large enough to hold at least three bytes. The meaning of the buffer parameters is driver-dependent. If a transmit buffer has previously been opened via a getxbuf call but has not yet been closed via a call to rtnxbuf, that buffer is reinitialized and remains the current buffer.

put(primary)
Put a byte into the current receive buffer. The value of the primary expression is inserted into the next available position, if any, in the current receive buffer. The returned value is zero if a byte was transferred, otherwise it is non-zero.

recv(lvalue)
Receive a character. The process delays until a character is available in the input silo. The character is then moved to the location specified by lvalue and the process is reactivated.

rsom(constant)
Skip to the beginning of a new receive frame. The receiver hardware is cleared and the value of constant is stored as the receive sync character. This call is used to synchronize the local receiver and remote transmitter when the process is ready to accept a new receive frame.

rtnrbuf(name)
Return a receive buffer. The original values of the buffer parameters for the current receive buffer are replaced with values from the array specified by name. The current receive buffer is then released to the driver.

rtnxbuf(name)
Return a transmit buffer. The original values of the buffer parameters for the current transmit buffer are replaced with values from the array specified by name. The current transmit buffer is then released to the driver.

xeom(constant)
Transmit end-of-message. The value of the constant is transmitted, then the transmitter is shut down.

xmt(primary)
Transmit a character. The value of the primary expression is transmitted over the communications line. If the output silo is full, the process waits until there is room in the silo.

xsom(constant)
Transmit start-of-message. The transmitter is cleared, then the value of constant is transmitted six times. This call is used to synchronize the local transmitter and the remote receiver at the beginning of a frame.

The following primitives are available only with the HDLC version of the interpreter:

abtxfrm()
The current transmission, if any, is aborted, if possible, by sending a frame-abort sequence (seven one bits, followed immediately by a
terminating flag). This operation is not feasible with some hardware interfaces, in which case this primitive is a no-operation.

getxfrnm(*primary*)
Get a transmit buffer. If the transmit-buffer queue is not empty, the buffer at the head of the queue is removed from the queue and attached to the sequence number specified by the value of the *primary* expression. If the sequence number is greater than seven or the sequence number already has a buffer attached, the process is terminated in error. The returned value is zero if a buffer was obtained, otherwise non-zero.

rcvfrnm(*name*)
Get a completed receive frame. If the queue of completed receive frames is non-empty, the frame at the head of the queue is removed and becomes the current receive frame. If a frame is obtained, the first five bytes of the frame are copied into the array specified by *name*. The returned value is true (non-zero) if a frame was obtained; otherwise, it is false (zero). The rightmost four bits of the returned value indicate the frame length as follows: if the value of the rightmost four bits is equal to fifteen, the frame length is greater than or equal to 15; otherwise the frame length is equal to the value of the rightmost four bits. The frame length includes the two CRC bytes at the end of the frame and any control information at the beginning of the frame. Bytes following the first two bytes of the frame, but not including the two CRC bytes, are copied into a receive buffer, if one is available at the time the frame is received. Bit 020 of the returned value is zero if a receive buffer was available, otherwise non-zero. The values of the leftmost three bits of the returned value are currently unspecified. If a frame was obtained, the first five bytes of the frame are copied into the array specified by *name*. Frames with errors are discarded; a count is kept for each type of error. Frames may be discarded for any of the following reasons: (1) CRC error, (2) frame too short (less than four bytes), (3) frame too long (buffer size exceeded), or (4) no receive buffer available. If a frame with a buffer attached was previously obtained with rcvfrnm, but the buffer has not been released to the driver with rtnrfrnm, that buffer is returned to the queue of empty receive buffers. At most one receive frame with no buffer attached is retained by the interpreter; if a new frame arrives before the frame with no buffer attached has been obtained with rcvfrnm, the new frame is discarded.

rtnrfrnm()
Return a receive buffer. The current receive buffer (the one obtained by the most recent rcvfrnm primitive) is returned to the driver. If there is no current receive buffer, the process is terminated in error.

rsxmtq()
Reset the transmit-buffer queue. The sequence number assignment is removed from all transmit buffers. If a transmission is currently in progress, the transmission is aborted, if possible.

rtnxfrnm(*primary*)
Return a transmit buffer. The transmit buffer currently attached to the sequence number specified by the value of the *primary* is returned to the driver and the sequence number assignment is removed from that buffer. If the specified sequence number does not have a
buffer attached, the process is terminated in error. Transmit buffers must be returned in the same sequence in which they were obtained, otherwise the process is terminated in error.

**setctl(name,primary)**

Specify transmit-control information. The number of bytes specified by the `primary` are copied from the array specified by `name` and saved for use with subsequent `xmtfrm` or `xmtdl` primitives. If the transmitter is currently busy, the process is terminated in error.

**xmtbusy()**

Test for transmitter busy. If a frame is currently being transmitted, the returned value is *true* (non-zero); otherwise the returned value is *false* (zero).

**xmtctrl()**

Transmit a control frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent `setctl` primitive, followed by a two-byte CRC. The CRC-CCITT polynomial \((x^{16}+x^{12}+x^5+1)\) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero.

**xmtfrm(primary)**

Transmit an information frame. If a transmission is not already in progress, a new transmission is initiated. The transmitted frame will contain the control information specified by the most recent `setctl` primitive, followed by the contents of the buffer which is currently attached to the sequence number specified by the value of the `primary` expression, followed by a two-byte CRC. The CRC-CCITT polynomial \((x^{16}+x^{12}+x^5+1)\) is used for the CRC calculation. The returned value is zero if a new transmission was initiated, otherwise non-zero. If the sequence number is greater than seven or the sequence number does not have a buffer attached, the process is terminated in error.

The following primitives are available with all versions of the interpreter:

**dsrwait()**

Wait for modem-ready and then set modem-ready mode. The process delays until the modem-ready signal from the modem interface is asserted. If the modem-ready signal subsequently drops, the process is terminated. If `dsrwait` is never invoked, the modem-ready signal is ignored.

**exit(primary)**

Terminate execution. The process is halted and the value of the `primary` expression is passed to the driver.

**getemd(name)**

Get a command from the driver. If a command has been received from the driver since the last call to `getemd`, four bytes of command information are copied into the array specified by `name` and a value of *true* (non-zero) is returned. If no command is available, the returned value is *false* (zero).

**pause()**

Return control to the dispatcher. This primitive informs the dispatcher that the virtual process may be suspended until the next occurrence of an event that might affect the state of the protocol for this line. Examples of such events are: (1) completion of an
output transfer, (2) completion of an input transfer, (3) timer expiration, and (4) a buffer-in command from the driver. In a multi-line implementation, the pause primitive allows the process for a given line to give up control to allow the processor to service another line.

\texttt{rtntpt(name)}

Return a report to the driver. Four bytes from the array specified by \texttt{name} are transferred to the driver. The process delays until the transfer is complete.

\texttt{testop(primary)}

Test for odd parity. The returned value is \texttt{true} (non-zero) if the value of the primary expression has odd parity, otherwise the returned value is \texttt{false} (zero).

\texttt{timeout(primary)}

Schedule or cancel a timer interrupt. If the value of the \texttt{primary} expression is non-zero, the current values of the program counter and stack pointer are saved and a timer is loaded with the value of \texttt{primary}. The system call then returns immediately with a value of \texttt{false} (zero) as the returned value. The timer is decremented each tenth of a second thereafter. If the timer is decremented to zero, the saved values of the program counter and stack pointer are restored and the system call returns with a value of \texttt{true} (non-zero). The effect of the timer interrupt is to return control to the code immediately following the \texttt{timeout} system call, at which point a non-zero return value indicates that the timer has expired. The \texttt{timeout} system call with a non-zero argument is normally written as the condition part of an if statement. A \texttt{timeout} system call with a zero argument value cancels all previous \texttt{timeout} requests, as does a \texttt{return} from the function in which the \texttt{timeout} system call was made. A \texttt{timeout} system call with a non-zero argument value overrides all previous \texttt{timeout} requests. The maximum permissible value for the argument is 255, which gives a timeout period of 25.5 seconds.

\texttt{timer(primary)}

Start a timer or test for timer expiration. If the value of the \texttt{primary} is non-zero, a software timer is loaded with the value of the \texttt{primary} and a value of \texttt{true} (non-zero) is returned. The timer is decremented each tenth of a second thereafter until it reaches zero. If the value of the \texttt{primary} is zero, the returned value is the current value of the timer; this will be \texttt{true} (non-zero) if the value of the timer is currently non-zero, otherwise \texttt{false} (zero). The timer used by this primitive is different from the timer used by the \texttt{timeout} primitive.

\texttt{trace(primary[,primary])}

The values of the two primary expressions and the current value of the script location counter are passed to the driver. If the second \texttt{primary} is omitted, a zero is used instead. The process delays until the values have been accepted by the host computer.

Constants

A constant is a decimal, octal, or hexadecimal integer, or a single character enclosed in single quotes. A token consisting of a string of digits is taken to be an octal integer if the first digit is a zero, otherwise the string is interpreted as a decimal integer. If a token begins with \texttt{0x} or \texttt{0X}, the remainder of the token is interpreted as a hexadecimal integer. The hexadecimal
digits include a through f or, equivalently, A through F.

Variables
Variable names may be used without having been previously declared. All names are global. All values are treated as 8-bit unsigned integers.

Arrays of contiguous storage may be allocated using the array declaration:

array name[constant]

where constant is a decimal integer. Elements of arrays can be referenced using constant subscripts:

name[constant]

Indexing of arrays assumes that the first element has an index of zero.

Names
A name is a sequence of letters and digits; the first character must be a letter. Upper- and lower-case letters are considered to be distinct. Names longer than 31 characters are truncated to 31 characters. The underscore (_ ) may be used within a name to improve readability, but is discarded by RATFOR.

Preprocessor Commands
If the -m option is omitted, comments, macro definitions, and file inclusion statements are written as in C. Otherwise, the following rules apply:

1. If the character # appears in an input line, the remainder of the line is treated as a comment.

2. A statement of the form:
   define(name,text)

causes every subsequent appearance of name to be replaced by text. The defining text includes everything after the comma up to the balancing right parenthesis; multi-line definitions are allowed. Macros may have arguments. Any occurrence of $n$ within the replacement text for a macro will be replaced by the $n$th actual argument when the macro is invoked.

3. A statement of the form:
   include(file)

inserts the contents of file in place of the include command. The contents of the included file is often a set of definitions.

EXAMPLES
These examples require the use of the -m option.

# The function defined below transmits a frame in transparent BISYNC.
# A transmit buffer must be obtained with getxbuf before the function
# is invoked.
#
# Define symbolic constants:
#
define(DLE,0x10)
define(ETB,0x26)
define(PAD,0xff)
define(STX,0x02)
define(SYNC,0x32)
#
# Define a macro with an argument:
define(xmtcrc, {crc16($1); xmt($1);})

Declare an array:
array crc[2];

Define the function:
function xmtblk(
    crcloc(crc);
    xsom(SYNC);
    xmt(DLE);
    xmt(STX);
    while(get(byte) == 0){
        if(byte == DLE)
            xmt(DLE);
        xmtcrc(byte);
    }
    xmt(DLE);
    xmtcrc(ETB);
    xmt(crc[0]);
    xmt(crc[1]);
    xeom(PAD);
}

The following example illustrates the use of macros to simulate a
function call with arguments.

The macro definition:
define(xmtctl, {c=$1; d=$2; xmtctl1()} )

The function definition:
function xmtctl1(
    xsom(SYNC);
    xmt(c);
    if(d != 0)
        xmt(d);
    xeom(PAD);
)

Sample invocation:
function test(
    xmtctl(DLE, 0x70);
)

FILES
sas_temp* temporary
/tmp/sas_ta?? temporary
/tmp/sas_tb?? temporary
/usr/lib/vpm/pass* compiler phases
/usr/lib/vpm/pl compiler phase
/usr/lib/vpm/vratfor compiler phase
SEE ALSO
m4(1), ratfor(1), vpmstart(1C), vpm(4).
RATFOR— A Preprocessor for a Rational Fortran by B. W. Kernighan.
The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.
Software Tools by B. W. Kernighan and P. J. Plauger (pp. 28-30).
NAME
  vpmstart, vpmsnap, vpmtrace — load the KMC11-B; print VPM traces

SYNOPSIS
  vpmstart device n [ filen ]
  vpmsnap
  vpmtrace

DESCRIPTION
  Vpmstart writes filen (a.out by default) to the KMC11-B specified by device.
  The argument n is a magic number that the KMC11-B driver saves to identify
  the running program. This number is checked when the VPM driver is
  opened to provide some assurance that the program running in the
  KMC11-B is the one expected. The magic number for VPM interpreters is
  6. When filen has been written to the KMC11-B, its execution is begun.
  Filen may be any file executable by the KMC11-B.

  If filen is made using vpmc(1C), the VPM interpreter will be started by
  vpmstart. The VPM interpreter waits for a RUN command from the VPM
  driver before beginning execution of the protocol script. The RUN com-
  mand is sent by the VPM driver when the corresponding VPM device file is
  opened.

  Vpmsnap opens the trace driver (minor device number 1) and reads and
  prints time-stamped event records until killed.

  Vpmtrace opens the trace driver (minor device number 0) and reads and
  prints event records until killed.

SEE ALSO
  vpmc(1C), trace(4), vpm(4).
NAME
vpr — Versatec printer spooler

SYNOPSIS
vpr [ options ] [ files ]

DESCRIPTION
Vpr causes the named files to be queued for printing on a Versatec printer. If no names appear, the standard input is assumed; thus vpr may be used as a filter.

The following options may be given (each as a separate argument and in any order) before any file name arguments:

- c Makes a copy of the file to be sent before returning to the user.
- r Removes the file after sending it.
- m When printing is complete, reports that fact by mail(1).
- n Does not report the completion of printing by mail(1). This is the default option.
- f Uses the next argument as a dummy file name when reporting completion by mail(1), thus forcing the -m option. (This is useful for distinguishing multiple runs, especially when vpr is being used as a filter).
- p [ -e raster ]
  Uses the plot filter vplot to output files produced by graph(1G). The -e option will cause a previously scan converted file raster to be sent to the Versatec.
- t Uses the troff filter vcat to output files produced by troff(1). Troff must be invoked with the -t option.
- nF For n between 1 and 4, assumes font F is mounted in font position n, where F is R, I, B, or S.

EXAMPLES
Two common uses are:

troff -t [ options ] file | vpr -t

and

graph [ options ] file | vpr -p

FILES
/etc/passwd user's identification and accounting data
/usr/spool/vpd/* spool area
/usr/lib/vpd line printer daemon
/usr/lib/vpd.pr print filter
/usr/lib/vcat troff filter
/usr/lib/vplot plot filter

SEE ALSO
dpr(1C), lpr(1), tplot(1G).
NAME
wait — await completion of process

SYNOPSIS
wait

DESCRIPTION
Wait until all processes started with & have completed, and report on
abnormal terminations.

Because the wait(2) system call must be executed in the parent process, the
shell itself executes wait, without creating a new process.

SEE ALSO
sh(1).

BUGS
Not all the processes of a 3- or more-stage pipeline are children of the
shell, and thus can’t be waited for.
NAME
wall — write to all users

SYNOPSIS
/etc/wall

DESCRIPTION
Wall reads its standard input until an end-of-file. It then sends this message to all currently logged in users preceded by "Broadcast Message from ...". It is used to warn all users, typically prior to shutting down the system.

The sender should be super-user to override any protections the users may have invoked.

FILES
/dev/tty*

SEE ALSO
mesg(1), write(1).

DIAGNOSTICS
"Cannot send to ..." when the open on a user's tty file fails.
NAME
wc — word count

SYNOPSIS
wc [ -lwce ] [ 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 -lwce.

When names are specified on the command line, they will be printed along with the counts.
NAME
what — identify SCCS files

SYNOPSIS
what files

DESCRIPTION
What searches the given files for all occurrences of the pattern that get(1)
substitutes for %Z% (this is @( ) at this printing) and prints out what fol-
lows until the first *, >, new-line, \, or null character. For example, if the
C program in file f.c contains

char ident[] = "(@( )identification information ");

and f.c is compiled to yield f.o and a.out, then the command

what f.c f.o a.out

will print

f.c:
   identification information
f.o:
   identification information
a.out:
   identification information

What is intended to be used in conjunction with the command get(1),
which automatically inserts identifying information, but it can also be used
where the information is inserted manually.

SEE ALSO
get(1), help(1).

DIAGNOSTICS
Use help(1) for explanations.

BUGS
It’s possible that an unintended occurrence of the pattern @( ) could be
found just by chance, but this causes no harm in nearly all cases.
NAME
who — who is on the system

SYNOPSIS
who [ who-file ] [ am I ]

DESCRIPTION
Who, without an argument, lists the login name, terminal name, and login
time for each current UNIX user.

Without an argument, who examines the /etc/utmp file to obtain its infor-
mation. If a file is given, that file is examined. Typically the given file will
be /usr/adm/wtmp, which contains a record of all the logins since it was
created. Then who lists logins, logouts, and crashes since the creation of
the wtmp file. Each login is listed with user name, terminal name (with
/dev/ suppressed), and date and time. When an argument is given, logouts
produce a similar line without a user name. Reboots produce a line with x
in the place of the device name, and a fossil time indicative of when the
system went down.

With two arguments, as in who am I (and also who are you), who tells who
you are logged in as.

FILES
/etc/utmp

SEE ALSO
getuid(2), utmp(5).
WHODO(1M)

NAME
whodo — who is doing what

SYNOPSIS
/etc/whodo

DESCRIPTION
Whodo produces merged, reformatted, and dated output from the who(1) and ps(1) commands.

SEE ALSO
ps(1), who(1).
NAME
write — write to another user

SYNOPSIS
write user [ tty ]

DESCRIPTION
Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from your-logname your-tty ...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point, write writes EOF on the other terminal and exits.

If you want to write to a user who is logged in more than once, the tty argument may be used to indicate the appropriate terminal.

Permission to write may be denied or granted by use of the mesg(1) command. At the outset, writing is allowed. Certain commands, in particular nroff(1) and pr(1), disallow messages in order to prevent messy output.

If the character ! is found at the beginning of a line, write calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using write: when you first write to another user, wait for him or her to write back before starting to send. Each party should end each message with a distinctive signal ((o) for “over” is conventional), indicating that the other may reply; (oo) for “over and out” is suggested when conversation is to be terminated.

FILES
/etc/utmp to find user
/bin/sh to execute !

SEE ALSO
mail(1), mesg(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 number of arguments read for each command invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for, using one's $PATH. If command is omitted, /bin/echo is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted: Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the initial-arguments, followed by some number of arguments read from standard input (Exception: see -i flag). Flags -i, -I, and -n determine how arguments are selected for each command invocation. When none of these flags are coded, the initial-arguments are followed by arguments read continuously from standard input until an internal buffer is full, and then command is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., -I vs. -n), the last flag has precedence. Flag values are:

- Inumber
  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.

- ireplstr
  Insert mode: command is executed for each line from standard input, taking the entire line as a single arg, inserting it in initial-arguments for each occurrence of replstr. A maximum of 5 arguments in initial-arguments may each contain one or more instances of replstr. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option -x is also forced. {} is assumed for replstr if not specified.

- nnumber
  Execute command using as many standard input arguments as possible, up to number arguments maximum. Fewer arguments will be used if their total size is greater than size characters, and for the last invocation if there are fewer than number arguments remaining. If option -x is also coded, each number arguments must fit in the size limitation, else xargs terminates execution.
Trace mode: The command and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

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.

Causes xargs to terminate if any argument list would be greater than size characters; -x is forced by the options -i, -l, or -n are coded, the total length of all arguments must be within the size limit.

The maximum total size of each argument list is set to size characters; size must be a positive integer less than or equal to 470. If -s is not coded, 470 is taken as the default. Note that the character count for size includes one extra character for each argument and the count of characters in the command name.

EOF/str is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if -e is not coded. -e with no eofstr coded turns off the logical EOF string capability (underbar is taken literally). Xargs reads standard input until either end-of-file or the logical EOF string is encountered.

Xargs will terminate if either it receives a return code of -1 from, or if it cannot execute, command. When command is a shell program, it should explicitly exit (see sh(1)) with an appropriate value to avoid accidentally returning with -1.

EXAMPLES
The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

ls $1 | xargs -i -t mv $1/{} $2/{}

The following will combine the output of the-parenthesized commands onto one line, which is then echoed to the end of file log:

(logname; date; echo $0 $*) | xargs >>log

The user is asked which files in the current directory are to be archived and archives them into arch (1.) one at a time, or (2.) many at a time.

1. ls | xargs -p -l ar r arch
2. ls | xargs -p -l | xargs ar r arch

The following will execute diff(1) with successive pairs of arguments originally typed as shell arguments:

echo $* | xargs -n2 diff

DIAGNOSTICS
Self explanatory.
NAME
xref — cross reference for C programs

SYNOPSIS
xref [ file ... ]

DESCRIPTION
Xref reads the named files or the standard input if no file is specified and prints a cross reference consisting of lines of the form

    identifier    file-name    line-numbers ...

Function definition is indicated by a plus sign (+) preceding the line number.

SEE ALSO
cref(1).
NAME
yacc — yet another compiler-compiler

SYNOPSIS
yacc [-vd] grammar

DESCRIPTION
Yacc converts a context-free grammar into a set of tables for a simple
automaton which executes an LR(1) parsing algorithm. The grammar may
be ambiguous; specified precedence rules are used to break ambiguities.

The output file, y.tab.c, must be compiled by the C compiler to produce a
program yyparse. This program must be loaded with the lexical analyzer
program, yylex, as well as main and yyerror, an error handling routine.
These routines must be supplied by the user; lex(1) is useful for creating
lexical analyzers usable by yacc.

If the -v flag is given, the file y.output is prepared, which contains a
description of the parsing tables and a report on conflicts generated by
ambiguities in the grammar.

If the -d flag is used, the file y.tab.h is generated with the #define sta-
tements that associate the yacc-assigned "token codes" with the user-declared "token names". This allows source files other than y.tab.c to
access the token codes.

FILES
y.output
y.tab.c
y.tab.h
defines for token names
yacc.tmp, yacc.acts
temporary files
/usr/lib/yaccpar
parser prototype for C programs

SEE ALSO
lex(1).
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June,
1974.
YACC — Yet Another Compiler Compiler by S. C. Johnson.

DIAGNOSTICS
The number of reduce-reduce and shift-reduce conflicts is reported on the
standard output; a more detailed report is found in the y.output file. Simi-
larly, if some rules are not reachable from the start symbol, this is also
reported.

BUGS
Because file names are fixed, at most one yacc process can be active in a
given directory at a time.
INTRO(2)

NAME
intro — introduction to system calls and error numbers

SYNOPSIS
#include <errno.h>

DESCRIPTION
This section describes all of the system calls. Most of these calls have one
or more error returns. An error condition is indicated by an otherwise
impossible returned value. This is almost always −1; the individual
descriptions specify the details. An error number is also made available in
the external variable errno. Errno is not cleared on successful calls, so it
should be tested only after an error has been indicated.

All of the possible error numbers are not listed in each system call descrip-
tion because many errors are possible for most of the calls. The following
is a complete list of the error numbers and their names as defined in
<error.h>.

1 EPERM Not owner
   Typically this error indicates an attempt to modify a file in some
   way forbidden except to its owner or super-user. It is also returned
   for attempts by ordinary users to do things allowed only to the
   super-user.

2 ENOENT No such file or directory
   This error occurs when a file name is specified and the file should
   exist but doesn't, or when one of the directories in a path name
does not exist.

3 ESRCH No such process
   No process can be found corresponding to that specified by pid in
   kill or ptrace.

4 EINTR Interrupted system call
   An asynchronous signal (such as interrupt or quit), which the user
   has elected to catch, occurred during a system call. If execution is
   resumed after processing the signal, it will appear as if the interrup-
ted system call returned this error condition.

5 EIO I/O error
   Some physical I/O error. This error may in some cases occur on a
   call following the one to which it actually applies.

6 ENXIO No such device or address
   I/O on a special file refers to a subdevice which does not exist, or
   beyond the limits of the device. It may also occur when, for exam-
   ple, a tape drive is not on-line or no disk pack is loaded on a drive.

7 E2BIG Arg list too long
   An argument list longer than 5,120 bytes is presented to a member
   of the exec family.

8 ENOEXEC Exec format error
   A request is made to execute a file which, although it has the
   appropriate permissions, does not start with a valid magic number
   (see a.out(5)).

9 EBADF Bad file number
   Either a file descriptor refers to no open file, or a read (respectively
   write) request is made to a file which is open only for writing
   (respectively reading).
10 ECHILD No child processes
   A *wait*, was executed by a process that had no existing or
   unwaited-for child processes.

11 EAGAIN No more processes
   A *fork*, failed because the system's process table is full or the user
   is not allowed to create any more processes.

12 ENOMEM Not enough space
   During an *exec*, *brk*, or *sbrk*, a program asks for more space than
   the system is able to supply. This is not a temporary condition; the
   maximum space size is a system parameter. The error may also
   occur if the arrangement of text, data, and stack segments requires
   too many segmentation registers, or if there is not enough swap
   space during a *fork*.

13 EACCES Permission denied
   An attempt was made to access a file in a way forbidden by the pro-
   tection system.

14 EFAULT Bad address
   The system encountered a hardware fault in attempting to use an
   argument of a system call.

15 ENOTBLK Block device required
   A non-block file was mentioned where a block device was required,
   e.g., in *mount*.

16 EBUSY Mount device busy
   An attempt to mount a device that was already mounted or an
   attempt was made to dismount a device on which there is an active
   file (open file, current directory, mounted-on file, active text seg-
   ment). It will also occur if an attempt is made to enable accounting
   when it is already enabled.

17 EEXIST File exists
   An existing file was mentioned in an inappropriate context, e.g.,
   *link*.

18 EXDEV Cross-device link
   A link to a file on another device was attempted.

19 ENODEV No such device
   An attempt was made to apply an inappropriate system call to a
   device; e.g., read a write-only device.

20 ENOTDIR Not a directory
   A non-directory was specified where a directory is required, for
   example in a path prefix or as an argument to *chdir*(2).

21 EISDIR Is a directory
   An attempt to write on a directory.

22 EINVAL Invalid argument
   Some invalid argument (e.g., dismounting a non-mounted device;
   mentioning an undefined signal in *signal*, or *kill*; reading or writing
   a file for which *lseek* has generated a negative pointer). Also set by
   the math functions described in the (3M) entries of this manual.

23 ENFILE File table overflow
   The system's table of open files is full, and temporarily no more
   *opens* can be accepted.

24 EMFILE Too many open files
   No process may have more than 20 file descriptors open at a time.
25 ENOTTY  Not a typewriter

26 ETXTBSY  Text file busy
   An attempt to execute a pure-procedure program which is currently
   open for writing (or reading). Also an attempt to open for writing
   a pure-procedure program that is being executed.

27 EFBIG  File too large
   The size of a file exceeded the maximum file size (1,082,201,088
   bytes) or ULIMIT; see ulimit(2).

28 ENOSPC  No space left on device
   During a write to an ordinary file, there is no free space left on the
   device.

29 ESPIME  Illegal seek
   An lseek was issued to a pipe.

30 EROFS  Read-only file system
   An attempt to modify a file or directory was made on a device
   mounted read-only.

31 EMLINK  Too many links
   An attempt to make more than the maximum number of links
   (1000) to a file.

32 EPIPE  Broken pipe
   A write on a pipe for which there is no process to read the data.
   This condition normally generates a signal; the error is returned if
   the signal is ignored.

33 EDOM  Math argument
   The argument of a function in the math package (3M) is out of the
   domain of the function.

34 ERANGE  Result too large
   The value of a function in the math package (3M) is not represent-
   able within machine precision.

DEFINITIONS

Process ID
   Each active process in the system is uniquely identified by a positive integer
   called a process ID. The range of this ID is from 0 to 30,000.

Parent Process ID
   A new process is created by a currently active process; see fork(2). The
   parent process ID of a process is the process ID of its creator.

Process Group ID
   Each active process is a member of a process group that is identified by a
   positive integer called the process group ID. This ID is the process ID of
   the group leader. This grouping permits the signaling of related processes;
   see kill(2).

Tty Group ID
   Each active process can be a member of a terminal group that is identified
   by a positive integer called the tty group ID. This grouping is used to termi-
   nate a group of related process upon termination of one of the processes
   in the group; see exit(2) and signal(2).

Real User ID and Real Group ID
   Each user allowed on the system is identified by a positive integer called a
   real user ID.
Each user is also a member of a group. The group is identified by a positive integer called the real group ID.

An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.

**Effective User ID and Effective Group ID**

An active process has an effective user ID and an effective group ID that are used to determine file access permissions (see below). The effective user ID and effective group ID are equal to the process's real user ID and real group ID respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group ID bit set; see `exec(2)`.

**Super-user**

A process is recognized as a super-user process and is granted special privileges if its effective user ID is 0.

**Special Processes**

The processes with a process ID of 0 and a process ID of 1 are special processes and are referred to as `proc0` and `proc1`.

`Proc0` is the scheduler. `Proc1` is the initialization process (`init`). `Proc1` is the ancestor of every other process in the system and is used to control the process structure.

**File Name**

Names consisting of up to 14 characters may be used to name an ordinary file, special file or directory.

These characters may be selected from the set of all character values excluding 0 (null) and the ASCII code for `/` (slash).

Note that it is generally unwise to use `*`, `?`, `[`, or `]` as part of file names because of the special meaning attached to these characters by the shell. See `sh(1)`.

**Path Name and Path Prefix**

A path name is a null-terminated character string starting with an optional slash (`/`), followed by zero or more directory names separated by slashes, optionally followed by a file name.

More precisely, a path name is a null-terminated character string constructed as follows:

\[
\text{<path-name>} ::= \text{<file-name>1}<\text{path-prefix}><\text{file-name>1}|
\text{<path-prefix>} ::= \text{<rtprefix>1}<\text{rtprefix>1}
\text{<rtprefix>} ::= \text{<dirname>1}<\text{dirname>}/\text{<rtprefix>1}<\text{rtprefix>1}<\text{dirname>}/
\]

where `<file-name>` is a string of 1 to 14 characters other than the ASCII slash and null, and `<dirname>` is a string of 1 to 14 characters (other than the ASCII slash and null) that names a directory.

If a path name begins with a slash, the path search begins at the `root` directory. Otherwise, the search begins from the current working directory.

A slash by itself names the root directory.

Unless specifically stated otherwise, the null path name is treated as if it named a non-existent file.

**Directory**

Directory entries are called links. By convention, a directory contains at least two links, `. ` and `..`, referred to as `dot` and `dot-dot` respectively. Dot refers to the directory itself and dot-dot refers to its parent directory.
Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. A process’s root directory need not be the root directory of the root file system.

File Access Permissions.
Read, write, and execute/search permissions on a file are granted to a process if one or more of the following are true:

- The process’s effective user ID is super-user.
- The process’s effective user ID matches the user ID of the owner of the file and the appropriate access bit of the “owner” portion (0700) of the file mode is set.
- The process’s effective user ID does not match the user ID of the owner of the file, and the process’s group ID matches the group of the file and the appropriate access bit of the “group” portion (070) of the file mode is set.
- The process’s effective user ID does not match the user ID of the owner of the file, and the process’s effective group ID does not match the group ID of the file, and the appropriate access bit of the “other” portion (07) of the file mode is set.

Otherwise, the corresponding permissions are denied.

SEE ALSO
intro(3).
NAME
access — determine accessibility of a file

SYNOPSIS
int access (path, amode)
char *path;
int amode;

DESCRIPTION
Path points to a path name naming a file. Access checks the named file for accessibility according to the bit pattern contained in amode, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID. The bit pattern contained in amode is constructed as follows:

04 read
02 write
01 execute (search)
00 check existence of file

Access to the file is denied if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]
Read, write, or execute (search) permission is requested for a null path name. [ENOENT]
The named file does not exist. [ENOENT]
Search permission is denied on a component of the path prefix. [EACCES]
Write access is requested for a file on a read-only file system. [EROFS]
Write access is requested for a pure procedure (shared text) file that is being executed. [ETXTBSY]
Permission bits of the file mode do not permit the requested access. [EACCES]
Path points outside the process’s allocated address space. [EFAULT]
The owner of a file has permission checked with respect to the “owner” read, write, and execute mode bits, members of the file’s group other than the owner have permissions checked with respect to the “group” mode bits, and all others have permissions checked with respect to the “other” mode bits.

RETURN VALUE
If the requested access is permitted, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
chmod(2), stat(2).
NAME
acct — enable or disable process accounting

SYNOPSIS
int acct (path)
char *path;

DESCRIPTION
Acct is used to enable or disable the system’s process accounting routine. If the routine is enabled, an accounting record will be written on an accounting file for each process that terminates. Termination can be caused by one of two things: an exit call or a signal; see exit(2) and signal(2). The effective user ID of the calling process must be super-user to use this call.

Path points to a path name naming the accounting file. The accounting file format is given in acct(5).

The accounting routine is enabled if path is non-zero and no errors occur during the system call. It is disabled if path is zero and no errors occur during the system call.

Acct will fail if one or more of the following are true:

- The effective user ID of the calling process is not super-user. [EPERM]
- An attempt is being made to enable accounting when it is already enabled. [EBUSY]
- A component of the path prefix is not a directory. [ENOTDIR]
- One or more components of the accounting file’s path name do not exist. [ENOENT]
- A component of the path prefix denies search permission. [EACCES]
- The file named by path is not an ordinary file. [EACCES]
- Mode permission is denied for the named accounting file. [EACCES]
- The named file is a directory. [EISDIR]
- The named file resides on a read-only file system. [EROFS]
- Path points to an illegal address. [EFAULT]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
acct(1M), acct(5).
NAME
alarm — set a process's alarm clock

SYNOPSIS
unsigned alarm (sec)
unsigned sec;

DESCRIPTION
Alarm instructs the calling process's alarm clock to send the signal
SIGALRM to the calling process after the number of real time seconds
specified by sec have elapsed; see signal(2).

Alarm requests are not stacked; successive calls reset the calling process's
alarm clock.

If sec is 0, any previously made alarm request is canceled.

RETURN VALUE
Alarm returns the amount of time previously remaining in the calling
process's alarm clock.

SEE ALSO
pause(2), signal(2).
NAME
brk, sbrk — change data segment space allocation

SYNOPSIS
int brk (endds)
char *endds;
char *sbrk (incr)
int incr;

DESCRIPTION
Brk and sbrk are used to change dynamically the amount of space allocated
for the calling process's data segment; see exec(2). The change is made by
resetting the process's break value. The break value is the address of the
first location beyond the end of the data segment. The amount of allocated
space increases as the break value increases.

Brk sets the break value to endds and changes the allocated space accord­
ingly.

Sbrk adds incr bytes to the break value and changes the allocated space
accordingly. Incr can be negative, in which case the amount of allocated
space is decreased.

Brk and sbrk will fail without making any change in the allocated space if
such a change would result in more space being allocated than is allowed by
a system-imposed maximum (see ulimit(2)). [ENOMEM]

RETURN VALUE
Upon successful completion, brk returns a value of 0 and sbrk returns the
old break value. Otherwise, a value of -1 is returned and errno is set to
indicate the error.

SEE ALSO
exec(2).
NAME
   chdir — change working directory

SYNOPSIS
   int chdir (path)
   char *path;

DESCRIPTION
   Path points to the path name of a directory. Chdir causes the named directory to
   become the current working directory, the starting point for path searches for path names
   not beginning with /.

   Chdir will fail and the current working directory will be unchanged if one or more of
   the following are true:

   A component of the path name is not a directory. [ENOTDIR]
   The named directory does not exist. [ENOENT]
   Search permission is denied for any component of the path name. [EACCESS]

   Path points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
   Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is
   returned and errno is set to indicate the error.

SEE ALSO
   chroot(2).
NAME
chmod — change mode of file

SYNOPSIS
int chmod (path, mode)
char *path;
int mode;

DESCRIPTION
Path points to a path name naming a file. Chmod sets the access permission portion of the named file's mode according to the bit pattern contained in mode.

Access permission bits are interpreted as follows:

- `04000` Set user ID on execution.
- `02000` Set group ID on execution.
- `01000` Save text image after execution
- `00400` Read by owner
- `00200` Write by owner
- `00100` Execute (or search if a directory) by owner
- `00070` Read, write, execute (search) by group
- `00007` Read, write, execute (search) by others

The effective user ID of the process must match the owner of the file or be super-user to change the mode of a file.

If the effective user ID of the process is not super-user, mode bit `01000` (save text image on execution) is cleared.

If the effective user ID of the process is not super-user or the effective group ID of the process does not match the group ID of the file, mode bit `02000` (set group ID on execution) is cleared.

If an executable file is prepared for sharing then mode bit `01000` prevents the system from abandoning the swap-space image of the program-text portion of the file when its last user terminates. Thus, when the next user of the file executes it, the text need not be read from the file system but can simply be swapped in, saving time.

Chmod will fail and the file mode will be unchanged if one or more of the following are true:

- A component of the path prefix is not a directory. [ENOTDIR]
- The named file does not exist. [ENOENT]
- Search permission is denied on a component of the path prefix. [EACCES]
- The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]
- The named file resides on a read-only file system. [EROFS]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO
chown(2), mknod(2).
NAME
chown — change owner and group of a file

SYNOPSIS
int chown (path, owner, group)
char *path;
int owner, group;

DESCRIPTION
Path points to a path name naming a file. The owner ID and group ID of the named file are set to the numeric values contained in owner and group respectively.

Only processes with effective user ID equal to the file owner or super-user may change the ownership of a file.

If chown is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Chown will fail and the owner and group of the named file will remain unchanged if one or more of the following are true:

- A component of the path prefix is not a directory. [ENOTDIR]
- The named file does not exist. [ENOENT]
- Search permission is denied on a component of the path prefix. [EACCES]
- The effective user ID does not match the owner of the file and the effective user ID is not super-user. [EPERM]
- The named file resides on a read-only file system. [EROFS]

Path points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO
chmod(2).
NAME
chroot — change root directory

SYNOPSIS
int chroot (path)
char *path;

DESCRIPTION
Path points to a path name naming a directory. Chroot causes the named
directory to become the root directory, the starting point for path searches
for path names beginning with /.

The effective user ID of the process must be super-user to change the root
directory.

The .. entry in the root directory is interpreted to mean the root directory
itself. Thus, .. can not be used to access files outside the subtree rooted at
the root directory.

Chroot will fail and the root directory will remain unchanged if one or more
of the following are true:

Any component of the path name is not a directory. [ENOTDIR]
The named directory does not exist. [ENOENT]
The effective user ID is not super-user. [EPERM]
Path points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value
of −1 is returned and errno is set to indicate the error.

SEE ALSO
chdir(2).
NAME
close — close a file descriptor

SYNOPSIS
int close (fildes)
int fildes;

DESCRIPTION
Fildes is a file descriptor obtained from a creat, open, dup, fcntl, or pipe system call. Close closes the file descriptor indicated by fildes.

Close will fail if fildes is not a valid open file descriptor. [EBADF]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
creat(2), dup(2), exec(2), fcntl(2), open(2), pipe(2).
NAME
creat — create a new file or rewrite an existing one

SYNOPSIS
int creat (path, mode)
char *path;
int mode;

DESCRIPTION
Create creates a new ordinary file or prepares to rewrite an existing file
named by the path name pointed to by path.

If the file exists, the length is truncated to 0 and the mode and owner are
unchanged. Otherwise, the file’s owner ID is set to the process’s effective
user ID, the file’s group ID is set to the process’s effective group ID, and
the low-order 12 bits of the file mode are set to the value of mode modified
as follows:

All bits set in the process’s file mode creation mask are cleared.
See umask(2).

The “save text image after execution bit” of the mode is cleared.
See chmod(2).

Upon successful completion, a non-negative integer, namely the file
descriptor, is returned and the file is open for writing, even if the mode
does not permit writing. The file pointer is set to the beginning of the file.
The file descriptor is set to remain open across exec system calls. See
fcntl(2). No process may have more than 20 files open simultaneously. A
new file may be created with a mode that forbids writing.

Create will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOTDIR]
A component of the path prefix does not exist. [ENOENT]
Search permission is denied on a component of the path prefix.
[EACCES]
The path name is null. [ENOENT]
The file does not exist and the directory in which the file is to be
created does not permit writing. [EACCES]
The named file resides or would reside on a read-only file system.
[EROFS]
The file is a pure procedure (shared text) file that is being execu-
ted. [ETXTBSY]
The file exists and write permission is denied. [EACCES]
The named file is an existing directory. [EISDIR]
Twenty (20) file descriptors are currently open. [EMFILE]
Path points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion, a non-negative integer, namely the file
descriptor, is returned. Otherwise, a value of −1 is returned and errno is
set to indicate the error.

SEE ALSO
close(2), dup(2), lseek(2), open(2), read(2), umask(2), write(2).
NAME
dup — duplicate an open file descriptor

SYNOPSIS
int dup (fildes)
int fildes;

DESCRIPTION
Fildes is a file descriptor obtained from a creat, open, dup, fcntl, or pipe system call. Dup returns a new file descriptor having the following in common with the original:

Same open file (or pipe).
Same file pointer. (i.e., both file descriptors share one file pointer.)
Same access mode (read, write or read/write).

The new file descriptor is set to remain open across exec system calls. See fcntl(2).
The file descriptor returned is the lowest one available.

Dup will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]
Twenty (20) file descriptors are currently open. [EMFILE]

RETURN VALUE
Upon successful completion a non-negative integer, namely the file descriptor, is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
creat(2), close(2), exec(2), fcntl(2), open(2), pipe(2).
EXEC(2) NAME
execl, execv, execle, execve, execlp, execvp — execute a file

SYNOPSIS

int execl (path, arg0, arg1, ..., argn, 0)  
char *path, *arg0, *arg1, ..., *argn;

int execv (path, argv)  
char *path, *argv[ ];

int execline (path, arg0, arg1, ..., argn, 0, envp)  
char *path, *arg0, *arg1, ..., *argn, *envp[ ];

int execve (path, argv, envp);  
char *path, *argv[ ], *envp[ ];

int execlp (file, arg0, arg1, ..., argn, 0)  
char *file, *arg0, *arg1, ..., *argn;

int execvp (file, argv)  
char *file, *argv[ ];

DESCRIPTION

Exec in all its forms transforms the calling process into a new process. The new process is constructed from an ordinary, executable file called the new process file. This file consists of a header (see a.out(5)), a text segment, and a data segment. The data segment contains an initialized portion and an uninitialized portion (bss). There can be no return from a successful exec because the calling process is overlaid by the new process.

Path points to a path name that identifies the new process file.

File points to the new process file. The path prefix for this file is obtained by a search of the directories passed as the environment line "PATH = " (see environ(7)). The environment is supplied by the shell (see sh(1)).

Arg0, arg1, ..., argn are pointers to null-terminated character strings. These strings constitute the argument list available to the new process. By convention, at least arg0 must be present and point to a string that is the same as path (or its last component).

Argv is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process. By convention, argv must have at least one member, and it must point to a string that is the same as path (or its last component). Argv is terminated by a null pointer.

Envp is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process. Envp is terminated by a null pointer.

File descriptors open in the calling process remain open in the new process, except for those whose close-on-exec flag is set; see fcntl(2). For those file descriptors that remain open, the file pointer is unchanged.

Signals set to terminate the calling process will be set to terminate the new process. Signals set to be ignored by the calling process will be set to be ignored by the new process. Signals set to be caught by the calling process will be set to terminate new process; see signal(2).

If the set-user-ID mode bit of the new process file is set (see chmod(2)), exec sets the effective user ID of the new process to the owner ID of the new process file. Similarly, if the set-group-ID mode bit of the new process file is set, the effective group ID of the new process is set to the group ID of the new process file. The real user ID and real group ID of the new process
remain the same as those of the calling process.
Profiling is disabled for the new process; see profl(2).
The new process also inherits the following attributes from the calling process:

- nice value (see nice(2))
- process ID
- parent process ID
- process group ID
- tty group ID (see exit(2) and signal(2))
- trace flag (see ptrace(2) request 0)
- time left until an alarm clock signal (see alarm(2))
- current working directory
- root directory
- file mode creation mask (see umask(2))
- file size limit (see ulimit(2))
- utime, stime, cutime, and cstime (see times(2))

Exec will fail and return to the calling process if one or more of the following are true:

- One or more components of the new process file’s path name do not exist. [ENOENT]
- A component of the new process file’s path prefix is not a directory. [ENOTDIR]
- Search permission is denied for a directory listed in the new process file’s path prefix. [EACCES]
- The new process file is not an ordinary file. [EACCES]
- The new process file mode denies execution permission. [EACCES]
- The new process file has the appropriate access permission, but has an invalid magic number in its header. [ENOEXEC]
- The new process file is a pure procedure (shared text) file that is currently open for writing by some process. [ETXTBSY]
- The new process requires more memory than is allowed by the system-imposed maximum MAXMEM. [ENOMEM]
- The number of bytes in the new process’s argument list is greater than the system-imposed limit of 5120 bytes. [E2BIG]
- The new process file is not as long as indicated by the size values in its header. [EFAULT]
- Path, argv, or envp point to an illegal address. [EFAULT]

RETURN VALUE

If exec returns to the calling process an error has occurred; the return value will be -1 and errno will be set to indicate the error.

SEE ALSO

exit(2), fork(2).
NAME
exit — terminate process

SYNOPSIS
exit (status)
int status;

DESCRIPTION
Exit terminates the calling process with the following consequences:

All of the file descriptors open in the calling process are closed.

If the parent process of the calling process is executing a wait, it is notified of the calling process’s termination and the low order eight bits (i.e., bits 0377) of status are made available to it; see wait(2).

If the parent process of the calling process is not executing a wait, the calling process is transformed into a zombie process. A zombie process is a process that only occupies a slot in the process table, it has no other space allocated either in user or kernel space. The process table slot that it occupies is partially overlaid with time accounting information (see <sys/proc.h>) to be used by times.

The parent process ID of all of the calling process’s existing child processes and zombie processes is set to 1. This means the initialization process (see intro(2)) inherits each of these processes.

An accounting record is written on the accounting file if the system’s accounting routine is enabled; see acct(2).

If the process ID, tty group ID, and process group ID of the calling process are equal, the SIGHUP signal is sent to each processes that has a process group ID equal to that of the calling process.

SEE ALSO
signal(2), wait(2).

WARNING
See WARNING in signal(2).
NAME
fcnt! - file control

SYNOPSIS
#include <fcntl.h>
int fcntl (fildes, cmd, arg)
int fildes, cmd, arg;

DESCRIPTION
Fcntl provides for control over open files. Fildes is an open file descriptor obtained from a creat, open, dup, fcntl, or pipe system call.
The cmds available are:
F_DUPFD Return a new file descriptor as follows:
Lowest numbered available file descriptor greater than or equal to arg.
Same open file (or pipe) as the original file.
Same file pointer as the original file (i.e., both file descriptors share one file pointer).
Same access mode (read, write or read/write).
Same file status flags (i.e., both file descriptors share the same file status flags).
The close-on-exec flag associated with the new file descriptor is set to remain open across exec(2) system calls.
F_GETFD Get the close-on-exec flag associated with the file descriptor fildes. If the low-order bit is 0 the file will remain open across exec, otherwise the file will be closed upon execution of exec.
F_SETFD Set the close-on-exec flag associated with fildes to the low-order bit of arg (0 or 1 as above).
F_GETFL Get file status flags.
F_SETFL Set file status flags to arg. Only certain flags can be set; see fcntl(7).

Fcntl will fail if one or more of the following are true:
Fildes is not a valid open file descriptor. [EBADF]
Cmd is F_DUPFD and 20 file descriptors are currently open. [EMFILE]
Cmd is F_DUPFD and arg is negative or greater than 20. [EINVAL]

RETURN VALUE
Upon successful completion, the value returned depends on cmd as follows:
F_DUPFD A new file descriptor.
F_GETFD Value of flag (only the low-order bit is defined).
F_SETFD Value other than -1.
F_GETFL Value of file flags.
F_SETFL Value other than -1.
Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
close(2), exec(2), open(2), fcntl(7).
NAME
fork — create a new process

SYNOPSIS
int fork ()

DESCRIPTION
Fork causes creation of a new process. The new process (child process) is an exact copy of the calling process (parent process) except for the following:

The child process has a unique process ID.

The child process has a different parent process ID (i.e., the process ID of the parent process).

The child process has its own copy of the parent's file descriptors. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.

The child process's utime, stime, cutime, and cstime are set to 0; see times(2).

Fork returns a value of 0 to the child process.

Fork returns the process ID of the child process to the parent process.

Fork will fail and no child process will be created if one or more of the following are true:

The system-imposed limit on the total number of processes under execution would be exceeded. [EAGAIN]

The system-imposed limit on the total number of processes under execution by a single user would be exceeded. [EAGAIN]

RETURN VALUE
Upon successful completion, fork returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and errno is set to indicate the error.

SEE ALSO
exec(2), wait(2).
NAME
getpid, getpgid, getppid — get process, process group, and parent process IDs

SYNOPSIS
int getpid()
int getpgid()
int getppid()

DESCRIPTION
Getpid returns the process ID of the calling process.
Getpgid returns the process group ID of the calling process.
Getppid returns the parent process ID of the calling process.

SEE ALSO
exec(2), fork(2), intro(2), setpgid(2), signal(2).
NAME
getuid, geteuid, getgid, getegid — get real user, effective user, real group,
and effective group IDs

SYNOPSIS
int getuid()
int geteuid()
int getgid()
int getegid()

DESCRIPTION
Getuid returns the real user ID of the calling process.
Geteuid returns the effective user ID of the calling process.
Getgid returns the real group ID of the calling process.
Getegid returns the effective group ID of the calling process.

SEE ALSO
intro(2), setuid(2).
NAME
ioctl — control device

SYNOPSIS
#include <sys/ioctl.h>
ioctl(fildes, request, arg)

DESCRIPTION
ioctl performs a variety of functions on character special files (devices). The writeups of various devices in Section 4 discuss how ioctl applies to them.

ioctl will fail if one or more of the following are true:

- Fildes is not a valid open file descriptor. [EBADF]
- Fildes is not associated with a character special device. [ENOTTY]
- Request or arg is not valid. See tty(4). [EINVAL]

RETURN VALUE
If an error has occurred, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
tty(4).
NAME

kill — send a signal to a process or a group of processes

SYNOPSIS

int kill (pid, sig)
int pid, sig;

DESCRIPTION

Kill sends a signal to a process or a group of processes. The process or
group of processes to which the signal is to be sent is specified by pid. The
signal that is to be sent is specified by sig and is either one from the list
given in signal(2), or 0. If sig is 0 (the null signal), error checking is per­
formed but no signal is actually sent. This can be used to check the validity
of pid.

The effective user ID of the sending process must match the real user ID of
the receiving process unless, the effective user ID of the sending process is
super-user, or the process is sending to itself.

The processes with a process ID of 0 and a process ID of 1 are special pro­
cesses (see intro(2)) and will be referred to below as proc0 and procl
respectively.

If pid is greater than zero, sig will be sent to the process whose process ID
is equal to pid. Pid may equal 1.

If pid is 0, sig will be sent to all processes excluding proc0 and procl whose
process group ID is equal to the process group ID of the sender.

If pid is -1 and the effective user ID of the sender is not super-user, sig
will be sent to all processes excluding proc0 and procl whose real user ID is
equal to the effective user ID of the sender.

If pid is -1 and the effective user ID of the sender is super-user, sig will be
sent to all processes excluding proc0 and procl.

If pid is negative but not -1, sig will be sent to all processes whose process
group ID is equal to the absolute value of pid.

Kill will fail and no signal will be sent if one or more of the following are
true:

Sig is not a valid signal number. [EINVAL]

No process can be found corresponding to that specified by pid. [ESRCH]

The sending process is not sending to itself, its effective user ID is
not super-user, and its effective user ID does not match the real
user ID of the receiving process. [EPERM]

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value
of -1 is returned and errno is set to indicate the error.

SEE ALSO

kill(1), getpid(2), setpgrp(2), signal(2).
NAME
link — link to a file

SYNOPSIS
int link (path1, path2)
char *path1, *path2;

DESCRIPTION
Path1 points to a path name naming an existing file. Path2 points to a path name naming the new directory entry to be created. Link creates a new link (directory entry) for the existing file.

Link will fail and no link will be created if one or more of the following are true:

A component of either path prefix is not a directory. [ENOTDIR]
A component of either path prefix does not exist. [ENOENT]
A component of either path prefix denies search permission. [EACCES]
The file named by path1 does not exist. [ENOENT]
The link named by path2 exists. [EXIST]
The file named by path1 is a directory and the effective user ID is not super-user. [EPERM]
The link named by path2 and the file named by path1 are on different logical devices (file systems). [EXDEV]
Path2 points to a null path name. [ENOENT]
The requested link requires writing in a directory with a mode that denies write permission. [EACCES]
The requested link requires writing in a directory on a read-only file system. [EROFS]
Path points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
link(1M), unlink(2).
NAME
lseek — move read/write file pointer

SYNOPSIS
long lseek (fildes, offset, whence)
int fildes;
long offset;
int whence;

DESCRIPTION
Fildes is a file descriptor returned from a creat, open, dup, or fcntl system call. Lseek sets the file pointer associated with fildes as follows:

If whence is 0, the pointer is set to offset bytes.
If whence is 1, the pointer is set to its current location plus offset.
If whence is 2, the pointer is set to the size of the file plus offset.

Upon successful completion, the resulting pointer location as measured in bytes from the beginning of the file is returned.
Lseek will fail and the file pointer will remain unchanged if one or more of the following are true:

Fildes is not an open file descriptor. [EBADF]
Fildes is associated with a pipe or fifo. [ESPIPE]
Whence is not 0, 1 or 2. [EINVAL and SIGSYS signal]
The resulting file pointer would be negative. [EINVAL]

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

RETURN VALUE
Upon successful completion, a non-negative integer indicating the file pointer value is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO
creat(2), dup(2), fcntl(2), open(2).
NAME
mknod — make a directory, or a special or ordinary file

SYNOPSIS
int mknod (path, mode, dev)
char *path;
int mode, dev;

DESCRIPTION
Mknod creates a new file named by the path name pointed to by path. The mode of the new file is initialized from mode. Where the value of mode is interpreted as follows:

0170000 file type; one of the following:
  0010000 fifo special
  0020000 character special
  0040000 directory
  0060000 block special
  0100000 or 0000000 ordinary file

0004000 set user ID on execution
0002000 set group ID on execution
0001000 save text image after execution
0000777 access permissions; constructed from the following
  0000400 read by owner
  0000200 write by owner
  0000100 execute (search on directory) by owner
  0000070 read, write, execute (search) by group
  0000007 read, write, execute (search) by others

Values of mode other than those above are undefined and should not be used.

The file's owner ID is set to the process’s effective user ID. The file’s group ID is set to the process’s effective group ID.

The low-order 9 bits of mode are modified by the process’s file mode creation mask: all bits set in the process’s file mode creation mask are cleared. See umask(2). If mode indicates a block or character special file, dev is a configuration dependent specification of a character or block I/O device. If mode does not indicate a block special or character special device, dev is ignored.

Mknod may be invoked only by the super-user for file types other than FIFO special.

Mknod will fail and the new file will not be created if one or more of the following are true:

- The process’s effective user ID is not super-user. [EPERM]
- A component of the path prefix is not a directory. [ENOTDIR]
- A component of the path prefix does not exist. [ENOENT]
- The directory in which the file is to be created is located on a read-only file system. [EROFS]
- The named file exists. [EEXIST]
- Path points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.
SEE ALSO

`mkdir(1), mknod(1M), chmod(2), exec(2), umask(2), fs(5).`
NAME
  mount — mount a file system

SYNOPSIS
  int mount (spec, dir, rwflag)
  char *spec, *dir;
  int rwflag;

DESCRIPTION
  Mount requests that a removable file system contained on the block special
  file identified by spec be mounted on the directory identified by dir. Spec
  and dir are pointers to path names.

  Upon successful completion, references to the file dir will refer to the root
directory on the mounted file system.

  The low-order bit of rwflag is used to control write permission on the
mounted file system; if 1, writing is forbidden, otherwise writing is permit­
ted according to individual file accessibility.

  Mount may be invoked only by the super-user.

  Mount will fail if one or more of the following are true:
    The effective user ID is not super-user. [EPERM]
    Any of the named files does not exist. [ENOENT]
    A component of a path prefix is not a directory. [ENOTDIR]
    Spec is not a block special device. [ENOTBLK]
    The device associated with spec does not exist. [ENXIO]
    Dir is not a directory. [ENOTDIR]
    Spec or dir points outside the process's allocated address space.
      [EFAULT]
    Dir is currently mounted on, is someone's current working direc­
tory or is otherwise busy. [EBUSY]
    The device associated with spec is currently mounted. [EBUSY]

RETURN VALUE
  Upon successful completion a value of 0 is returned. Otherwise, a value of
-1 is returned and errno is set to indicate the error.

SEE ALSO
  mount(1M), umount(2).
NAME
nice — change priority of a process

SYNOPSIS
int nice (incr)
int incr;

DESCRIPTION
Nice adds the value of incr to the nice value of the calling process. A process’s nice value is a positive number for which a more positive value results in lower CPU priority.

A maximum nice value of 39 and a minimum nice value of 0 are imposed by the system. Requests for values above or below these limits result in the nice value being set to the corresponding limit.

Nice will fail and not change the nice value if incr is negative and the effective user ID of the calling process is not super-user. [EPERM]

RETURN VALUE
Upon successful completion, nice returns the new nice value minus 20. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO
nice(1), exec(2).
NAME
open — open for reading or writing

SYNOPSIS
#include <fcntl.h>
int open (path, oflag[], mode)
char *path;
int oflag, mode;

DESCRIPTION
Path points to a path name naming a file. Open opens a file descriptor for
the named file and sets the file status flags according to the value of oflag.
Oflag values are constructed by or-ing flags from the following list (only
one of the first three flags below may be used):
O_RDONLY Open for reading only.
O_WRONLY Open for writing only.
O_RDWR Open for reading and writing.
O_NDELAY This flag may affect subsequent reads and writes. See
read(2) and write(2).

When opening a FIFO with O_RDONLY or O_WRONLY set:
If O_NDELAY is set:
An open for reading-only will return without delay.
An open for writing-only will return an error if no
process currently has the file open for reading.

If O_NDELAY is clear:
An open for reading-only will block until a process
opens the file for writing. An open for writing-only
will block until a process opens the file for reading.

When opening a file associated with a communication line:
If O_NDELAY is set:
The open will return without waiting for carrier.

If O_NDELAY is clear:
The open will block until carrier is present.

O_APPEND If set, the file pointer will be set to the end of the file prior
to each write.

O_CREAT If the file exists, this flag has no effect. Otherwise, the file's
owner ID is set to the process's effective user ID, the file's
group ID is set to the process's effective group ID, and the
low-order 12 bits of the file mode are set to the value of
mode modified as follows (see creat(2)):
All bits set in the process's file mode creation mask
are cleared. See umask(2).
The "save text image after execution bit" of the
mode is cleared. See chmod(2).

O_TRUNC If the file exists, its length is truncated to 0 and the mode
and owner are unchanged.

O_EXCL If O_EXCL and O_CREAT are set, open will fail if the file
exists.
Upon successful completion a non-negative integer, the file descriptor, is returned.

The file pointer used to mark the current position within the file is set to the beginning of the file.

The new file descriptor is set to remain open across exec system calls. See fcntl(2).

No process may have more than 20 file descriptors open simultaneously.

The named file is opened unless one or more of the following are true:

- A component of the path prefix is not a directory. [ENOTDIR]
- O_CREAT is not set and the named file does not exist. [ENOENT]
- A component of the path prefix denies search permission. [EACCES]
- Oflag permission is denied for the named file. [EACCES]
- The named file is a directory and oflag is write or read/write. [EISDIR]
- The named file resides on a read-only file system and oflag is write or read/write. [EROFS]
- Twenty (20) file descriptors are currently open. [EMFILE]
- The named file is a character special or block special file, and the device associated with this special file does not exist. [ENXIO]
- The file is a pure procedure (shared text) file that is being executed and oflag is write or read/write. [ETXTBSY]
- Path points outside the process’s allocated address space. [EFAULT]
- O_CREAT and O_EXCL are set, and the named file exists. [EEXIST]
- O_NDELAY is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading. [ENXIO]

RETURN VALUE

Upon successful completion, a non-negative integer, namely a file descriptor, is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO

close(2), creat(2), dup(2), fcntl(2), lseek(2), read(2), write(2).
NAME
  pause — suspend process until signal

SYNOPSIS
  pause ( )

DESCRIPTION
  Pause suspends the calling process until it receives a signal. The signal
  must be one that is not currently set to be ignored by the calling process.
  If the signal causes termination of the calling process, pause will not return.
  If the signal is caught by the calling process and control is returned from
  the signal catching-function (see signal(2)), the calling process resumes
  execution from the point of suspension; with a return value of -1 from
  pause and errno set to EINTR.

SEE ALSO
  alarm(2), kill(2), signal(2), wait(2).
NAME
pipe — create an interprocess channel

SYNOPSIS
int pipe (fildes)
int fildes[2];

DESCRIPTION
Pipe creates an I/O mechanism called a pipe and returns two file descriptors, fildes[0] and fildes[1]. fildes[0] is opened for reading and fildes[1] is opened for writing.

 Writes up to 5120 bytes of data are buffered by the pipe before the writing process is blocked. A read on file descriptor fildes[0] accesses the data written to fildes[1] on a first-in-first-out basis.

 No process may have more than 20 file descriptors open simultaneously.

 Pipe will fail if 19 or more file descriptors are currently open. [EMFILE]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
sh(1), read(2), write(2).
NAME
profil — execution time profile

SYNOPSIS
profil (buff, bufsiz, offset, scale)
char *buff;
int bufsiz, offset, scale;

DESCRIPTION
Buff points to an area of core whose length (in bytes) is given by bufsiz. After this call, the user’s program counter (pc) is examined each clock tick (60th second); offset is subtracted from it, and the result multiplied by scale. If the resulting number corresponds to a word inside buff, that word is incremented.

The scale is interpreted as an unsigned, fixed-point fraction with binary point at the left: 0177777 (octal) gives a 1-1 mapping of pc’s to words in buff; 077777 (octal) maps each pair of instruction words together. 02(8) maps all instructions onto the beginning of buff (producing a non-interrupting core clock).

Profiling is turned off by giving a scale of 0 or 1. It is rendered ineffective by giving a bufsiz of 0. Profiling is turned off when an exec is executed, but remains on in child and parent both after a fork. Profiling will be turned off if an update in buff would cause a memory fault.

RETURN VALUE
Not defined.

SEE ALSO
profil(1), monitor(3C).
NAME
ptrace — process trace

SYNOPSIS
int ptrace (request, pid, addr, data);
int request, pid, addr, data;

DESCRIPTION
Ptrace provides a means by which a parent process may control the execution of a child process. Its primary use is for the implementation of breakpoint debugging; see adb(1). The child process behaves normally until it encounters a signal (see signal(2) for the list), at which time it enters a stopped state and its parent is notified via wait(2). When the child is in the stopped state, its parent can examine and modify its "core image" using ptrace. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

The request argument determines the precise action to be taken by ptrace and is one of the following:

0 This request must be issued by the child process if it is to be traced by its parent. It turns on the child's trace flag that stipulates that the child should be left in a stopped state upon receipt of a signal rather than the state specified by func; see signal(2). The pid, addr, and data arguments are ignored, and a return value is not defined for this request. Peculiar results will ensue if the parent does not expect to trace the child.

The remainder of the requests can only be used by the parent process. For each, pid is the process ID of the child. The child must be in a stopped state before these requests are made.

1, 2 With these requests, the word at location addr in the address space of the child is returned to the parent process. If I and D space are separated (as on PDP-11s), request 1 returns a word from I space, and request 2 returns a word from D space. If I and D space are not separated (as on the VAX-11/780), either request 1 or request 2 may be used with equal results. The data argument is ignored. These two requests will fail if addr is not the start address of a word, or is outside the USER area, in which case a value of -1 is returned to the parent process and the parent's errno is set to EIO.

3 With this request, the word at location addr in the child's USER area in the system's address space (see <sys/user.h>) is returned to the parent process. Addresses in this area range from 0 to 1024 on the PDP-11s and 0 to 2048 on the VAX. The data argument is ignored. This request will fail if addr is not the start address of a word or is outside the USER area, in which case a value of -1 is returned to the parent process and the parent's errno is set to EIO.

4, 5 With these requests, the value given by the data argument is written into the address space of the child at location addr. If I and D space are separated (as on PDP-11s), request 4 writes a word into I space, and request 5 writes a word into D space. If I and D space are not separated (as on the VAX), either request 4 or request 5 may be used with equal results. Upon successful completion, the value written into the
address space of the child is returned to the parent. These two requests will fail if *addr* is a location in a pure procedure space and another process is executing in that space, or *addr* is not the start address of a word. Upon failure a value of −1 is returned to the parent process and the parent's *errno* is set to EIO.

6 With this request, a few entries in the child's USER area can be written. *Data* gives the value that is to be written and *addr* is the location of the entry. The few entries that can be written are:

- the general registers (i.e., registers 0—7 on PDP-11s, and registers 0—15 on the VAX)
- the floating point status register and six floating point registers on PDP-11s
- certain bits of the Processor Status Word on PDP-11s (i.e., bits 0—4, and 8—11)
- certain bits of the Processor Status Longword on the VAX (i.e., bits 0—7, 16—20, and 30—31)

7 This request causes the child to resume execution. If the *data* argument is 0, all pending signals including the one that caused the child to stop are canceled before it resumes execution. If the *data* argument is a valid signal number, the child resumes execution as if it had incurred that signal and any other pending signals are canceled. The *addr* argument must be equal to 1 for this request. Upon successful completion, the value of *data* is returned to the parent. This request will fail if *data* is not 0 or a valid signal number, in which case a value of −1 is returned to the parent process and the parent's *errno* is set to EIO.

8 This request causes the child to terminate with the same consequences as *exit*(2).

9 This request sets the trace bit in the Processor Status Word of the child (i.e., bit 4 on PDP-11s; bit 30 on the VAX) and then executes the same steps as listed above for request 7. The trace bit causes an interrupt upon completion of one machine instruction. This effectively allows single stepping of the child.

Note: the trace bit remains set after an interrupt on PDP-11s but is turned off after an interrupt on the VAX.

To forestall possible fraud, *ptrace* inhibits the set-user-id facility on subsequent *exec*(2) calls. If a traced process calls *exec*, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

**GENERAL ERRORS**

*Ptrace* will in general fail if one or more of the following are true:

- *Request* is an illegal number. [EIO]
- *Pid* identifies a child that does not exist or has not executed a *ptrace* with request 0. [ESRCH]

**SEE ALSO**

NAME
read — read from file

SYNOPSIS
int read (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

DESCRIPTION
fildes is a file descriptor obtained from a creat, open, dup,fcntl, or pipe system call.

Read attempts to read nbyte bytes from the file associated with fildes into the buffer pointed to by buf.

On devices capable of seeking, the read starts at a position in the file given by the file pointer associated with fildes. Upon return from read, the file pointer is incremented by the number of bytes actually read.

Devices that are incapable of seeking always read from the current position. The value of a file pointer associated with such a file is undefined.

Upon successful completion, read returns the number of bytes actually read and placed in the buffer; this number may be less than nbyte if the file is associated with a communication line (see ioctl(2) and tty(4)), or if the number of bytes left in the file is less than nbyte bytes. A value of 0 is returned when an end-of-file has been reached.

When attempting to read from an empty pipe (or FIFO):

   If O_NDELAY is set, the read will return a 0.
   If O_NDELAY is clear, the read will block until data is written to the file or the file is no longer open for writing.

When attempting to read a file associated with a tty that has no data currently available:

   If O_NDELAY is set, the read will return a 0.
   If O_NDELAY is clear, the read will block until data becomes available.

Read will fail if one or more of the following are true:

   fildes is not a valid file descriptor open for reading. [EBADF]
   Buf points outside the allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion a non-negative integer is returned indicating the number of bytes actually read. Otherwise, a -1 is returned and errno is set to indicate the error.

SEE ALSO
creat(2), dup(2), fcntl(2), ioctl(2), open(2), pipe(2), tty(4).
NAME
   setpgrp — set process group ID

SYNOPSIS
   int setpgrp ()

DESCRIPTION
   Setpgrp sets the process group ID of the calling process to the process ID of
   the calling process and returns the new process group ID.

RETURN VALUE
   Setpgrp returns the value of the new process group ID.

SEE ALSO
   exec(2), fork(2), getpid(2), intro(2), kill(2), signal(2).
NAME
setuid, setgid — set user and group IDs

SYNOPSIS
int setuid (uid)
int uid;
int setgid (gid)
int gid;

DESCRIPTION
Setuid is used to set the real user ID and effective user ID of the calling process.
Setgid is used to set the real group ID and effective group ID of the calling process.
If the effective user ID of the calling process is super-user, the real user (group) ID and effective user (group) ID are set to uid (gid).
If the effective user ID of the calling process is not super-user, but its real user (group) ID is equal to uid (gid), the effective user (group) ID is set to uid (gid).
Setuid will fail if the real user (group) ID of the calling process is not equal to uid (gid) and its effective user ID is not super-user. [EPERM]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
getuid(2), intro(2).
NAME
signal — specify what to do upon receipt of a signal

SYNOPSIS
\#include <signal.h>

int (*signal (int sig, int (*func)()) ()
int sig;
int (*func)();

DESCRIPTION
Signal allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. Sig specifies the signal and func specifies the choice.

Sig can be assigned any of the following except SIGKILL:

<table>
<thead>
<tr>
<th>Sig</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>hangup</td>
</tr>
<tr>
<td>SIGINT</td>
<td>interrupt</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>quit</td>
</tr>
<tr>
<td>SIGILL</td>
<td>illegal instruction (not reset when caught)</td>
</tr>
<tr>
<td>SICTRAP</td>
<td>trace trap (not reset when caught)</td>
</tr>
<tr>
<td>SIGIOT</td>
<td>IOT instruction</td>
</tr>
<tr>
<td>SIGEMT</td>
<td>EMT instruction</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>floating point exception</td>
</tr>
<tr>
<td>SICKILL</td>
<td>kill (cannot be caught or ignored)</td>
</tr>
<tr>
<td>SIGBUS</td>
<td>bus error</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>segmentation violation</td>
</tr>
<tr>
<td>SIGSYS</td>
<td>bad argument to system call</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>write on a pipe with no one to read it</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>alarm clock</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>software termination signal</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>user defined signal 1</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>user defined signal 2</td>
</tr>
<tr>
<td>SIGCLD</td>
<td>death of a child (see WARNING below)</td>
</tr>
<tr>
<td>SIGPWR</td>
<td>power fail (see WARNING below)</td>
</tr>
</tbody>
</table>

See below for the significance of the asterisk in the above list.

Func is assigned one of three values: SIG_DFL, SIG_IGN, or a function address. The actions prescribed by these values of are as follows:

SIG_DFL — terminate process upon receipt of a signal

Upon receipt of the signal sig, the receiving process is to be terminated with the following consequences:

All of the receiving process's open file descriptors will be closed.

If the parent process of the receiving process is executing a wait, it will be notified of the termination of the receiving process and the terminating signal's number will be made available to the parent process; see wait(2).

If the parent process of the receiving process is not executing a wait, the receiving process will be transformed into a zombie process (see exit(2) for definition of zombie process).

The parent process ID of each of the receiving process's existing child processes and zombie processes will be set to 1. This means the initialization process (see intro(2)) inherits each of these processes.
An accounting record will be written on the accounting file if the system's accounting routine is enabled; see `acct(2)`.

If the receiving process's process ID, tty group ID, and process group ID are equal, the signal SIGHUP will be sent to all of the processes that have a process group ID equal to the process group ID of the receiving process.

A "core image" will be made in the current working directory of the receiving process if `sig` is one for which an asterisk appears in the above list and the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named core exists and is writable or can be created. If the file must be created, it will have the following properties:

- a mode of 0666 modified by the file creation mask (see `umask(2)`)
- a file owner ID that is the same as the effective user ID of the receiving process
- a file group ID that is the same as the effective group ID of the receiving process

**SIG_IGN** — ignore signal

The signal `sig` is to be ignored.

Note: the signal SIGKILL cannot be ignored.

**function address** — catch signal

Upon receipt of the signal `sig`, the receiving process is to execute the signal-catching function pointed to by `func`. The signal number `sig` will be passed as the only argument to the signal-catching function.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted and the value of `func` for the caught signal will be set to SIG_DFL unless the signal is SIGILL, SIGTRAP, SIGCLD, or SIGPWR.

When a signal that is to be caught occurs during a `read`, a `write`, an `open`, or an `ioctl` system call on a slow device (like a terminal; but not a file), during a `pause` system call, or during a `wait` system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal catching function will be executed and then the interrupted system call will return a `-1` to the calling process with `errno` set to EINTR.

Note: the signal SIGKILL cannot be caught.

A call to `signal` cancels a pending signal `sig` except for a pending SIGKILL signal.

Signal will fail if one or more of the following are true:

- `sig` is an illegal signal number, including SIGKILL. [EINVAL]
- `Func` points to an illegal address. [EFAULT]

**RETURN VALUE**

Upon successful completion, `signal` returns the previous value of `func` for the specified signal `sig`. Otherwise, a value of `-1` is returned and `errno` is...
set to indicate the error.

SEE ALSO
kill(1), kill(2), pause(2), ptrace(2), wait(2), setjmp(3C).

WARNING
Two other signals that behave differently than the signals described above exist in this release of the system; they are:

```
SIGCLD  18 death of a child (not reset when caught)
SIGPWR  19 power fail (not reset when caught)
```

There is no guarantee that, in future releases of UNIX, these signals will continue to behave as described below; they are included only for compatibility with other versions of UNIX. Their use in new programs is strongly discouraged.

For these signals, `func` is assigned one of three values: SIG_DFL, SIG_IGN, or a `function address`. The actions prescribed by these values of are as follows:

```
SIG_DFL - ignore signal
  The signal is to be ignored.
SIG_IGN - ignore signal
  The signal is to be ignored. Also, if `sig` is SIGCLD, the calling
  process’s child processes will not create zombie processes when
  they terminate; see `exit(2)`.
function address - catch signal
  If the signal is SIGPWR, the action to be taken is the same as
  that described above for `func` equal to `function address`. The
  same is true if the signal is SIGCLD except, that while the pro-
  cess is executing the signal-catching function any received
  SIGCLD signals will be queued and the signal-catching function
  will be continually reentered until the queue is empty.
```

The SIGCLD affects two other system calls (`wait(2)`, and `exit(2)`) in the following ways:

```
wait  If the `func` value of SIGCLD is set to SIG_IGN and a `wait` is ex-
       ecuted, the `wait` will block until all of the calling process’s child
       processes terminate; it will then return a value of -1 with `errno`
       set to ECHILD.
exit  If in the exiting process’s parent process the `func` value of
       SIGCLD is set to SIG_IGN, the exiting process will not create a
       zombie process.
```

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set SIGCLD to be caught.
NAME
stat, fstat — get file status

SYNOPSIS
#include <sys/types.h>
#include <sys/stat.h>
int stat (path, buf)
    char *path;
    struct stat *buf;
int fstat (fildes, buf)
    int fildes;
    struct stat *buf;

DESCRIPTION
Path points to a path name naming a file. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable. Stat obtains information about the named file.

Similarly, fstat obtains information about an open file known by the file descriptor fildes, obtained from a successful open, creat, dup, fcntl, or pipe system call.

Buf is a pointer to a stat structure into which information is placed concerning the file.

The contents of the structure pointed to by buf include the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ushort st_mode;</td>
<td>File mode; see mknod(2) /*</td>
</tr>
<tr>
<td>ino_t st_ino;</td>
<td>Inode number */</td>
</tr>
<tr>
<td>dev_t st_dev;</td>
<td>ID of device containing /*</td>
</tr>
<tr>
<td>dev_t st_rdev;</td>
<td>ID of device */</td>
</tr>
<tr>
<td>short st_nlink;</td>
<td>Number of links */</td>
</tr>
<tr>
<td>ushort st_uid;</td>
<td>User ID of the file’s owner */</td>
</tr>
<tr>
<td>ushort st_gid;</td>
<td>Group ID of the file’s group */</td>
</tr>
<tr>
<td>off_t st_size;</td>
<td>File size in bytes */</td>
</tr>
<tr>
<td>time_t st_atime;</td>
<td>Time of last access */</td>
</tr>
<tr>
<td>time_t st_mtime;</td>
<td>Time of last data modification */</td>
</tr>
<tr>
<td>time_t st_ctime;</td>
<td>Time of last file status change */</td>
</tr>
</tbody>
</table>

st_atime Time when file data was last accessed. Changed by the following system calls: creat(2), mknod(2), pipe(2), utime(2), and read(2).

st_mtime Time when data was last modified. Changed by the following system calls: creat(2), mknod(2), pipe(2), utime(2), and write(2).

st_ctime Time when file status was last changed. Changed by the following system calls: chmod(2), chown(2), creat(2), link(2), mknod(2), pipe(2), unlink(2), utime(2), and write(2).

Stat will fail if one or more of the following are true:

A component of the path prefix is not a directory. [ENOENTDIR]

The named file does not exist. [ENOENT]
Search permission is denied for a component of the path prefix. [EACCES]

Buf or path points to an invalid address. [EFAULT]

Fstat will fail if one or more of the following are true:

Fildes is not a valid open file descriptor. [EBADF]

Buf points to an invalid address. [EFAULT]

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of
-1 is returned and errno is set to indicate the error.

SEE ALSO

chmod(2), chown(2), creat(2), link(2), mknod(2), time(2), unlink(2).
NAME
stime — set time

SYNOPSIS
int stime (tp)
long *tp;

DESCRIPTION
Stime sets the system's idea of the time and date. Tp points to the value of
time as measured in seconds from 00:00:00 GMT January 1, 1970.
Stime will fail if the effective user ID of the calling process is not super-
user. [EPERM]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value
of −1 is returned and errno is set to indicate the error.

SEE ALSO
time(2).
NAME
  sync — update super-block

SYNOPSIS
  sync ( )

DESCRIPTION
  Sync causes all information in memory that should be on disk to be written out. This includes modified super blocks, modified i-nodes, and delayed block I/O.

  It should be used by programs which examine a file system, for example fsck, df, etc. It is mandatory before a boot.

  The writing, although scheduled, is not necessarily complete upon return from sync.

SEE ALSO
  sync(1M).
NAME  
  time — get time

SYNOPSIS  
  long time ((long *) 0)
  long time (tloc)
  long *tloc;

DESCRIPTION  
  Time returns the value of time in seconds since 00:00:00 GMT, January 1, 1970.

  If tloc (taken as an integer) is non-zero, the return value is also stored in the location to which tloc points.

  Time will fail if tloc points to an illegal address. [EFAULT]

RETURN VALUE  
  Upon successful completion, time returns the value of time. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO  
  stime(2).
NAME

times — get process and child process times

SYNOPSIS

long times (buffer)
struct tbuffer *buffer;
struct tbuffer {
    long utime;
    long stime;
    long cutime;
    long cstime;
}

DESCRIPTION

Times fills the structure pointed to by buffer with time-accounting information. This information comes from the calling process and each of its terminated child processes for which it has executed a wait.

All times are in 60ths of a second.

Utime is the CPU time used while executing instructions in the user space of the calling process.

Stime is the CPU time used by the system on behalf of the calling process.

Cutime is the sum of the utimes and cutimes of the child processes.

Cstime is the sum of the stimes and cstimes of the child processes.

Times will fail if buffer points to an illegal address. [EFAULT]

RETURN VALUE

Upon successful completion, times returns the elapsed real time, in 60ths of a second, since an arbitrary point in the past (e.g., system start-up time). This point does not change from one invocation of times to another. If times fails, a -1 is returned and errno is set to indicate the error.

SEE ALSO

exec(2), fork(2), time(2), wait(2).
NAME
ulimit — get and set user limits

SYNOPSIS
long ulimit (cmd, newlimit)
int cmd;
long newlimit;

DESCRIPTION
This function provides for control over process limits. The cmd values available are:

1  Get the process's file size limit. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read.

2  Set the process's file size limit to the value of newlimit. Any process may decrease this limit, but only a process with an effective user ID of super-user may increase the limit. Ulimit will fail and the limit will be unchanged if a process with an effective user ID other than super-user attempts to increase its file size limit. [EPERM]

3  Get the maximum possible break value. See brk(2).

RETURN VALUE
Upon successful completion, a non-negative value is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO
brk(2), write(2).
NAME
 umask — set and get file creation mask

SYNOPSIS
 int umask (cmask)
 int cmask;

DESCRIPTION

_Umask_ sets the process’s file mode creation mask to _cmask_ and returns the
previous value of the mask. Only the low-order 9 bits of _cmask_ and the file
mode creation mask are used.

RETURN VALUE

The previous value of the file mode creation mask is returned.

SEE ALSO

mkdir(1), mknod(1M), sh(1), chmod(2), creat(2), mknod(2), open(2).
NAME
umount — unmount a file system

SYNOPSIS
int umount (spec)
char *spec;

DESCRIPTION
Umount requests that a previously mounted file system contained on the
block special device identified by spec be unmounted. Spec is a pointer to a
path name. After unmounting the file system, the directory upon which
the file system was mounted reverts to its ordinary interpretation.
Umount may be invoked only by the super-user.
Umount will fail if one or more of the following are true:
The process’s effective user ID is not super-user. [EPERM]
Spec does not exist. [ENXIO]
Spec is not a block special device. [ENOTBLK]
Spec is not mounted. [EINVAL]
A file on spec is busy. [EBUSY]
Spec points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion a value of 0 is returned. Otherwise, a value of
−1 is returned and errno is set to indicate the error.

SEE ALSO
mount(1M), mount(2).
NAME
uname — get name of current UNIX system

SYNOPSIS
#include <sys/utsname.h>
int uname (name)
struct utsname *name;

DESCRIPTION
Uname stores information identifying the current UNIX system in the structure pointed to by name.
Uname uses the structure defined in <sys/utsname.h>:

struct utsname {
    char sysname[9];
    char nodename[9];
    char release[9];
    char version[9];
};

extern struct utsname utsname;

Uname returns a null-terminated character string naming the current UNIX system in the character array sysname. Similarly, nodename contains the name that the system is known by on a communications network. Release and version further identify the operating system.

Uname will fail if name points to an invalid address. [EFAULT]

RETURN VALUE
Upon successful completion, a non-negative value is returned. Otherwise, -1 is returned and errno is set to indicate the error.

SEE ALSO
uname(1).
NAME
unlink — remove directory entry

SYNOPSIS
int unlink (path)
    char *path;

DESCRIPTION
Unlink removes the directory entry named by the path name pointed to be
path.

The named file is unlinked unless one or more of the following are true:

- A component of the path prefix is not a directory. [ENOTDIR]
- The named file does not exist. [ENOENT]
- Search permission is denied for a component of the path prefix. [EACCES]
- Write permission is denied on the directory containing the link to
be removed. [EACCES]
- The named file is a directory and the effective user ID of the pro-
cess is not super-user. [EPERM]
- The entry to be unlinked is the mount point for a mounted file sys-
tem. [EBUSY]
- The entry to be unlinked is the last link to a pure procedure
(shared text) file that is being executed. [ETXTBSY]
- The directory entry to be unlinked is part of a read-only file system.
[EROFS]

Path points outside the process’s allocated address space. [EFAULT]

When all links to a file have been removed and no process has the file
open, the space occupied by the file is freed and the file ceases to exist. If
one or more processes have the file open when the last link is removed, the
removal is postponed until all references to the file have been closed.

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value
of −1 is returned and errno is set to indicate the error.

SEE ALSO
rm(1), close(2), link(2), open(2).
NAME
ustat — get file system statistics

SYNOPSIS
#include <sys/types.h>
#include <ustat.h>

int ustat (dev, buf)
int dev;
struct ustat *buf;

DESCRIPTION
Ustat returns information about a mounted file system. Dev is a device
number identifying a device containing a mounted file system. Buf is a
pointer to a ustat structure that includes to following elements:

    daddr_t f_tfree;       /* Total free blocks */
    ino_t f_tinode;        /* Number of free inodes */
    char f_fname[6];      /* Filsys name */
    char f_fpack[6];      /* Filsys pack name */

Ustat will fail if one or more of the following are true:

    Dev is not the device number of a device containing a mounted file
    system. [EINVAL]

    Buf points outside the process’s allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value
of -1 is returned and errno is set to indicate the error.

SEE ALSO
stat(2), fs(5).
NAME
utime — set file access and modification times

SYNOPSIS
#include <sys/types.h>
int utime (path, times)
char *path;
struct utimbuf *times;

DESCRIPTION
Path points to a path name naming a file. Ut ime sets the access and modification times of the named file.

If times is NULL, the access and modification times of the file are set to the current time. A process must be the owner of the file or have write permission to use utime in this manner.

If times is not NULL, times is interpreted as a pointer to a utimbuf structure and the access and modification times are set to the values contained in the designated structure. Only the owner of the file or the super-user may use utime this way.

The times in the following structure are measured in seconds since 00:00:00 GMT, Jan. 1, 1970.

struct utimbuf {
    time_t actime; /* access time */
    time_t modtime; /* modification time */
};

Ut ime will fail if one or more of the following are true:

The named file does not exist. [ENOENT]

A component of the path prefix is not a directory. [ENOTDIR]

Search permission is denied by a component of the path prefix. [EACCES]

The effective user ID is not super-user and not the owner of the file and times is not NULL. [EPERM]

The effective user ID is not super-user and not the owner of the file and times is NULL and write access is denied. [EACCES]

The file system containing the file is mounted read-only. [EROFS]

Times is not NULL and points outside the process's allocated address space. [EFAULT]

Path points outside the process's allocated address space. [EFAULT]

RETURN VALUE
Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
stat(2).
NAME
wait — wait for child process to stop or terminate

SYNOPSIS
int wait (stat_loc)
int *stat_loc;
int wait ((int *)0)

DESCRIPTION
Wait suspends the calling process until it receives a signal that is to be caught (see signal(2)), or until any one of the calling process's child processes stops in a trace mode (see ptrace(2)) or terminates. If a child process stopped or terminated prior to the call on wait, return is immediate.

If stat_loc (taken as an integer) is non-zero, 16 bits of information called status are stored in the low order 16 bits of the location pointed to by stat_loc. Status can be used to differentiate between stopped and terminated child processes and if the child process terminated, status identifies the cause of termination and pass useful information to the parent. This is accomplished in the following manner:

If the child process stopped, the high order 8 bits of status will be zero and the low order 8 bits will be set equal to 0177.

If the child process terminated due to an exit call, the low order 8 bits of status will be zero and the high order 8 bits will contain the low order 8 bits of the argument that the child process passed to exit; see exit(2).

If the child process terminated due to a signal, the high order 8 bits of status will be zero and the low order 8 bits will contain the number of the signal that caused the termination. In addition, if the low order seventh bit (i.e., bit 200) is set, a "core image" will have been produced; see signal(2).

If a parent process terminates without waiting for its child processes to terminate, the parent process ID of each child process is set to 1. This means the initialization process inherits the child processes; see intro(2).

Wait will fail and return immediately if one or more of the following are true:

The calling process has no existing unwaited-for child processes.  
[ECHILD]

Stat_loc points to an illegal address. [EFAULT]

RETURN VALUE
If wait returns due to the receipt of a signal, a value of -1 is returned to the calling process and errno is set to EINTR. If wait returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of -1 is returned and errno is set to indicate the error.

SEE ALSO
exec(2), exit(2), fork(2), pause(2), signal(2).

WARNING
See WARNING in signal(2).
NAME
write — write on a file

SYNOPSIS

int write (fildes, buf, nbyte)
int fildes;
char *buf;
unsigned nbyte;

DESCRIPTION

Fildes is a file descriptor obtained from a creat, open, dup, fcntl, or pipe system call.

Write attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with the fildes.

On devices capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file pointer. Upon return from write, the file pointer is incremented by the number of bytes actually written.

On devices incapable of seeking, writing always takes place starting at the current position. The value of a file pointer associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file pointer will be set to the end of the file prior to each write.

Write will fail and the file pointer will remain unchanged if one or more of the following are true:

Fildes is not a valid file descriptor open for writing. [EBADF]
An attempt is made to write to a pipe that is not open for reading by any process. [EPIPE and SIGPIPE signal]
An attempt was made to write a file that exceeds the process’s file size limit or the maximum file size. See ulimit(2). [EFBIG]

Buf points outside the process’s allocated address space. [EFAULT]

If a write requests that more bytes be written than there is room for (e.g., the ulimit (see ulimit(2)) or the physical end of a medium), only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes will give a failure return (except as noted below).

If the file being written is a pipe (or FIFO), no partial writes will be permitted. Thus, the write will fail if a write of nbyte bytes would exceed a limit.

If the file being written is a pipe (or FIFO) and the O_NDELAY flag of the file flag word is set, then write to a full pipe (or FIFO) will return a count of 0. Otherwise (O_NDELAY clear), writes to a full pipe (or FIFO) will block until space becomes available.

RETURN VALUE

Upon successful completion the number of bytes actually written is returned. Otherwise, -1 is returned and errno is set to indicate the error.

SEE ALSO

creat(2), dup(2), lseek(2), open(2), pipe(2), ulimit(2).
INTRO(3)

NAME
intro — introduction to subroutines and libraries

SYNOPSIS

#include <stdio.h>
#include <math.h>

DESCRIPTION

This section describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume. Certain major collections are identified by a letter after the section number:

(3C) These functions, together with those of Section 2 and those marked (3S), constitute library libc, which is automatically loaded by the C compiler, cc(1). The link editor ld(1) searches this library under the -lc option. Declarations for some of these functions may be obtained from #include files indicated on the appropriate pages.

(3M) These functions constitute the math library, libm. They are automatically loaded as needed by the FORTRAN compiler f77(1). The link editor searches this library under the -lm option. Declarations for these functions may be obtained from the #include file <math.h>.

(3S) These functions constitute the "standard I/O package" (see stdio(3S)). These functions are in the library libc, already mentioned. Declarations for these functions may be obtained from the #include file <stdio.h>.

(3X) Various specialized libraries. The files in which these libraries are found are given on the appropriate pages.

The descriptions of some functions refer to NULL. This is the value that is obtained by casting 0 into a character pointer. The C language guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return it, for example, to indicate an error. NULL is defined in <stdio.h> as 0; the user can include his own definition if he is not using <stdio.h>.

FILES

/lib/libc.a
/lib/libm.a

SEE ALSO

ar(1), cc(1), f77(1), ld(1), nm(1), intro(2), stdio(3S).

DIAGNOSTICS

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases, the external variable errno (see intro(2)) is set to the value EDOM or ERANGE.
NAME

a64l, l64a — convert between long and base-64 ASCII

SYNOPSIS

long a64l (s)
char *s;

char *l64a (l)
long l;

DESCRIPTION

These routines are used to maintain numbers stored in base-64 ASCII. This
is a notation by which long integers can be represented by up to six charac-
ters; each character represents a "digit" in a radix-64 notation.

The characters used to represent "digits" are . for 0, / for 1, 0 through 9
for 2—11, A through Z for 12—37, and a through z for 38—63.

A64l takes a pointer to a null-terminated base-64 representation and
returns a corresponding long value. L64a takes a long argument and
returns a pointer to the corresponding base-64 representation.

BUGS

The value returned by l64a is a pointer into a static buffer, the contents of
which are overwritten by each call.
NAME
 abort — generate an IOT fault

SYNOPSIS
 abort ( )

DESCRIPTION
 Abort causes an IOT signal to be sent to the process. This usually results in
 termination with a core dump.

 It is possible for abort to return control if SIGIOT is caught or ignored.

SEE ALSO
 adb(1), exit(2), signal(2).

DIAGNOSTICS
 Usually "abort — core dumped" from the shell.
NAME
   abs — integer absolute value

SYNOPSIS
   int abs (i)
   int i;

DESCRIPTION
   Abs returns the absolute value of its integer operand.

SEE ALSO
   fabs(3M).

BUGS
   You get what the hardware gives on the largest negative integer.
NAME
assert — program verification

SYNOPSIS
#include <assert.h>
assert (expression);

DESCRIPTION
This macro is useful for putting diagnostics into programs. When it is executed, if expression is false, it prints "Assertion failed: file XYZ, line nnn" on the standard error file and exits. XYZ is the source file and nnn the source line number of the assert statement. Compiling with the preprocessor option -DNDEBUG (see cc (1)) will cause assert to be ignored.
NAME
atof, atoi, atol — convert ASCII to numbers

SYNOPSIS

    double atof (nptr)
    char *nptr;

    int atoi (nptr)
    char *nptr;

    long atol (nptr)
    char *nptr;

DESCRIPTION

These functions convert a string pointed to by nptr to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

    Atof recognizes an optional string of tabs and spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optionally signed integer.

    Atoi and atol recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

SEE ALSO

    scanf(3S).

BUGS

There are no provisions for overflow.
NAME
j0, j1, jn, y0, y1, yn — bessel functions

SYNOPSIS
#include <math.h>
double j0 (x)
double x;
double j1 (x)
double x;
double jn (n, x);
double x;
double y0 (x)
double x;
double y1 (x)
double x;
double yn (n, x)
int n;
double x;

DESCRIPTION
These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS
Negative arguments cause y0, y1, and yn to return a huge negative value.
NAME
bsearch — binary search

SYNOPSIS
char *bsearch (key, base, nel, width, compar)
char *key;
char *base;
int nel, width;
int (*compar)();

DESCRIPTION
Bsearch is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table indicating the location at which a datum may be found. The table must be previously sorted in increasing order. The first argument is a pointer to the datum to be located in the table. The second argument is a pointer to the base of the table. The third is the number of elements in the table. The fourth is the width of an element in bytes. The last is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

DIAGNOSTICS
Zero is returned if the key can not be found in the table.

SEE ALSO
lsearch(3C), qsort(3C).
NAME
toupper, tolower, toascii — character translation

SYNOPSIS

```
#include <ctype.h>

int toupper (c)
int c;

int tolower (c)
int c;

int _toupper (c)
int c;

int _tolower (c)
int c;

int toascii (c)
int c;
```

DESCRIPTION

toupper and tolower have as domain the range of getc: the integers from
-1 through 255. If the argument of toupper represents a lower-case letter,
the result is the corresponding upper-case letter. If the argument of tolower
represents an upper-case letter, the result is the corresponding lower-case
letter. All other arguments in the domain are returned unchanged.

_toupper and _tolower are macros that accomplish the same thing as toupper
and tolower but have restricted domains and are faster. _toupper requires a
lower-case letter as its argument; its result is the corresponding upper-case
letter. _tolower requires an upper-case letter as its argument; its result is
the corresponding lower-case letter. Arguments outside the domain cause
garbage results.

Toascii yields its argument with all bits turned off that are not part of a
standard ASCII character; it is intended for compatibility with other systems.

SEE ALSO

cctype(3C).
NAME
   crypt, setkey, encrypt — DES encryption

SYNOPSIS
   char *crypt (key, salt)
   char *key, *salt;
   setkey (key)
   char *key;
   encrypt (block, ed8ag)
   char *block;
   int ed8ag;

DESCRIPTION
   Crypt is the password encryption routine. It is based on the NBS Data
   Encryption Standard (DES), with variations intended (among other things)
   to frustrate use of hardware implementations of the DES for key search.
   The first argument to crypt is a user's typed password. The second is a
   2-character string chosen from the set [a-zA-ZO-9./]; this salt string is used
   to perturb the DES algorithm in one of 4096 different ways, after which the
   password is used as the key to encrypt repeatedly a constant string. The
   returned value points to the encrypted password, in the same alphabet as
   the salt. The first two characters are the salt itself.
   The setkey and encrypt entries provide (rather primitive) access to the actual
   DES algorithm. The argument of setkey is a character array of length 64
   containing only the characters with numerical value 0 and 1. If this string
   is divided into groups of 8, the low-order bit in each group is ignored,
   leading to a 56-bit key which is set into the machine.
   The argument to the encrypt entry is likewise a character array of length 64
   containing 0's and 1's. The argument array is modified in place to a similar
   array representing the bits of the argument after having been subjected to
   the DES algorithm using the key set by setkey. If edflag is 0, the argument
   is encrypted; if non-zero, it is decrypted.

SEE ALSO
   login(1), passwd(1), getpass(3C), passwd(5).

BUGS
   The return value points to static data that are overwritten by each call.
NAME
ctermid — generate file name for terminal

SYNOPSIS

```c
#include <stdio.h>

char *ctermid(s)
char *s;
```

DESCRIPTION

*Ctermid* generates a string that refers to the controlling terminal for the current process when used as a file name.

If (int)s is zero, the string is stored in an internal static area, the contents of which are overwritten at the next call to *ctermid*, and the address of which is returned. If (int)s is non-zero, then s is assumed to point to a character array of at least L_ctermid elements; the string is placed in this array and the value of s is returned. The manifest constant L_ctermid is defined in `<stdio.h>`.

NOTES

The difference between *ctermid* and *ttyname(3C)* is that *ttyname* must be handed a file descriptor and returns the actual name of the terminal associated with that file descriptor, while *ctermid* returns a magic string (/dev/tty) that will refer to the terminal if used as a file name. Thus *ttyname* is useless unless the process already has at least one file open to a terminal.

SEE ALSO

*ttyname(3C)*.

- 1 -
NAME
ctime, localtime, gmtime, asctime, tzset — convert date and time to ASCII

SYNOPSIS
char *ctime (clock)
long *clock;
#include <time.h>
struct tm *localtime (clock)
long *clock;
struct tm *gmtime (clock)
long *clock;
char *asctime (tm)
struct tm *tm;
tzset ( )

DESCRIPTION
Ctime converts a time pointed to by clock such as returned by time(2) into
ASCII and returns a pointer to a 26-character string in the following form.
All the fields have constant width.

Sun Sep 16 01:03:52 1973

Localtime and gmtime return pointers to structures containing the broken­
down time. Localtime corrects for the time zone and possible daylight
savings time; gmtime converts directly to GMT, which is the time the UNIX
system uses. Asctime converts a broken-down time to ASCII and returns a
pointer to a 26-character string.

The structure declaration from the include file is:

struct tm {
    int tm_sec;
    int tm_min;
    int tm_hour;
    int tm_mday;
    int tm_mon;
    int tm_year;
    int tm_wday;
    int tm_yday;
    int tm_isdst;
};

These quantities give the time on a 24-hour clock, day of month (1-31),
month of year (0-11), day of week (Sunday = 0), year — 1900, day of
year (0-365), and a flag that is non-zero if daylight saving time is in effect.

The external long variable timezone contains the difference, in seconds,
between GMT and local standard time (in EST, timezone is 5*60*60); the
external variable daylight is non-zero if and only if the standard U.S.A.
Daylight Savings Time conversion should be applied. The program knows
about the peculiarities of this conversion in 1974 and 1975; if necessary, a
table for these years can be extended.

If an environment variable named TZ is present, asctime uses the contents
of the variable to override the default time zone. The value of TZ must be
a three-letter time zone name, followed by a number representing the
difference between local time and Greenwich time in hours, followed by an
optional three-letter name for a daylight time zone. For example, the set­
ting for New Jersey would be EST5EDT. The effects of setting TZ are thus
to change the values of the external variables *timezone* and *daylight*; in addition, the time zone names contained in the external variable

```
char *tzname[2] = {"EST", "EDT"};
```

are set from the environment variable. The function *tzset* sets the external variables from *TZ*; it is called by *asctime* and may also be called explicitly by the user.

**SEE ALSO**

time(2), getenv(3C), environ(7).

**BUGS**
The return values point to static data whose content is overwritten by each call.
NAME
  isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint,
isgraph, iscntrl, isascii — character classification

SYNOPSIS
  #include <ctype.h>
  int isalpha (c)
  int c;
  ...

DESCRIPTION
  These macros classify ASCII-coded integer values by table lookup. Each is a
  predicate returning nonzero for true, zero for false. Isascii is defined on all
  integer values; the rest are defined only where isascii is true and on the
  single non-ASCII value EOF (see stdio(3S)).

  isalpha  c is a letter
  isupper  c is an upper case letter
  islower  c is a lower case letter
  isdigit  c is a digit [0-9]
  isxdigit  c is a hexadecimal digit [0-9], [A-F] or [a-f]
  isalnum  c is an alphanumeric
  isspace  c is a space, tab, carriage return, new-line, vertical tab, or
           form-feed
  ispunct  c is a punctuation character (neither control nor
           alphanumeric)
  isprint  c is a printing character, code 040 (space) through 0176
           (tilde)
  isgraph  c is a printing character, like isprint except false for space
  iscntrl  c is a delete character (0177) or ordinary control character
           (less than 040).
  isascii  c is an ASCII character, code less than 0200

SEE ALSO
  ascii(7).
NAME
cuserid — character login name of the user

SYNOPSIS
#include <stdio.h>
char *cuserid (s)
char *s;

DESCRIPTION
Cuserid generates a character representation of the login name of the owner
of the current process. If (int)s is zero, this representation is generated in
an internal static area, the address of which is returned. If (int)s is non-zero,
s is assumed to point to an array of at least L_cuserid characters; the
representation is left in this array. The manifest constant L_cuserid is
defined in <stdio.h>.

DIAGNOSTICS
If the login name cannot be found, cuserid returns NULL; if s is non-zero
in this case, \0 will be placed at *s.

SEE ALSO
getlogin(3C), getpwuid(3C).

BUGS
Cuserid uses getpwnam(3C); thus the results of a user’s call to the latter
will be obliterated by a subsequent call to the former.
The name cuserid is rather a misnomer.
NAME
ecvt, fcvt — output conversion

SYNOPSIS
char *ecvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
char *fcvt (value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;
char *gcvt (value, ndigit, buf)
double value;
char *buf;

DESCRIPTION
Ecvet converts the value to a null-terminated string of ndigit ASCII digits and
returns a pointer thereto. The position of the decimal point relative to the
beginning of the string is stored indirectly through decpt (negative means to
the left of the returned digits). If the sign of the result is negative, the
word pointed to by sign is non-zero, otherwise it is zero. The low-order
digit is rounded.

Fcvt is identical to ecvt, except that the correct digit has been rounded for
Fortran F-format output of the number of digits specified by _ndigits.

Gcvt converts the value to a null-terminated ASCII string in buf and returns
a pointer to buf. It attempts to produce ndigit significant digits in Fortran F
format if possible, otherwise E format, ready for printing. Trailing zeros
may be suppressed.

SEE ALSO
printf(3S).

BUGS
The return values point to static data whose content is overwritten by each
call.
NAME
   end, etext, edata — last locations in program

SYNOPSIS
   extern end;
   extern etext;
   extern edata;

DESCRIPTION
   These names refer neither to routines nor to locations with interesting
   contents. The address of etext is the first address above the program text,
   edata above the initialized data region, and end above the uninitialized data
   region.

   When execution begins, the program break coincides with end, but the pro-
   gram break may be reset by the routines of brk(2), malloc(3C), standard
   input/output (stdio(3S)), the profile (-p) option of cc(1), and so on.
   Thus, the current value of the program break should be determined by
   "sbrk(0)" (see brk(2)).

   These symbols are accessible from assembly language if it is remembered
   that they should be prefixed by _.

SEE ALSO
   brk(2), malloc(3C).
NAME
exp, log, pow, sqrt — exponential, logarithm, power, square root functions

SYNOPSIS
#include <math.h>
double exp (x)
double x;
double log (x)
double x;
double pow (x, y)
double x, y;
double sqrt (x)
double x;

DESCRIPTION
Exp returns the exponential function of x.
Log returns the natural logarithm of x.
Pow returns \textit{x}^y.
Sqrt returns the square root of x.

SEE ALSO
intro(2), hypot(3M), sinh(3M).

DIAGNOSTICS
\textit{Exp} and \textit{pow} return a huge value when the correct value would overflow.
A truly outrageous argument may also result in \textit{errno} being set to \textit{ERANGE}.
\textit{Log} returns a huge negative value and sets \textit{errno} to \textit{EDOM} when \textit{x} is non-positive.
\textit{Pow} returns a huge negative value and sets \textit{errno} to \textit{EDOM} when \textit{x} is non-positive and \textit{y} is not an integer, or when \textit{x} and \textit{y} are both zero.
\textit{Sqrt} returns 0 and sets \textit{errno} to \textit{EDOM} when \textit{x} is negative.
NAME
fclose, fflush — close or flush a stream

SYNOPSIS
#include <stdio.h>
int fclose (stream)
FILE *stream;

int fflush (stream)
FILE *stream;

DESCRIPTION
Fclose causes any buffers for the named stream to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling exit(2).

Fflush causes any buffered data for the named output stream to be written to that file. The stream remains open.

These functions return 0 for success, and EOF if any errors were detected.

SEE ALSO
close(2), fopen(3S), setbuf(3S).
NAME

ferror, feof, clearerr, fileno — stream status inquiries

SYNOPSIS

#include <stdio.h>

int feof (stream)
FILE *stream;

int ferror (stream)
FILE *stream

clearerr (stream)
FILE *stream

fileno (stream)
FILE *stream;

DESCRIPTION

Feof returns non-zero when end of file is read on the named input stream, otherwise zero.

Ferror returns non-zero when error has occurred reading or writing the named stream, otherwise zero. Unless cleared by clearerr, the error indication lasts until the stream is closed.

Clearerr resets the error indication on the named stream.

Fileno returns the integer file descriptor associated with the stream, see open(2).

Feof, ferror, and fileno are implemented as macros; they cannot be re-declared.

SEE ALSO

open(2), fopen(3S).
NAME
floor, fabs, ceil, fmod — absolute value, floor, ceiling, remainder functions

SYNOPSIS
#include <math.h>
double floor (x)
double x;
double ceil (x)
double x;
double fmod (x, y)
double x, y;
double fabs (x)
double x;

DESCRIPTION
fabs returns |x|.

Floor returns the largest integer (as a double precision number) not greater than x.

Ceil returns the smallest integer not less than x.

fmod returns the number f such that x = iy + f, for some integer i, and 0 ≤ f < y.

SEE ALSO
abs(3C).
NAME
fopen, freopen, fdopen — open a stream

SYNOPSIS

```c
#include <stdio.h>

FILE *fopen (file-name, type)
char *file-name, *type;

FILE *freopen (file-name, type, stream)
char *file-name, *type;
FILE *stream;

FILE *fdopen (fd, type)
int fd;
char *type;
```

DESCRIPTION

`fopen` opens the file named by `file-name` and associates a stream with it. `fopen` returns a pointer to be used to identify the stream in subsequent operations.

`Type` is a character string having one of the following values:

- "r" open for reading
- "w" create for writing
- "a" append; open for writing at end of file, or create for writing
- "r+" open for update (reading and writing)
- "w+" create for update
- "a+" append; open or create for update at end of file

`freopen` substitutes the named file in place of the open `stream`. It returns the original value of `stream`. The original stream is closed, regardless of whether the open ultimately succeeds.

`freopen` is typically used to attach the preopened constant names `stdin`, `stdout`, and `stderr` to specified files.

`fdopen` associates a stream with a file descriptor obtained from `open`, `dup`, `creat`, or `pipe(2)`. The `type` of the stream must agree with the mode of the open file.

When a file is opened for update, both input and output may be done on the resulting stream. However, output may not be directly followed by input without an intervening `fseek` or `rewind`, and input may not be directly followed by output without an intervening `fseek`, `rewind`, or an input operation which encounters end of file.

SEE ALSO
open(2), fclose(3S).

DIAGNOSTICS

`fopen` and `freopen` return the pointer `NULL` if `file-name` cannot be accessed.

- 1 -
NAME
  fptrap — floating point interpreter

SYNOPSIS
  sys signal; 4; fptrap

DESCRIPTION
  Fptrap is a simulator of the 11/45 FP11-B floating point unit. It works by
  intercepting illegal instruction traps and decoding and executing the floating
  point operation codes.

  Fptrap is not supported under the UNIX 3.0 system; it is included only to
  ease conversion to other machines.

FILES
  There is a fake routine in /lib/libc.a with this name; when simulation is
  desired, the real version should be put in /lib/libc.a.

SEE ALSO
  cc(1) (-f option), signal(2).

DIAGNOSTICS
  A breakpoint trap is given when a real illegal instruction trap occurs.

BUGS
  Rounding mode is not interpreted. It's slow.
NAME
fread, fwrite — buffered binary input/output

SYNOPSIS
#include <stdio.h>
int fread ((char *) ptr, sizeof (*ptr), nitems, stream)
   FILE *stream;
int fwrite ((char *) ptr, sizeof (*ptr), nitems, stream)
   FILE *stream;

DESCRIPTION
Fread reads, into a block beginning at ptr, nitems of data of the type of *ptr
from the named input stream. It returns the number of items actually read.
Fwrite appends at most nitems of data of the type of *ptr beginning at ptr to
the named output stream. It returns the number of items actually written.

SEE ALSO
read(2), write(2), fopen(3S), getc(3S), putc(3S), gets(3S), puts(3S),
printf(3S), scanf(3S).
NAME
frexp, ldexp, modf — split into mantissa and exponent

SYNOPSIS
double frexp (value, eptr)
double value;
int *eptr;
double ldexp (value, exp)
double value;

double modf (value, iptr)
double value, *iptr;

DESCRIPTION
Frexp returns the mantissa of a double value as a double quantity, x, of
magnitude less than 1 and stores an integer n such that value = x*2**n
indirectly through eptr.

Ldexp returns the quantity value*2**exp.

Modf returns the positive fractional part of value and stores the integer part
indirectly through iptr.
NAME
fseek, ftell, rewind — reposition a stream

SYNOPSIS
#include <stdio.h>

int fseek (stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

long ftell (stream)
FILE *stream;

rewind(stream)
FILE *stream;

DESCRIPTION
Fseek sets the position of the next input or output operation on the stream. The new
position is at the signed distance offset bytes from the beginning, the current
position, or the end of the file, according as ptrname has the value 0, 1, or 2.
Fseek undoes any effects of ungetc(3S).
After fseek or rewind, the next operation on an update file may be either
input or output.

Ftell returns the current value of the offset relative to the beginning of the
file associated with the named stream. The offset is measured in bytes on
UNIX 3.0 and UNIX/RT; on some other systems, it is a magic cookie and is
the only foolproof way to obtain an offset for fseek.

Rewind(stream) is equivalent to fseek(stream, 0L, 0).

SEE ALSO
lseek(2), fopen(3S).

DIAGNOSTICS
Fseek returns non-zero for improper seeks, otherwise zero.
NAME
  gamma — log gamma function

SYNOPSIS
  #include <math.h>
  extern int signgam;
  double gamma (x)
  double x;

DESCRIPTION
  Gamma returns \( \ln|\Gamma(|x|)| \). The sign of \( \Gamma(|x|) \) is returned in the external
  integer signgam. The following C program fragment might be used to cal­
  culate \( \Gamma \):

  \[
  y = \gamma (x);
  \text{if } (y > 88.0)
  \text{error }();
  y = \exp (y) \times \text{signgam};
  \]

DIAGNOSTICS
  For negative integer arguments, a huge value is returned, and errno is set
  to EDOM.
NAME

getc, getchar, fgetc, getw — get character or word from stream

SYNOPSIS

```c
#include <stdio.h>

int getc (stream)
FILE *stream;

int getchar()

int fgetc (stream)
FILE *stream;

int getw (stream)
FILE *stream;
```

DESCRIPTION

Getc returns the next character from the named input stream.

Getchar() is identical to getc(stdin).

Fgetc behaves like getc, but is a genuine function, not a macro; it may therefore be used as an argument. Fgetc runs more slowly than getc, but takes less space per invocation.

Getw returns the next word from the named input stream. It returns the constant EOF upon end of file or error, but since that is a valid integer value, ferror and perror(3S) should be used to check the success of getw. Getw assumes no special alignment in the file.

SEE ALSO

ferror(3S), fopen(3S), fread(3S), gets(3S), putc(3S), scanf(3S).

DIAGNOSTICS

These functions return the integer constant EOF at end of file or upon read error.

A stop with message "Reading bad file" means that an attempt has been made to read from a stream that has not been opened for reading by fopen.

BUGS

Getc and its variant getchar return EOF on end of file; this is wiser than, but incompatible with, the older getchar(3S).

Because it is implemented as a macro, getc treats incorrectly a stream argument with side effects. In particular, getc(*f++); doesn’t work sensibly.
NAME
getenv — value for environment name

SYNOPSIS
char *getenv (name)
char *name;

DESCRIPTION
Getenv searches the environment list (see environ(7)) for a string of the form name=value and returns value if such a string is present, otherwise 0 (NULL).

SEE ALSO
environ(7).
GETGREN T(3C)

NAME
getgrent, getgrgid, getgrnam, setgrent, endgrent — get group file entry

SYNOPSIS
#include <grp.h>

struct group *getgrent ( );
struct group *getgrgid (gid)
int gid;
struct group *getgrnam (name)
char *name;
int setgrent ( );
int endgrent ( );

DESCRIPTION
Getgrent, getgrgid and getgrnam each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

struct group {
  char *gr_name;
  char *gr_passwd;
  int gr_gid;
  char **gr_mem;
};

The members of this structure are:

gr_name       The name of the group.
gr_passwd     The encrypted password of the group.
gr_gid        The numerical group ID.
gr_mem        Null-terminated vector of pointers to the individual member names.

Getgrent reads the next line of the file, so successive calls may be used to search the entire file. Getgrgid and getgrnam search from the beginning of the file until a matching gid or name is found, or EOF is encountered.

A call to setgrent has the effect of rewinding the group file to allow repeated searches. Endgrent may be called to close the group file when processing is complete.

FILES
/etc/group

SEE ALSO
getlogin(3C), getpwent(3C), group(5).

DIAGNOSTICS
A null pointer (0) is returned on EOF or error.

BUGS
All information is contained in a static area so it must be copied if it is to be saved.
NAME
getlogin — get login name

SYNOPSIS
char *getlogin ( );

DESCRIPTION
Getlogin returns a pointer to the login name as found in /etc/utmp. It may
be used in conjunction with getpwnam to locate the correct password file
entry when the same user ID is shared by several login names.

If getlogin is called within a process that is not attached to a typewriter, it
returns NULL. The correct procedure for determining the login name is to
call cuserid, or to call getlogin and if it fails, to call getpwuid.

FILES
/etc/utmp

SEE ALSO
cuserid(3S), getgrent(3C), getpwent(3C), utmp(5).

DIAGNOSTICS
Returns NULL if name not found.

BUGS
The return values point to static data whose content is overwritten by each
call.
NAME
getopt — get option letter from argv

SYNOPSIS
int getopt (argc, argv, optstring)
int argc;
char **argv;
char *optstring;
extern char *optarg;
extern int optind;

DESCRIPTION
Getopt returns the next option letter in argv that matches a letter in optstring. Optstring is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument that may or may not be separated from it by white space. Optarg is set to point to the start of the option argument on return from getopt.

Getopt places in optind the argv index of the next argument to be processed. Because optind is external, it is normally initialized to zero automatically before the first call to getopt.

When all options have been processed (i.e., up to the first non-option argument), getopt returns EOF. The special option -- may be used to delimit the end of the options; EOF will be returned, and -- will be skipped.

DIAGNOSTICS
Getopt prints an error message on stderr and returns a question mark (?) when it encounters an option letter not included in optstring.

EXAMPLE
The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options a and b, and the options f and o, both of which require arguments:

```c
main (argc, argv)
int argc;
char **argv;
{
    int c;
    extern int optind;
    extern char *optarg;
    ...
    while ((c = getopt (argc, argv, "abf:o:")) != EOF)
        switch (c) { 
            case 'a':
                if (bflg)
                    errflg++;
                else
                    aflg++;
                break;
            case 'b':
                if (aflg)
                    errflg++;
                else
                    bproc();
                break;
            case 'f':
                ifile = optarg;
```
GETOPT(3C)

break;
case 'o':
ofile = optarg;
bufsiza = 512;
break;
case '?':
    errflg++;
}
if (errflg) {
    fprintf (stderr, "usage: . . . ");
    exit (2);
}
for( ; optind < argc; optind++) {
    if (access (argv[optind], 4)) {
        ...
    }
}
NAME
getpass — read a password

SYNOPSIS
char *getpass (prompt)
char *prompt;

DESCRIPTION
Getpass reads a password from the file /dev/tty, or if that cannot be
opened, from the standard input, after prompting with the null-terminated
string prompt and disabling echoing. A pointer is returned to a null-
terminated string of at most 8 characters.

FILES
/dev/tty

SEE ALSO
crypt(3C).

BUGS
The return value points to static data whose content is overwritten by each
call.
NAME

getpw — get name from UID

SYNOPSIS

getpw (uid, buf)

int uid;
char *buf;

DESCRIPTION

Getpw searches the password file for the (numerical) uid, and fills in buf
with the corresponding line; it returns non-zero if uid could not be found.
The line is null-terminated.

This routine is included only for compatibility with prior systems and
should not be used; see getpwent(3C) for routines to use instead.

FILES

/etc/passwd

SEE ALSO

getpwent(3C), passwd(5).

DIAGNOSTICS

Non-zero return on error.
NAME
getpwent, getpwuid, getpwnam, setpwent, endpwent — get password file entry

SYNOPSIS
#include <pwd.h>
struct passwd *getpwent ();
struct passwd *getpwuid (uid)
int uid;
struct passwd *getpwnam (name)
char *name;
int setpwent ();
int endpwent ();

DESCRIPTION
Getpwent, getpwuid and getpwnam each returns a pointer to an object with
the following structure containing the broken-out fields of a line in the
password file.

struct passwd {
    char *pw_name;
    char *pw_passwd;
    int pw_uid;
    int pw_gid;
    char *pw_age;
    char *pw_comment;
    char *pw_gecos;
    char *pw_dir;
    char *pw_shell;
};

The pw_comment field is unused; the others have meanings described in
passwd(5).

Getpwent reads the next line in the file, so successive calls can be used to
search the entire file. Getpwuid and getpwnam search from the beginning of
the file until a matching uid or name is found, or EOF is encountered.

A call to setpwent has the effect of rewinding the password file to allow
repeated searches. Endpwent may be called to close the password file when
processing is complete.

FILES
/etc/passwd

SEE ALSO
getlogin(3C), getgrent(3C), passwd(5).

DIAGNOSTICS
Null pointer (0) returned on EOF or error.

BUGS
All information is contained in a static area so it must be copied if it is to
be saved.
NAME
gets, fgets — get a string from a stream

SYNOPSIS
```
#include <stdio.h>
char *gets (s)
char *s;
char *fgets (s, n, stream)
char *s;
int n;
FILE *stream;
```

DESCRIPTION
`Gets` reads a string into `s` from the standard input stream `stdin`. The string is terminated by a new-line character, which is replaced in `s` by a null character. `Gets` returns its argument.

`Fgets` reads `n-1` characters, or up to a new-line character (which is retained), whichever comes first, from the `stream` into the string `s`. The last character read into `s` is followed by a null character. `Fgets` returns its first argument.

SEE ALSO
ferror(3S), fopen(3S), fread(3S), getc(3S), puts(3S), scanf(3S).

DIAGNOSTICS
`Gets` and `fgets` return the constant pointer NULL upon end-of-file or error.

NOTE
`Gets` deletes the new-line ending its input, but `fgets` keeps it.
NAME
hypot — Euclidean distance

SYNOPSIS
#include <math.h>
double hypot (x, y)
double x, y;

DESCRIPTION
Hypot returns

\sqrt{x^2 + y^2},

taking precautions against unwarranted overflows.

SEE ALSO
sqrt(3M).
NAME
l3tol, ltol3 — convert between 3-byte integers and long integers

SYNOPSIS
l3tol (Ip, cp, n)
long *lp;
char *cp;
int n;
ltol3 (cp, lp, n)
char *cp;
long *lp;
int n;

DESCRIPTION
L3tol converts a list of n three-byte integers packed into a character string pointed to by cp into a list of long integers pointed to by lp.
Ltol3 performs the reverse conversion from long integers (lp) to three-byte integers (cp).
These functions are useful for file-system maintenance where the block numbers are three bytes long.

SEE ALSO
fs(5).
NAME
  logname — login name of user

SYNOPSIS
  char *logname();

DESCRIPTION
  Logname returns a pointer to the null-terminated login name; it extracts the
  $LOGNAME variable from the user's environment.
  This routine is kept in /lib/libPW.a.

FILES
  /etc/profile

SEE ALSO
  env(1), login(1), profile(5), environ(7).
NAME
lsearch — linear search and update

SYNOPSIS

    char *lsearch (key, base, nelp, width, compar)
    char *key;
    char *base;
    int *nelp;
    int width;
    int (*compar)();

DESCRIPTION

Lsearch is a linear search routine generalized from Knuth (6.1) Algorithm Q. It returns a pointer into a table indicating the location at which a datum may be found. If the item does not occur, it is added at the end of the table. The first argument is a pointer to the datum to be located in the table. The second argument is a pointer to the base of the table. The third is the address of an integer containing the number of items in the table. It is incremented if the item is added to the table. The fourth is the width of an element in bytes. The last is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return zero if the items are equal and non-zero otherwise.

BUGS

Unpredictable events can occur if there is not enough room in the table to add a new item.

SEE ALSO

bsearch(3C), qsort(3C).
NAME
malloc, free, realloc, calloc — main memory allocator

SYNOPSIS

char *malloc (size) unsigned size;
free (ptr)
char *ptr;
char *realloc (ptr, size)
char *ptr;
unsigned size;
char calloc (nelem, elsize)
unsigned elem, elsize;

DESCRIPTION
Malloc and free provide a simple general-purpose memory allocation package. Malloc returns a pointer to a block of at least size bytes beginning on a word boundary.

The argument to free is a pointer to a block previously allocated by malloc; this space is made available for further allocation, but its contents are left undisturbed.

 Needless to say, grave disorder will result if the space assigned by malloc is overrun or if some random number is handed to free.

Malloc allocates the first big enough contiguous reach of free space found in a circular search from the last block allocated or freed, coalescing adjacent free blocks as it searches. It calls sbrk (see brk(2)) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by ptr to size bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

Realloc also works if ptr points to a block freed since the last call of malloc, realloc, or calloc; thus sequences of free, malloc and realloc can exploit the search strategy of malloc to do storage compaction.

Calloc allocates space for an array of nelem elements of size elsize. The space is initialized to zeros.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS
Malloc, realloc and calloc return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. When realloc returns 0, the block pointed to by ptr may be destroyed.
NAME
mktemp — make a unique file name

SYNOPSIS

char *mktemp (template)
char *template;

DESCRIPTION

Mktemp replaces template by a unique file name, and returns the address of
the template. The template should look like a file name with six trailing
Xs, which will be replaced with a letter and the current process ID. The let-
ter will be chosen so that the resulting name does not duplicate an existing
file.

SEE ALSO

getpid(2).

BUGS

It is possible to run out of letters.
NAME
monitor — prepare execution profile

SYNOPSIS
monitor (lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[];
int bufsize, nfunc;

DESCRIPTION
An executable program created by cc -p automatically includes calls for
monitor with default parameters; monitor needn’t be called explicitly except
to gain fine control over profiling.

Monitor is an interface to profil(2). Lowpc and highpc are the addresses of
two functions; buffer is the address of a (user supplied) array of bufsize
short integers. Monitor arranges to record a histogram of periodically sam-
pelled values of the program counter, and of counts of calls of certain func-
tions, in the buffer. The lowest address sampled is that of lowpc and the
highest is just below highpc. At most nfunc call counts can be kept; only
calls of functions compiled with the profiling option -p of cc(1) are recor-
ded. For the results to be significant, especially where there are small,
heavily used routines, it is suggested that the buffer be no more than a few
times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use
extern etext();
...
monitor(2, etext, buf, bufsize, nfunc);

Etext lies just above all the program text, see end(3C).

To stop execution monitoring and write the results on the file mon.out, use
monitor(0);

prof(1) can then be used to examine the results.

FILES
mon.out

SEE ALSO
cc(1), prof(1), profil(2).
NAME
nlist — get entries from name list

SYNOPSIS
#include <a.out.h>
nlist (file-name, nl)
char *file-name;
struct nlist nl[ ];

DESCRIPTION
Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See a.out(5) for a discussion of the symbol table structure.

This subroutine is useful for examining the system name list kept in the file /unix. In this way programs can obtain system addresses that are up to date.

SEE ALSO
a.out(5).

DIAGNOSTICS
All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.
NAME
perror, sys_errlist, sys_nerr, errno — system error messages

SYNOPSIS
perror (s)
char *s;

int sys_nerr;
char *sys_errlist[ ];
int errno;

DESCRIPTION
perror produces a short error message on the standard error, describing the last error encountered during a system call from a C program. First the argument string s is printed, then a colon, then the message and a newline. To be of most use, the argument string should be the name of the program that incurred the error. The error number is taken from the external variable errno, which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings sys_errlist is provided; errno can be used as an index in this table to get the message string without the new-line. Sys_nerr is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO
intro(2).
NAME
plot — graphics interface subroutines

SYNOPSIS
openpl ()
erase ()
label (s)
char *s;
line (x1, y1, x2, y2)
circle (x, y, r)
arcc (x, y, x0, y0, x1,
move (x, y)
cont (x, y)
point (x, y)
linemod (s)
char *s;
space (x0, y0, x1, y1)
closepl ()

DESCRIPTION
These subroutines generate graphic output in a relatively device-independent manner. See plot(5) for a description of their effect. Openpl must be used before any of the others to open the device for writing. Closepl flushes the output.

String arguments to label and linemod are terminated by nulls and do not contain new-lines.

The library files listed below provide several flavors of these routines.

FILES
/usr/lib/libplot.a produces output for tplot(1G) filters
/usr/lib/lib300.a for DASI 300
/usr/lib/lib300s.a for DASI 300s
/usr/lib/lib450.a for DASI 450
/usr/lib/lib4014.a for Tektronix 4014

SEE ALSO
graph(1G), tplot(1G), plot(5).
NAME
  popen, pclose — initiate I/O to/from a process

SYNOPSIS
  #include <stdio.h>
  FILE *popen (command, type)
  char *command, *type;
  int pclose (stream)
  FILE *stream;

DESCRIPTION
  The arguments to popen are pointers to null-terminated strings containing,
  respectively, a shell command line and an I/O mode, either r for reading or
  w for writing. Popen creates a pipe between the calling process and the
  command to be executed. The value returned is a stream pointer that can
  be used (as appropriate) to write to the standard input of the command or
  read from its standard output.

  A stream opened by popen should be closed by pclose, which waits for the
  associated process to terminate and returns the exit status of the command.

  Because open files are shared, a type r command may be used as an input
  filter, and a type w as an output filter.

SEE ALSO
  pipe(2), wait(2), fclose(3S), fopen(3S), system(3S).

DIAGNOSTICS
  Popen returns a null pointer if files or processes cannot be created, or if the
  shell cannot be accessed.

  Pclose returns -1 if stream is not associated with a “popen ed” command.

BUGS
  Only one stream opened by popen can be in use at once.

  Buffered reading before opening an input filter may leave the standard
  input of that filter mispositioned. Similar problems with an output filter
  may be forestalled by careful buffer flushing, e.g. with fflush; see fclose(3S).
NAME
printf, fprintf, sprintf — output formatters

SYNOPSIS

```c
#include <stdio.h>

int printf (format [, arg ] ...)
char *format;

int fprintf (stream, format [, arg ] ...)
FILE *stream;
char *format;

int sprintf (s, format [, arg ] ...)
char *s, format;
```

DESCRIPTION

`Printf` places output on the standard output stream `stdout`. `Fprintf` places output on the named output stream. `Sprintf` places "output", followed by the null character (\0) in consecutive bytes starting at *s; it is the user’s responsibility to ensure that enough storage is available. Each function returns the number of characters transmitted (not including the \0 in the case of `sprintf`), or a negative value if an output error was encountered.

Each of these functions converts, formats, and prints its `args` under control of the `format`. The `format` is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching of zero or more `args`. The results are undefined if there are insufficient `args` for the format. If the format is exhausted while `args` remain, the excess `args` are simply ignored.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- Zero or more `flags`, which modify the meaning of the conversion specification.
- An optional decimal digit string specifying a minimum `field width`. If the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment flag (see below) has been given) to the field width;
- A `precision` that gives the minimum number of digits to appear for the d, o, u, x, or X conversions, the number of digits to appear after the decimal point for the e and f conversions, the maximum number of significant digits for the g conversion, or 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.
- An optional l specifying that a following d, o, u, x, or X conversion character applies to a long integer `arg`.
- A character that indicates the type of conversion to be applied.

A field width or precision 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 `args` specifying field width or precision must appear before the `arg` (if any) to be converted.

The flag characters and their meanings are:
The result of the conversion will be left-justified within the field.

The result of a signed conversion will always begin with a sign (+ or −).

blank
If the first character of a signed conversion is not a sign, a blank will be prepended to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.

#
This flag specifies that the value is to be converted to an “alternate form.” For c, d, s, and u conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x (X) conversion, a non-zero result will have 0x (0X) prepended to it. For e, E, f, g, and G conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conversions, trailing zeroes will not be removed from the result (which they normally are).

The conversion characters and their meanings are:

- d, o, u, x, X The integer arg is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (x and X), respectively; the letters abedef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of zero is a null string (unless the conversion is o, x, or X and the # flag is present).

- f The float or double arg is converted to decimal notation in the style “[−]ddd.ddd”, where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly 0, no decimal point appears. The E format code will produce a number with E instead of e introducing the exponent. The exponent always contains exactly two digits.

- e, E The float or double arg is converted in the style “[−]d.ddde±dd”, where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The E format code will produce a number with E instead of e introducing the exponent. The exponent always contains exactly two digits.

- g, G The float or double arg is printed in style f or e (or in style E in the case of a G format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style e will be used only if the exponent resulting from the conversion is less than −4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

- c The character arg is printed.

- s The arg is taken to be a string (character pointer) 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.

- % Print a %; no argument is converted.

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. Characters generated by
printf and fprintf are printed as if putchar had been called (see putc(3S)).

EXAMPLES
To print a date and time in the form "Sunday, July 3, 10:02", where weekday and
month are pointers to null-terminated strings:

printf("%s, %s %d, %2d:%2d", weekday, month, day, hour, min);

To print $\pi$ to 5 decimal places:

printf("\pi = %.5f", 4*atan(1.0));

SEE ALSO
ecvt(3C), putc(3S), scanf(3S), stdio(3S).
NAME
putc, putchar, fputc, putw — put character or word on a stream

SYNOPSIS
#include <stdio.h>

int pute (c, stream)
char c;
FILE *stream;
putc (c)

fpute (c, stream)
FILE *stream;

putw (w, stream)
int w;
FILE *stream;

DESCRIPTION
Putc appends the character c to the named output stream. It returns the
character written.

Putchar(c) is defined as pute(c, stdout).

Fpute behaves like pute, but is a genuine function rather than a macro; it
may therefore be used as an argument. Fpute runs more slowly than pute,
but takes less space per invocation.

Putw appends the word (i.e., integer) w to the output stream. Putw neither
assumes nor causes special alignment in the file.

The standard stream stdout is normally buffered if and only if the output
does not refer to a terminal; this default may be changed by setbuf(3S).
The standard stream stderr is by default unbuffered unconditionally, but
use of freopen(3S) will cause it to become unbuffered; setbuf, again, will set
the state to whatever is desired. When an output stream is unbuffered
information appears on the destination file or terminal as soon as written;
when it is buffered many characters are saved up and written as a block.
See also fflush(3S).

SEE ALSO
ferror(3S), fopen(3S), fwrite(3S), getc(3S), printf(3S), puts(3S).

DIAGNOSTICS
These functions return the constant EOF upon error. Since this is a good
integer, ferror(3S) should be used to detect putw errors.

BUGS
Because it is implemented as a macro, pute treats incorrectly a stream
argument with side effects. In particular, pute(c, *f++); doesn't work
sensibly.
NAME
putpwent — write password file entry

SYNOPSIS
#include <pwd.h>

int putpwent (p, f)
struct passwd *p;
FILE *f;

DESCRIPTION
Putpwent is the inverse of getpwent(3C). Given a pointer to a passwd structure created by getpwent (or getpwuid(3C) or getpwnam(3C)), putpwuid writes a line on the stream f which matches the format of /etc/passwd.

DIAGNOSTICS
Putpwent returns non-zero if an error was detected during its operation, otherwise zero.
NAME
puts, fputs — put a string on a stream

SYNOPSIS
#include <stdio.h>
int puts (s)
    char *s;
int fputs (s, stream)
    char *s;
    FILE *stream;

DESCRIPTION
    Puts copies the null-terminated string s to the standard output stream stdout
    and appends a new-line character.
    Fputs copies the null-terminated string s to the named output stream.
    Neither routine copies the terminating null character.

DIAGNOSTICS
    Both routines return EOF on error.

SEE ALSO
    ferror(3S), fopen(3S), fwrite(3S), gets(3S), printf(3S), putc(3S).

NOTES
    Puts appends a new-line, fputs does not.
NAME
qsort — quicker sort

SYNOPSIS
qsort (base, nel, width, compar)
char *base;
int nel, width;
int (*compar)( );

DESCRIPTION
Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine. It is called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO
sort(1), bsearch(3C), lsearch(3C), strcmp(3C).
NAME
rand, srand — random number generator

SYNOPSIS
srand (seed)
unsigned seed;
rand ( )

DESCRIPTION
Rand uses a multiplicative congruential random number generator with period $2^{32}$ to return successive pseudo-random numbers in the range from 0 to $2^{15} - 1$.

The generator is reinitialized by calling srand with 1 as argument. It can be set to a random starting point by calling srand with whatever you like as argument.
NAME
regex, regcmp — regular expression compile/execute

SYNOPSIS
char *regcmp(string1[,string2, ...],0);
char *string1, *string2, ...
char *regex(re,subject[,ret0, ...]);
char *re, *subject, *ret0, ...

DESCRIPTION
Regcmp compiles a regular expression and returns a pointer to the compiled
form. Malloc(3C) is used to create space for the vector. It is the user's
responsibility to free unneeded space so allocated. A zero return from
regcmp indicates an incorrect argument. Regcmp(1) has been written to
generally preclude the need for this routine at execution time.
Regex executes a compiled pattern against the subject string. Additional
arguments are passed to receive values back. Regex returns zero on failure
or a pointer to the next unmatched character on success. A global charac­
ter pointer locl points to where the match began. Regcmp and regex were
mostly borrowed from the editor, ed(1) however, the syntax and semantics
have been changed slightly. The following are the valid symbols and their
associated meanings.

[ ]*" These symbols retain their current meaning.
\$ Matches the end of the string, \n matches the new-line.
— Within brackets the minus means through. For example, [a—z]
is equivalent to [abcd...xyz]. The — can appear as itself only if
used as the last or first character. For example, the character
class expression [ ]— ] matches the characters ] and —.

+ A regular expression followed by + means one or more times.
For example, [0-9]+ is equivalent to [0-9][0-9]*.

{m} {m,} {m,u}
Integer values enclosed in {} indicate the number of times the
preceding regular expression is to be applied. m is the minimum
number and u is a number, less than 256, which is the max­
imum. If only m is present (e.g., {m}), it indicates the exact
number of times the regular expression is to be applied. {m,} is
analogous to {m,infinity}. The plus (+) and star (*) operations
are equivalent to [1,] and [0,] respectively.

(...)$n The value of the enclosed regular expression is to be returned.
The value will be stored in the (n+1)th argument following the
subject argument. At present, at most ten enclosed regular
expressions are allowed. Regex makes its assignments uncondi­
tionally.

Parentheses are used for grouping. An operator, e.g. *, +, {},
can work on a single character or a regular expression enclosed in
parenthesis. For example, (a*(cb+)*)$0.

By necessity, all the above defined symbols are special. They must, there­
fore, be escaped to be used as themselves.

EXAMPLES
Example 1:
char *cursor, *newcursor, *ptr;
...
newcursor = regex((ptr=regcmp("\n",0)),cursor);
free(ptr);
This example will match a leading new-line in the subject string pointed at by cursor.

Example 2:
char ret0[9];
char *newcursor, *name;

... name = regcmp("([A-Za-z][A-za-z0-9.][0,7]$0*,0); 
newcursor = regex(name,"123Testing321",ret0);

This example will match through the string "Testing3" and will return the address of the character after the last matched character (cursor+11). The string "Testing3" will be copied to the character array ret0.

Example 3:
#include "file.i"
char *string, *newcursor;

... newcursor = regex(name,string);

This example applies a precompiled regular expression in file.i (see regcmp(1) against string.
This routine is kept in /lib/libPW.a.

SEE ALSO
ed(1), regcmp(1), free(3C), malloc(3C).

BUGS
The user program may run out of memory if regcmp is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for malloc(3C) re-uses the same vector saving time and space:

/* user's program */

... malloc(n)
static int rebuf[256];
    return &rebuf;
}
NAME
scanf, fscanf, sscanf — formatted input conversion

SYNOPSIS
#include <stdio.h>
scanf (format [ , pointer ] ... )
char *format;
fscanf (stream, format [ , pointer ] ... )
FILE *stream;
char *format;
sscanf (s, format [ , pointer ] ... )
char *s, *format;

DESCRIPTION
Scanf reads from the standard input stream stdin. Fscanf reads from the
named input stream. Sscanf reads from the character string s. Each func-
tion reads characters, interprets them according to a format, and stores the
results in its arguments. Each expects, as arguments, a control string for-
mat described below, and a set of pointer arguments indicating where the
converted input should be stored.

The control string usually contains conversion specifications, which are
used to direct interpretation of input sequences. The control string may
contain:
1. Blanks, tabs, or new-lines, which cause input to be read up to the next
   non-white-space character.
2. An ordinary character (not %), which must match the next character of
   the input stream.
3. Conversion specifications, consisting of the character %, an optional
   assignment suppressing character *, an optional numerical maximum
   field width, and a conversion character.

A conversion specification directs the conversion of the next input field;
the result is placed in the variable pointed to by the corresponding
argument, unless assignment suppression was indicated by *. An input
field is defined as a string of non-space characters; it extends to the next
inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the
 corresponding pointer argument must usually be of a restricted type. The
following conversion characters are legal:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>a single % is expected in the input at this point; no assignment is done.</td>
</tr>
<tr>
<td>d</td>
<td>a decimal integer is expected; the corresponding argument should be an integer pointer.</td>
</tr>
<tr>
<td>o</td>
<td>an octal integer is expected; the corresponding argument should be an integer pointer.</td>
</tr>
<tr>
<td>x</td>
<td>a hexadecimal integer is expected; the corresponding argument should be an integer pointer.</td>
</tr>
<tr>
<td>s</td>
<td>a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating \0, which will be added automatically. The input field is terminated by a space character or a new-line.</td>
</tr>
<tr>
<td>c</td>
<td>a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, use...</td>
</tr>
</tbody>
</table>
%s. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.

e,f a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an E or an e, followed by an optionally signed integer.

[ ] indicates a string that is not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not a circumflex (^), the input field consists of all characters up to the first character that is not in the set between the brackets; if the first character after the left bracket is a -, the input field consists of all characters up to the first character that is in the set of the remaining characters between the brackets. The corresponding argument must point to a character array.

The conversion characters d, o, and x may be capitalized and/or preceded by l to indicate that a pointer to long rather than to int is in the argument list. Similarly, the conversion characters e and f may be capitalized and/or preceded by l to indicate that a pointer to double rather than to float is in the argument list. The character h will, some time in the future, indicate short data items.

Scanf conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

Scanf returns the number of successfully matched and assigned input items; this number can be zero in the event of an early conflict between an input character and the control string. If the input ends before the first conflict or conversion, EOF is returned.

EXAMPLES
The call:

```c
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input line:

```plaintext
25 54.32E-1 thompson
```

will assign to i the value 25, to x the value 5.432, and name will contain thompson\0. Or:

```c
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input:

```plaintext
56789 0123 56a72
```

will assign 56 to i, 789.0 to x, skip 0123, and place the string 56\0 in name. The next call to getchar (see getc(3S)) will return a.

SEE ALSO
atof(3C), getc(3S), printf(3S).

NOTE
Trailing white space (including a new-line) is left unread unless matched in the control string.
DIAGNOSTICS
These functions return EOF on end of input and a short count for missing or illegal data items.

BUGS
The success of literal matches and suppressed assignments is not directly determinable.
NAME
setbuf — assign buffering to a stream

SYNOPSIS
#include <stdio.h>
setbuf (stream, buf)
FILE *stream;
char *buf;

DESCRIPTION
Setbuf is used after a stream has been opened but before it is read or written. It causes the character array buf to be used instead of an automatically allocated buffer. If buf is the constant pointer NULL, input/output will be completely unbuffered.

A manifest constant BUFSIZ tells how big an array is needed:
char buf[BUFSIZ];

A buffer is normally obtained from malloc(3C) upon the first getc or putc(3S) on the file, except that output streams directed to terminals, and the standard error stream stderr are normally not buffered.

A common source of error is allocation of buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

SEE ALSO
fopen(3S), getc(3S), malloc(3C), putc(3S).
NAME
setjmp, longjmp — non-local goto

SYNOPSIS
#include <setjmp.h>
int setjmp (env)
jmp_buf env;
longjmp (env, val)
jmp_buf env;

DESCRIPTION
These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in env for later use by longjmp. It returns value 0.

Longjmp restores the environment saved by the last call of setjmp. It then returns in such a way that execution continues as if the call of setjmp had just returned the value val to the corresponding call to setjmp, which must not itself have returned in the interim. Longjmp cannot return the value 0. If longjmp is invoked with a second argument of 0, it will return 1. All accessible data have values as of the time longjmp was called.

SEE ALSO
signal(2).
NAME

sinh, cosh, tanh — hyperbolic functions

SYNOPSIS

```c
#include <math.h>

double sinh (x)
double x;

double cosh (x)
double x;

double tanh (x)
double x;
```

DESCRIPTION

These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS

Sinh and cosh return a huge value of appropriate sign when the correct value would overflow.
NAME
sleep — suspend execution for interval

SYNOPSIS
unsigned sleep (seconds)
unsigned seconds;

DESCRIPTION
The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) Because scheduled wakeups occur at fixed 1-second intervals, and (2) because any caught signal will terminate the sleep following execution of that signal's catching routine. Also, the suspension time may be longer than requested by an arbitrary amount due to the scheduling of other activity in the system. The value returned by sleep will be the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested sleep time, or premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling sleep; if the sleep time exceeds the time till such alarm signal, the process sleeps only until the alarm signal would have occurred, and the caller's alarm catch routine is executed just before the sleep routine returns, but if the sleep time is less than the time till such alarm, the prior alarm time is reset to go off at the same time it would have without the intervening sleep.

SEE ALSO
alarm(2), pause(2), signal(2).
NAME
ssignal, gsignal — software signals

SYNOPSIS
#include <signal.h>
int (*ssignal (sig, action))( )
int sig, (*action)( );
int gsignal (sig)
int sig;

DESCRIPTION
Ssignal and gsignal implement a software facility similar to signal(2). This facility is used by the Standard C Library to enable the user to indicate the disposition of error conditions, and is also made available to the user for his own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 15. An action for a software signal is established by a call to ssignal, and a software signal is raised by a call to gsignal. Raising a software signal causes the action established for that signal to be taken.

The first argument to ssignal is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a (user defined) action function or one of the manifest constants SIG_DFL (default) or SIG_IGN (ignore). Ssignal returns the action previously established for that signal type; if no action has been established or the signal number is illegal, ssignal returns SIG_DFL.

Gsignal raises the signal identified by its argument, sig:

If an action function has been established for sig, then that action is reset to SIG_DFL and the action function is entered with argument sig. Gsignal returns the value returned to it by the action function.
If the action for sig is SIG_IGN, gsignal returns the value 1 and takes no other action.
If the action for sig is SIG_DFL, gsignal returns the value 0 and takes no other action.
If sig has an illegal value or no action was ever specified for sig, gsignal returns the value 0 and takes no other action.

NOTES
There are some additional signals with numbers outside the range 1 through 15 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 15 are legal, although their use may interfere with the operation of the Standard C Library.
NAME
stdio — standard buffered input/output package

SYNOPSIS
#include <stdio.h>
FILE *stdin, *stdout, *stderr;

DESCRIPTION

A file with associated buffering is called a stream and is declared to be a pointer to a defined type FILE. *fopen(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are 3 open streams with constant pointers declared in the “include” file and associated with the standard open files:

```
(stdin) standard input file
(standard output file
(stderr) standard error file.
```

A constant “pointer” NULL (0) designates the null stream.

An integer constant EOF (-1) is returned upon end-of-file or error by most integer functions that deal with streams (see the individual descriptions for details).

Any program that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include <stdio.h>
```

The functions and constants mentioned in the entries of sub-class 3S of this manual are declared in that “include” file and need no further declaration. The constants and the following “functions” are implemented as macros (redclaration of these names is perilous): *getc, *getchar, *putc, *putchar, *feof, *ferror, and *fileno.

SEE ALSO
open(2), close(2), read(2), write(2), ctermid(3S), cuserid(3S), fclose(3S), *ferror(3S), *fopen(3S), *fread(3S), *fseek(3S), *getc(3S), *gets(3S), *popen(3S), *printf(3S), *putc(3S), *puts(3S), *scanf(3S), *setbuf(3S), *system(3S), *tmpnam(3S).

DIAGNOSTICS
Invalid stream pointers will usually cause grave disorder, possibly including program termination. Individual function descriptions describe the possible error conditions.
NAME
strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok — string operations

SYNOPSIS
char *strcat (s1, s2)
char *s1, *s2;
char *strncat (s1, s2, n)
char *s1, *s2;
int n;
int strcmp (s1, s2)
char *s1, *s2;
int strncmp (s1, s2, n)
char *s1, *s2;
int n;
char *strcpy (s1, s2)
char *s1, *s2;
char *strncpy (s1, s2, n)
char *s1, *s2;
int n;
int strlen (s)
char *s;
char *strchr (s, c)
char *s, c;
char *strrchr (s, c)
char *s, c;
char *strpbrk (s1, s2)
char *s1, *s2;
int strspn (s1, s2)
char *s1, *s2;
int strcspn (s1, s2)
char *s1, *s2;
char *strtok (s1, s2)
char *s1, *s2;

DESCRIPTION
These functions operate on null-terminated strings. They do not check for
overflow of any receiving string.

Strcat appends a copy of string s2 to the end of string s1. Strncat copies at
most n characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal
to, or less than 0, according as s1 is lexicographically greater than, equal to,
or less than s2. Strncmp makes the same comparison but looks at at most
n characters.

Strcpy copies string s2 to s1, stopping after the null character has been
moved. Strncpy copies exactly n characters, truncating or null-padding s2;
the target may not be null-terminated if the length of s2 is n or more.
Both return s1.

Strlen returns the number of non-null characters in s.
Strchr (strrchr) returns a pointer to the first (last) occurrence of character c in string s, or NULL if c does not occur in the string. The null character terminating a string is considered to be part of the string.

Strpbrk returns a pointer to the first occurrence in string sl of any character from string s2, or NULL if no character from s2 exists in sl.

Strspn (strcspn) returns the length of the initial segment of string sl which consists entirely of characters from (not from) string s2.

Strtok considers the string sl to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string s2. The first call (with pointer sl specified) returns a pointer to the first character of the first token, and will have written a NULL character into sl immediately following the returned token. Subsequent calls with zero for the first argument, will work through the string sl in this way until no tokens remain. The separator string s2 may be different from call to call. When no token remains in sl, a NULL is returned.

BUGS

Strcmp uses native character comparison, which is signed on PDP-11s, unsigned on other machines.

All string movement is performed character by character starting at the left. Thus overlapping moves toward the left will work as expected, but overlapping moves to the right may yield surprises.
NAME
swab — swap bytes

SYNOPSIS
swab (from, to, nbytes)
char *from, *to;
int nbytes;

DESCRIPTION
Swab copies nbytes bytes pointed to by from to the position pointed to by to, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP-11s and other machines. Nbytes should be even.
NAME
system — issue a shell command

SYNOPSIS
#include <stdio.h>
int system (string)
char *string;

DESCRIPTION
System causes the string to be given to sh(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO
sh(1), exec(2).

DIAGNOSTICS
System stops if it can't execute sh(1).
NAME
tmpfile — create a temporary file

SYNOPSIS
#include <stdio.h>
FILE *tmpfile ( )

DESCRIPTION
Tmpfile creates a temporary file and returns a corresponding FILE pointer.
Arrangements are made so that the file will automatically be deleted when
the process using it terminates. The file is opened for update.

SEE ALSO
creat(2), unlink(2), fopen(3S), mktemp(3C), tmpnam(3S).
NAME
tmpnam — create a name for a temporary file

SYNOPSIS
#include <stdio.h>

char *tmpnam (s)
char *s;

DESCRIPTION
Tmpnam generates a file name that can safely be used for a temporary file. If (int)s is zero, tmpnam leaves its result in an internal static area and returns a pointer to that area. The next call to tmpnam will destroy the contents of the area. If (int)s is nonzero, s is assumed to be the address of an array of at least L_tmpnam bytes; tmpnam places its result in that array and returns s as its value.

Tmpnam generates a different file name each time it is called.

Files created using tmpnam and either fopen or creat are only temporary in the sense that they reside in a directory intended for temporary use, and their names are unique. It is the user's responsibility to use unlink (2) to remove the file when its use is ended.

SEE ALSO
creat(2), unlink(2), fopen(3S), mktemp(3C).

BUGS
If called more than 17,576 times in a single process, tmpnam will start recycling previously used names.
Between the time a file name is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using tmpnam or mktemp, and the file names are chosen so as to render duplication by other means unlikely.
NAME
sin, cos, tan, asin, acos, atan, atan2 — trigonometric functions

SYNOPSIS
#include <math.h>
double sin (x)
double x;
double cos (x)
double x;
double asin (x)
double x;
double acos (x)
double x;
double atan (x)
double x;
double atan2 (y, x)
double x, y;

DESCRIPTION
sin, cos and tan return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.
acos returns the arc cosine in the range 0 to $\pi$.
atan returns the arc tangent of $x$ in the range $-\pi/2$ to $\pi/2$.
atan2 returns the arc tangent of $y/x$ in the range $-\pi$ to $\pi$.

DIAGNOSTICS
Arguments of magnitude greater than 1 cause asin and acos to return value 0.
NAME

ttyname, isatty — find name of a terminal

SYNOPSIS

char *ttyname (fildes)
int isatty (fildes)

DESCRIPTION

*Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *fildes.*

Isatty returns 1 if *fildes* is associated with a terminal device, 0 otherwise.

FILES

/dev/*

DIAGNOSTICS

*Ttyname returns a null pointer (0) if *fildes* does not describe a terminal device in directory /dev.*

BUGS

The return value points to static data whose content is overwritten by each call.
NAME
ungetc — push character back into input stream

SYNOPSIS
#include <stdio.h>
int ungetc (c, stream)
  char c;
  FILE *stream;

DESCRIPTION
Ungetc pushes the character c back on an input stream. That character will
be returned by the next getc call on that stream. Ungetc returns c.

One character of pushback is guaranteed provided something has been read
from the stream and the stream is actually buffered. Attempts to push
EOF are rejected.

Fseek(3S) erases all memory of pushed back characters.

SEE ALSO
fseek(3S), getc(3S), setbuf(3S).

DIAGNOSTICS
Ungetc returns EOF if it can’t push a character back.
NAME
intro — introduction to special files

DESCRIPTION
This section describes various special files that refer to specific DEC peripherals and UNIX device drivers. The names of the entries are generally derived from DEC names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding UNIX device driver are discussed where applicable.

BUGS
While the names of the entries generally refer to DEC hardware names, in certain cases these names are seemingly arbitrary for various historical reasons.
CAT(4) (PDP-11 only) CAT(4)

NAME

cat — phototypesetter interface

DESCRIPTION

Cat provides the interface to a Wang Laboratories, Inc. C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding will not be described here.

Only one process may have this file open at a time. It is write-only.

FILES

/dev/cat

SEE ALSO

troff(1).
Wang Laboratories, Inc. specification (available on request).
NAME
dj — DJ-11 asynchronous multiplexor

DESCRIPTION
Each line attached to a DJ-11 communications multiplexer behaves as described in tty(4). Line speeds and other characteristics are not programmable but are set by switches in the hardware in groups of 4 lines. Only parameters such as character delays and mapping can be altered.

FILES
/dev/tty*

SEE ALSO
tty(4).
NAME

dmc — communications link with built-in DDCMP protocol

DESCRIPTION

The DMC11 allows local connection of PDP-11 systems over high-speed (1Mb or 56kb) links and remote connection over leased (up to 19.2kb) or dial-up (up to 4,800b) lines. It implements in hardware the DDCMP data-link protocol, which includes error control. This driver handles two DMC11 devices.

FILES

/dev/dmc

BUGS

There are quite a few bugs in the DEC microcode for the different versions of the DMC11.
NAME
dn — DN-11 ACU interface

DESCRIPTION
The dn? files are write-only. The permissible codes are:
   0—9       dial 0-9
   * or :     dial *
   # or ;     dial #
   —         4 second delay for second dial tone
   e or <     end-of-number
   w or =     wait for secondary dial tone
   f         flash off hook for 1 second

The entire telephone number must be presented in a single write system call.

FILES
/dev/dn?

SEE ALSO
dh(4), du(4).
NAME
dqs — DQS-11 interface for two-point BSC

DESCRIPTION
This interface defines a special file that looks like a concatenation of Binary Synchronous Communication (BSC) text blocks. This file may be both written to and read from, but not simultaneously. Data transfer with the two-point BSC discipline is strictly half-duplex.

The device can be opened by only one process at a time. It is expected that a process that successfully opens the DQS will spawn separate subprocesses to handle reading and writing. However, no distinction is made among the several processes that may have the DQS open. For example, reads within a message, even from a single block, may be executed by several processes in sequence. The overriding constraint is that a complete message must be read from or written to the DQS before any transfer of data in the opposite direction can begin. A process that tries to write while the DQS is reading, or vice versa, will be put to sleep until the transfer of the currently active message has been completed.

A complete message consists of one or more text blocks. A message being written to the DQS is terminated by a write of zero bytes, which causes an EOT to be transmitted. A message being read from the DQS is terminated by the reception of an EOT (which is not passed on to the reader, but is registered as a read of zero bytes). By convention, an EOT follows each block which ends in an ETX.

The length of a text block cannot exceed 512 bytes, including the line prefix and appendix. These two sequences, which must be present in blocks being written and will be passed on in blocks read, are constructed from the control bytes SOH, STX, ETB, ETX, DLE. The DQS itself will supply leading SYN bytes and trailing block check and pad bytes. The interface examines only the last byte of each text block received and so is unaware of the presence of headings or transparent text. The selection and interpretation of these features is the user's responsibility.

Line control functions, such as the alternating affirmative responses (ACK0 and ACK1), are automatically interspersed with text blocks as required by the line discipline. The interface handles the initial line bid and the EOT reset at the end of a transmission. A 3-second time-out is also respected. The interface will send TTD's and respond WACK's if its buffers are not serviced fast enough. When receiving, expiration of the time-out will cause the interface to abort the active message by sending EOT. When transmitting, the failure to send a block successfully after seven tries will cause the interface to terminate the active message prematurely. Such aborts cannot be appealed.

Reads on the DQS will return bytes from a single text block. If one read does not exhaust a text block, successive reads will return additional bytes from the same block. A returned count of zero indicates the end of a message. Until the remote station bids for the line, all reads will return zero bytes. The error bit will never be set by the interface itself. must be read to the end of a message before it will accept writes.

Writes to the DQS must consist of a single, entire text block. A write that specifies a count of zero bytes defines the end of a message. The count returned by a write call must be checked. A count of zero for the first write of a new message indicates that it was not possible to acquire the line. Otherwise, the DQS should return exactly the count specified in the write call. However, the error bit is set when a line error requires that the
message be aborted. Notification of the error is not punctual, because data blocks are buffered for transmission. A write of zero bytes must be issued, or an error must occur, before the DQS will accept reads.

An `open(2)` will fail if the DQS is already open or not ready. The DQS should be opened to allow both reading and writing.

The DQS interface steals a number of buffers from UNIX (currently two) for the duration of each message. This number is specified at system generation time and may be tuned to influence overall system throughput.

SEE ALSO

*DQS11-A/B PDP-11 Communications Controller Option Description*, Digital Equipment Corporation.
NAME
   du — DU-11 synchronous line interface

DESCRIPTION
   The files du0, du1, etc., represent interfaces to synchronous modems such
   as the Bell System 200-series synchronous DATA-PHONE® sets. Read and
   write calls to du? are unlimited, but work best when restricted to less than
   512 bytes. Each write call is sent as a single record. Seven bits from each
   byte are written, along with an eighth, odd-parity, bit. The "sync" charac-
   ters must be supplied by the user. Each read call returns the characters
   read from a single record. Seven bits are returned unaltered; the eighth bit
   is set if the byte was not received in odd parity. An error is returned if
   data-set ready is not present.

FILES
   /dev/du?

SEE ALSO
   dn(4).
NAME
dz, dzk, dh — DZ-11, DZ-11/KMC-11, DH-11 asynchronous multiplexers

DESCRIPTION
Each line attached to a DH-11 or DZ-11 communications multiplexer behaves as described in tty(4). Input and output for each line may independently be set to run at any of 16 speeds; see tty(4) for the encoding. (For DZ-11 lines, output speed is always the same as input speed. The 200 speed and the two externally clocked speeds (exta, extb) are missing on the DZ-11.) The behavior of dzk lines is indistinguishable from that of dz lines, except that on the dzk backspace delays are implemented using fill characters (rubouts) instead of timed delays.

Note that the DH-11 is considered obsolete and is not supported on the VAX-11/780.

FILES
/dev/tty*

SEE ALSO
kmc(4), tty(4).
NAME
err — error-logging interface

DESCRIPTION
Minor device 0 of the err driver is the interface between a process and the system's error-record collection routines. The driver may be opened only for reading by a single process with super-user permissions. Each read causes an entire error record to be retrieved; the record is truncated if the read request is for less than the record's length.

FILES
/dev/error special file

SEE ALSO
errdemon(1M).
NAME
hp — RP04/RP05/RP06 moving-head disk

DESCRIPTION
The files `rp0`, ..., `rp7` refer to sections of the RP04/RP05/RP06 disk drive 0. The files `rp10`, ..., `rp17` refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

<table>
<thead>
<tr>
<th>Section</th>
<th>RP04/05</th>
<th>Length</th>
<th>RP06</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>18392</td>
<td>0</td>
<td>18392</td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>153406</td>
<td>1</td>
<td>322278</td>
</tr>
<tr>
<td>2</td>
<td>201</td>
<td>87780</td>
<td>2</td>
<td>256652</td>
</tr>
<tr>
<td>3</td>
<td>358</td>
<td>22154</td>
<td>3</td>
<td>191026</td>
</tr>
<tr>
<td>4</td>
<td>—</td>
<td>—</td>
<td>4</td>
<td>515</td>
</tr>
<tr>
<td>5</td>
<td>—</td>
<td>—</td>
<td>5</td>
<td>125400</td>
</tr>
<tr>
<td>6</td>
<td>—</td>
<td>—</td>
<td>6</td>
<td>59774</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>171798</td>
<td>7</td>
<td>340670</td>
</tr>
</tbody>
</table>

The start address is a cylinder address, with each cylinder containing 418 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

The `rp` files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with `rrp` and end with a number which selects the same disk section as the corresponding `rp` file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise `lseek` calls should specify a multiple of 512 bytes.

FILES
/dev/rp*, /dev/rrp*

SEE ALSO
rp(4).
NAME
hs — RH11/RJS03-RJS04 fixed-head disk file

DESCRIPTION
The files hs0, ..., hs7 refer to RJS03 disk drives 0 through 7. The files hs8, ..., hs15 refer to RJS04 disk drives 0 through 7. The RJS03 drives are each 1024 blocks long and the RJS04 drives are 2048 blocks long.

The hs files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw HS files begin with rhs. The same minor device considerations hold for the raw interface as for the normal interface.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise lseek calls should specify a multiple of 512 bytes.

FILES
/dev/hs*, /dev/rhs*
NAME
ht — TU16 magnetic tape interface

DESCRIPTION
The files mt0, ..., mt15 refer to the Digital Equipment Corporation TU16 magnetic tape control and transports. The files mt0, ..., mt7 are 800bpi, and the files mt8, ..., mt15 are 1600bpi. The files mt0, ..., mt3, mt8, ..., mt11 are designated normal-rewind on close, and the files mt4, ..., mt7, mt12, ..., mt15 are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing mt files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The mt files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the “raw” interface is appropriate. The associated files are named rmt0, ..., rmt15. Each read or write call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

FILES
/dev/mt*, /dev/rmt*

BUGS
If any non-data error is encountered, it refuses to do anything more until closed. The driver is limited to four transports.
NAME
kl — KL-11 or DL-11 asynchronous interface

DESCRIPTION
The discussion of typewriter I/O given in tty(4) applies to these devices.
Since they run at a constant speed, attempts to change the speed are ignored.
The on-line console typewriter is normally interfaced using a KL-11 or DL-11.

FILES
/dev/console

SEE ALSO
tty(4), init(8).

BUGS
Modem control for the DL-11E is not implemented.
NAME
kmc - KMC11 microprocessor

DESCRIPTION
The files kmc? are used to manipulate the KMC11-A or -B microprocessors. The device handler provides the basic mechanism needed to load, run, and debug programs on the microprocessor.

The open is exclusive; at most one open at a time. The first open determines whether the microprocessor is a KMC11-A or -B.

Addresses 0—2047 (0—8195) reference the 1024 (4096) words of instructions in the control memory of the KMC11-A (-B). This portion is word oriented, that is, the address and byte count must be even.

Addresses 2048-3071 (8196—12211) reference the 1024 (4096) bytes of data in the data memory of the KMC11-A (-B). The data portion may be read or written with no restrictions on addressing.

The stty function is used to provide access to the basic microprocessor capabilities.

```c
stty(kmcfd, arg)
struct {
    int code;
    int *csr;
    int value;
} *arg;
```

The pointer csr contains the address of a 4 word buffer for the UNIBUS Control and Status Registers associated with the microprocessor. The value of code determines the function:

1. single step and return CSRs in csr.
2. maintenance step: execute value and then return CSRs.
3. return CSRs.
4. stop: clear the run bit.
5. reset: set then clear the master clear bit.
6. run: set the run bit and set the software state to value and running.
7. line unit maintenance: set the line unit bits from value.

FILES
/dev/kmc?

SEE ALSO
kas(1), kun(1), dh(4).
NAME
lp — line printer

DESCRIPTION
    Lp provides the interface to any of the standard Digital Equipment Corporation line printers. When it is opened or closed, a suitable number of page ejects is generated. Bytes written are printed.

    An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lower case letters are turned into upper case and certain characters are escaped according to the following table:

    \{   \+
    \}   \+
    \-   \-
    \|   \-

    The driver correctly interprets carriage returns, backspaces, tabs, and form-feeds. A new-line that extends over the end of a page is turned into a form-feed. The default line length is 80 characters, indent is 4 characters and lines per page is 66. Lines longer than the line length minus the indent (i.e. 76 characters, using the above defaults) are truncated.

FILES
    /dev/lp

SEE ALSO
    lpr(1).
NAME
mem, kmem - core memory

DESCRIPTION
Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system. Byte addresses in mem are interpreted as memory addresses. References to non-existent locations cause errors to be returned. Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present. The file kmem is the same as mem except that kernel virtual memory rather than physical memory is accessed. On the PDP-11, the I/O page begins at location 0160000 of kmem and per-process data for the current process begins at 0140000.

FILES
/dev/mem, /dev/kmem

BUGS
On the PDP-11, memory files are accessed one byte at a time, an inappropriate method for some device registers.
NAME
null — the null file

DESCRIPTION
Data written on a null special file is discarded.
Reads from a null special file always return 0 bytes.

FILES
/dev/null
NAME
pcl — parallel communications link interface

DESCRIPTION
Pcl provides the interface to the Digital Equipment Corporation PCL-IIB network bus. This bus can be used to interconnect up to 16 CPU’s, providing relatively fast communication without individual point-to-point connections.

The interface permits simultaneous bi-directional communication between any machines on the bus. Additionally, each such path is further subdivided into 8 independent channels. A control interface is also provided to reduce the line monitoring overhead for a daemon process.

FILES
/dev/pcl[a-z][0-7] normal machine and subchannel interface.
/dev/pclc control interface.
NAME

prf — operating system profiler

DESCRIPTION

The file prf provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file prf is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

config(1M), profiler(1M).
NAME
rf — RF11/RS11 fixed-head disk file

DESCRIPTION
This file refers to the concatenation of all RS-11 disks.

Each disk contains 1024 256-word blocks. The length of the combined RF file is $1024 \times (\text{minor} + 1)$ blocks. That is minor device zero is taken to be 1024 blocks long; minor device one is 2048, etc.

The rf0 file accesses the disk via the system’s normal buffering mechanism and may be read and written without regard to physical disk records. There is also a “raw” interface which provides for direct transmission between the disk and the user’s read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The name of the raw RF file is rrf0. The same minor device considerations hold for the raw interface as for the normal interface.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise seek calls should specify a multiple of 512 bytes.

FILES
/dev/rf0, /dev/rrf0

BUGS
The 512-byte restrictions on the raw device are not physically necessary, but are still imposed.
NAME
rk — RK-11/RK03 or RK05 disk

DESCRIPTION
Rk? refers to an entire RK03 disk as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 4871.

The rk files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RK files begin with rrk and end with a number which selects the same disk as the corresponding rk file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise seek calls should specify a multiple of 512 bytes.

FILES
/dev/rk*, /dev/rrk*
NAME

rl - RL-11/RL01 disk

DESCRIPTION

rl0, ..., rl3 refer to an entire RL01 disk drive as a single sequentially-addressed file. Its 256-word blocks are numbered 0 to 10239.

The rl files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O call and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RL files begin with rrl and end with a number which selects the same disk as the corresponding rl file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise lseek calls should specify a multiple of 512 bytes.

FILES

/dev/rl*, /dev/rrl*
NAME
rp — RP-11/RP03 moving-head disk

DESCRIPTION
The files rp0, ..., rp7 refer to sections of the RP03 disk drive 0. The files rp10, ..., rp17 refer to drive 1, etc. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

<table>
<thead>
<tr>
<th>section</th>
<th>start</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>71200</td>
</tr>
<tr>
<td>2</td>
<td>203</td>
<td>40600</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>81200</td>
</tr>
</tbody>
</table>

The start address is a cylinder address, with each cylinder containing 200 blocks. It is extremely unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter.

The rp files access the disk via the system’s normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw RP files begin with rrp and end with a number which selects the same disk section as the corresponding rp file.

In raw I/O the buffer must begin on a word boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise lseek calls should specify a multiple of 512 bytes.

FILES
/dev/rp*, /dev/rrp*

SEE ALSO
hp(4).
NAME
st — synchronous terminal interface

DESCRIPTION
The synchronous terminal interface is a pseudo-device driver that enables a
UNIX system to communicate with a TELETYPE® Model 40/4 ASCII syn­
chronous terminal. The driver utilizes the Virtual Protocol Machine (VPM)
to perform the end-to-end protocol and transmission assurance for the syn­
chronous line.

The user must be familiar with the operation of the Model 40/4 terminal.
Screen management functions are completely controlled by the user pro­
cess; when formatting a screen, the user must supply everything from the
initial STX (Start-of-Text) character to the ETX (End-of-Text) character.

By convention, /dev/st0 is the synchronous terminal control channel, while
other /dev/st? files represent user terminal channels. Communication with
the control channel is handled by the stcntl command (see st(1M)).

A user process will sleep when trying to open a channel, until a terminal
requests service. At that time, a channel will be assigned to that terminal,
and it will remain allocated until the user process closes the terminal.

In addition to the synchronous terminal equipment, a KMC11-B micropro­
cessor, and a DMC11-DA synchronous line unit are required.

FILES
/etc/stproto synchronous terminal prototype script
/dev/kmc? KMC11-B microprocessor
/dev/vpm? virtual protocol machine
/dev/st0 synchronous terminal control channel
/dev/st? synchronous terminal user channels

SEE ALSO
st(1M), kmc(4), trace(4), vpm(4).
NAME
tm — TMII/TUIO magnetic tape interface

DESCRIPTION
The files mt0, ..., mt7 refer to the Digital Equipment Corporation TMII/TUIO magnetic tape control and transports at 800bpi. The files mt0, ..., mt3 are designated normal-rewind on close, and the files mt4, ..., mt7 are no-rewind on close. When opened for reading or writing, the tape is assumed to be positioned as desired. When a file is closed, a double end-of-file (double tape mark) is written if the file was opened for writing. If the file was normal-rewind, the tape is rewound. If it is no-rewind and the file was open for writing, the tape is positioned before the second EOF just written. If the file was no-rewind and opened read-only, the tape is positioned after the EOF following the data just read. Once opened, reading is restricted to between the position when opened and the next EOF or the last write. The EOF is returned as a zero-length read. By judiciously choosing mt files, it is possible to read and write multi-file tapes.

A standard tape consists of several 512 byte records terminated by an EOF. To the extent possible, the system makes it possible, if inefficient, to treat the tape like any other file. Seeks have their usual meaning and it is possible to read or write a byte at a time (although very inadvisable).

The mt files discussed above are useful when it is desired to access the tape in a way compatible with ordinary files. When foreign tapes are to be dealt with, and especially when long records are to be read or written, the "raw" interface is appropriate. The associated files are named rmt0, ..., rmt7. Each read or write call reads or writes the next record on the tape. In the write case the record has the same length as the buffer given. During a read, the record size is passed back as the number of bytes read, up to the buffer size specified. In raw tape I/O, the buffer must begin on a word boundary and the count must be even. Seeks are ignored. An EOF is returned as a zero-length read, with the tape positioned after the EOF, so that the next read will return the next record.

FILES
/dev/mt?, /dev/rmt?

BUGS
If any non-data error is encountered, it refuses to do anything more until closed. The driver is limited to four transports.
NAME
trace — event-tracing driver

DESCRIPTION
Trace is a special file that allows UNIX kernel drivers to transfer event records to a user program, so that the activity of the driver may be monitored for debugging purposes.

An event record is generated from within a kernel driver by executing the following function:

```c
trsave(dev, chno, buf, cnt)
char dev, chno, *buf, cnt;
```

Dev is the minor device number of the trace driver; chno is an integer between 1 and 16, inclusive, identifying the data stream to which the record belongs; buf is a buffer containing the bytes that make up a single event record; and cnt is the number of bytes in buf. Calls to trsave will result in data being saved in a clist buffer, provided that some user program has opened the trace minor device number dev and has activated channel chno. Event records prefaced by chno and cnt are stored in a clist queue until a system-defined maximum (TRQMAX) is reached; event records are discarded while the queue is full. The clist queue is emptied by a user program reading the trace driver. The trace driver returns an integral number of event records; the read count must, therefore, be at least equal to the size of a record plus two, to allow for the chno and cnt bytes added to the event record by the trsave routine.

The trace driver supports open, close, read, and ioctl system calls. To activate a channel, ioctl is used as follows:

```c
#include <ioctl.h>
ioctl(fildes, VPMTRCO, chno)
```

SEE ALSO
vpmstart(1C), vpm(4).
NAME
tty — general terminal interface

DESCRIPTION
This section describes both a particular special file and the general nature of
the terminal interface.

The file /dev/tty is, in each process, a synonym for the control terminal
associated with the process group of that process, if any. It is useful for
programs or shell sequences that wish to be sure of writing messages on the
terminal no matter how output has been redirected. It can also be used for
programs that demand the name of a file for output, when typed output is
desired and it is tiresome to find out what terminal is currently in use.

As for terminals in general: all of the asynchronous communications ports
use the same general interface, no matter what hardware is involved. The
remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until
a connection is established. In practice, users' programs seldom open these
files; they are opened by `getty(8)` and become a user's standard input, output,
and error files. The very first terminal file opened by the process
group leader of a terminal file not already associated with a process group
becomes the *control terminal* for that process group. The control terminal
plays a special role in handling quit and interrupt signals, as discussed
below. The control terminal is inherited by a child process during a
`fork(2)`.

A process can break this association by changing its process group
using `setpgrp(2)`.

A terminal associated with one of these files ordinarily operates in full-
duplex mode. Characters may be typed at any time, even while output is
occurring, and are only lost when the system's character input buffers
become completely full, which is rare, or when the user has accumulated
the maximum allowed number of input characters that have not yet been
read by some program. Currently, this limit is 256 characters. When the
input limit is reached, all the saved characters are thrown away without
notice.

Normally, terminal input is processed in units of lines. A line is delimited
by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or
an end-of-line character. This means that a program attempting to read will
be suspended until an entire line has been typed. Also, no matter how
many characters are requested in the read call, at most one line will be
returned. It is not, however, necessary to read a whole line at once; any
number of characters may be requested in a read, even one, without losing
information.

During input, erase and kill processing is normally done. By default, the
character # erases the last character typed, except that it will not erase
beyond the beginning of the line. By default, the character @ kills (deletes)
the entire input line, and optionally outputs a new-line character.
Both these characters operate on a key-stroke basis, independently of any
backspacing or tabbing that may have been done. Both the erase and kill
characters may be entered literally by preceding them with the escape
character (\). In this case the escape character is not read. The erase and
kill characters may be changed.

Certain characters have special functions on input. These functions and
their default character values are summarized as follows:
INTR (Rubout or ASCII DEL) generates an interrupt signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see signal(2).

QUIT (Control-| or ASCII FS) generates a quit signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called core) will be created in the current working directory.

ERASE (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF) is the normal line delimiter. It can not be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

STOP (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a hangup signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several ioctl(2) system calls apply to terminal files. The primary calls use the following structure, defined in <termio.h>:
typedef NCC

struct termio {
    unsigned short c_iflag; /* input modes */
    unsigned short c_oflag; /* output modes */
    unsigned short c_cflag; /* control modes */
    unsigned short c_lflag; /* local modes */
    char c_line; /* line discipline */
    unsigned char c_cc[NCC]; /* control chars */
};

The special control characters are defined by the array c_cc. The relative positions and initial values for each function are as follows:

0 INTR DEL
1 QUIT FS
2 ERASE \$
3 KILL @
4 EOF EOT
5 EOL NUL
6 reserved
7 reserved

The c_iflag field describes the basic terminal input control:

IGNBRK 0000001 Ignore break condition.
BRKINT 0000002 Signal interrupt on break.
IGNPAR 0000004 Ignore characters with parity errors.
PARMRK 0000010 Mark parity errors.
INPCK 0000020 Enable input parity check.
ISTRIP 0000040 Strip character.
INLCR 0000100 Map NL to CR on input.
IGNCR 0000200 Ignore CR.
ICRNL 0000400 Map CR to NL on input.
IUCLC 0001000 Map upper-case to lower-case on input.
IXON 0002000 Enable start/stop output control.
IXANY 0004000 Enable any character to restart output.
IXOFF 0010000 Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.
If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all bits clear.

The c_oflag field specifies the system treatment of output:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPOST</td>
<td>Postprocess output.</td>
</tr>
<tr>
<td>OLCUC</td>
<td>Map lower case to upper on output.</td>
</tr>
<tr>
<td>ONLCR</td>
<td>Map NL to CR-NL on output.</td>
</tr>
<tr>
<td>OCRNL</td>
<td>Map CR to NL on output.</td>
</tr>
<tr>
<td>ONOCR</td>
<td>No CR output at column 0.</td>
</tr>
<tr>
<td>ONLRET</td>
<td>NL performs CR function.</td>
</tr>
<tr>
<td>OFILL</td>
<td>Use fill characters for delay.</td>
</tr>
<tr>
<td>OFDEL</td>
<td>Fill is DEL, else NUL.</td>
</tr>
<tr>
<td>NLDLY</td>
<td>Select new-line delays:</td>
</tr>
<tr>
<td>NL0</td>
<td>0</td>
</tr>
<tr>
<td>NL1</td>
<td>0000400</td>
</tr>
<tr>
<td>CRDLY</td>
<td>Select carriage-return delays:</td>
</tr>
<tr>
<td>CR0</td>
<td>0</td>
</tr>
<tr>
<td>CR1</td>
<td>0001000</td>
</tr>
<tr>
<td>CR2</td>
<td>0002000</td>
</tr>
<tr>
<td>CR3</td>
<td>0003000</td>
</tr>
<tr>
<td>TABDLY</td>
<td>Select horizontal-tab delays:</td>
</tr>
<tr>
<td>TAB0</td>
<td>0</td>
</tr>
<tr>
<td>TAB1</td>
<td>0004000</td>
</tr>
<tr>
<td>TAB2</td>
<td>0010000</td>
</tr>
<tr>
<td>TAB3</td>
<td>0014000</td>
</tr>
<tr>
<td>BSDLY</td>
<td>Expand tabs to spaces.</td>
</tr>
<tr>
<td>BS0</td>
<td>0</td>
</tr>
<tr>
<td>BS1</td>
<td>0020000</td>
</tr>
<tr>
<td>VTDLY</td>
<td>Select vertical-tab delays:</td>
</tr>
<tr>
<td>VT0</td>
<td>0</td>
</tr>
<tr>
<td>VT1</td>
<td>0040000</td>
</tr>
<tr>
<td>FFDLY</td>
<td>Select form-feed delays:</td>
</tr>
<tr>
<td>FF0</td>
<td>0</td>
</tr>
<tr>
<td>FF1</td>
<td>0100000</td>
</tr>
</tbody>
</table>

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.
The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The _c_flag_ field describes the hardware control of the terminal:

<table>
<thead>
<tr>
<th>Bit Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBAUD</td>
<td>000017</td>
<td>Baud rate:</td>
</tr>
<tr>
<td>B0</td>
<td>0</td>
<td>Hang up</td>
</tr>
<tr>
<td>B50</td>
<td>000001</td>
<td>50 baud</td>
</tr>
<tr>
<td>B75</td>
<td>000002</td>
<td>75 baud</td>
</tr>
<tr>
<td>B110</td>
<td>000003</td>
<td>110 baud</td>
</tr>
<tr>
<td>B134</td>
<td>000004</td>
<td>134.5 baud</td>
</tr>
<tr>
<td>B150</td>
<td>000005</td>
<td>150 baud</td>
</tr>
<tr>
<td>B200</td>
<td>000006</td>
<td>200 baud</td>
</tr>
<tr>
<td>B300</td>
<td>000007</td>
<td>300 baud</td>
</tr>
<tr>
<td>B600</td>
<td>000010</td>
<td>600 baud</td>
</tr>
<tr>
<td>B1200</td>
<td>000011</td>
<td>1200 baud</td>
</tr>
<tr>
<td>B1800</td>
<td>000012</td>
<td>1800 baud</td>
</tr>
<tr>
<td>B2400</td>
<td>000013</td>
<td>2400 baud</td>
</tr>
<tr>
<td>B4800</td>
<td>000014</td>
<td>4800 baud</td>
</tr>
<tr>
<td>B9600</td>
<td>000015</td>
<td>9600 baud</td>
</tr>
<tr>
<td>EXTA</td>
<td>000016</td>
<td>External A</td>
</tr>
<tr>
<td>EXTB</td>
<td>000017</td>
<td>External B</td>
</tr>
<tr>
<td>CSIZE</td>
<td>000060</td>
<td>Character size:</td>
</tr>
<tr>
<td>CS5</td>
<td>0</td>
<td>5 bits</td>
</tr>
<tr>
<td>CS6</td>
<td>000020</td>
<td>6 bits</td>
</tr>
<tr>
<td>CS7</td>
<td>000040</td>
<td>7 bits</td>
</tr>
<tr>
<td>CS8</td>
<td>000060</td>
<td>8 bits</td>
</tr>
<tr>
<td>CSTOPB</td>
<td>000100</td>
<td>Send two stop bits, else one.</td>
</tr>
<tr>
<td>CREAD</td>
<td>000200</td>
<td>Enable receiver.</td>
</tr>
<tr>
<td>PARENB</td>
<td>000400</td>
<td>Parity enable.</td>
</tr>
<tr>
<td>PARODD</td>
<td>001000</td>
<td>Odd parity, else even.</td>
</tr>
<tr>
<td>HUPCL</td>
<td>002000</td>
<td>Hang up on last close.</td>
</tr>
<tr>
<td>LOCAL</td>
<td>004000</td>
<td>Local line, else dial-up.</td>
</tr>
</tbody>
</table>

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal
will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The cjflag field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISIG</td>
<td>0000001</td>
<td>Enable signals.</td>
</tr>
<tr>
<td>ICANON</td>
<td>0000002</td>
<td>Canonical input (erase and kill processing).</td>
</tr>
<tr>
<td>XCASE</td>
<td>0000004</td>
<td>Canonical upper/lower presentation.</td>
</tr>
<tr>
<td>ECHO</td>
<td>0000010</td>
<td>Enable echo.</td>
</tr>
<tr>
<td>ECHOE</td>
<td>0000020</td>
<td>Echo erase character as BS-SP-BS.</td>
</tr>
<tr>
<td>ECHOK</td>
<td>0000040</td>
<td>Echo NL after kill character.</td>
</tr>
<tr>
<td>ECHONL</td>
<td>0000100</td>
<td>Echo NL.</td>
</tr>
<tr>
<td>NOFLSH</td>
<td>0000200</td>
<td>Disable flush after interrupt or quit.</td>
</tr>
</tbody>
</table>

If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g. 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:
for: use:
\n| ![ ]
| ![ ]
| ![ ]
| ![ ]

For example, A is input as \a, \n as \n, and \N as \n.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary `ioctl(2)` system calls have the form:
```
ioctcl (fildes, command, arg)
struct termio *arg;
```
The commands using this form are:

- **TCGETA**: Get the parameters associated with the terminal and store in the `termio` structure referenced by `arg`.
- **TCSETA**: Set the parameters associated with the terminal from the structure referenced by `arg`. The change is immediate.
- **TCSETAW**: Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
- **TCSETAF**: Wait for the output to drain, then flush the input queue and set the new parameters.

Additional `ioctl(2)` calls have the form:
```
ioctcl (fildes, command, arg)
int arg;
```
The commands using this form are:

- **TCSBRK**: Wait for the output to drain. If `arg` is 0, then send a break (zero bits for 0.25 seconds).
- **TCXONC**: Start/stop control. If `arg` is 0, suspend output; if 1, restart suspended output.
- **TCFLSH**: If `arg` is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.
FILES
    /dev/tty
    /dev/tty*
    /dev/console

SEE ALSO
    stty(1), ioctl(2).
NAME
  vp — Versatec printer

DESCRIPTION
  Vp provides the interface to the Versatec electro-static line printer. Both
  printing and plotting capabilities are implemented.

FILES
  /dev/vp

SEE ALSO
  vpr(1), lp(4).
NAME

vpm — The Virtual Protocol Machine

DESCRIPTION

This entry describes a particular kind of special file and gives an introduction to the Virtual Protocol Machine (VPM).

The VPM is a software construct for implementing link protocols on the KMCII in a high-level language. This is accomplished by a compiler that runs on UNIX and that translates a high-level language description of a protocol into an intermediate language that is interpreted by an interpreter running in the KMCII.

The VPM driver is functionally split into two parts: a top VPM device and a bottom VPM device. The top device may be modified or replaced to suit particular applications; the bottom device interfaces with the VPM interpreter using the KMC driver. When using the mknod command to make a directory entry and corresponding i-node for a VPM special file, the minor device number identifies the top, bottom, and physical KMC devices to be used for this special file. The two most significant bits of the minor device number denote the physical KMC device; the next two bits denote the VPM bottom device; the four least significant bits denote the VPM top device. For example, if top device 1 is to be used with bottom device 2, which in turn is to be used with KMC device 3, the minor device number would be 0341 (octal).

UNIX user processes transfer data to or from a remote terminal or computer system through VPM using normal open, read, write, and close operations. Flow control and error recovery are provided by the protocol description residing in the KMCII.

The VPM software consists of six components:

1. vpmc(1C): compiler for the protocol description language; it runs on UNIX.
2. VPM interpreter: a KMCII program that controls the overall operation of the KMCII and interprets the protocol script.
3. vpm.c: a UNIX driver that provides the interface to the VPM.
4. vpmstart(1C): a UNIX command that copies a load module into the KMCII and starts it.
5. vpmssnap(1C): a UNIX command that prints a time-stamped event trace while the protocol is running.
6. vpmtrace(1C): a UNIX command that prints an event trace for debugging purposes while the protocol is running.

The VPM open for reading-and-writing is exclusive; opens for reading-only or writing-only are not. The VPM open checks that the correct interpreter is running in the KMCII, then sends a RUN command to the interpreter (causing it to start interpreting the protocol script), and supplies a 512-byte receive buffer to the interpreter.

The VPM read returns either the number of bytes requested or the number remaining in the current receive buffer, whichever is less. Bytes remaining in a receive buffer are used to satisfy subsequent reads. The VPM write copies the user data into 512-byte system buffers and passes them to the VPM interpreter in the KMCII for transmission.

The VPM close arranges for the return of system buffers and for a general cleanup when the last transmit buffer has been returned by the interpreter.

The user command vpmtrace(1C) reads the trace driver and prints event records. While this command is executing, the VPM driver will generate a
number of event records, allowing the activity of the VPM driver and protocol script to be monitored for debugging purposes. The system functions `vpmopen`, `vpmread`, `vpmwrite`, and `vpmclose` generate event records (identified respectively by o, r, w, and e). Calls to the `vpmc(1C)` primitive `trace(arg1, arg2)` cause the VPM interpreter to pass `arg1` and `arg2` along with the current value of the script location counter to the VPM driver, which generates an event record identified by a `T`. Each event record is structured as follows:

```c
struct event {
    short e_seqn;  /* sequence number */
    char e_type;  /* record identifier */
    char e_dev;  /* minor device number */
    short e_short1;  /* data */
    short e_short2;  /* data */
}
```

When the script terminates for any reason, the driver is notified and generates an event record identified by an `E`. This record also contains the minor device number, the script location counter, and a termination code defined as follows:

0  Normal termination; the interpreter received a `halt` command from the driver.
1  Undefined virtual-machine operation code.
2  Script program counter out of bounds.
3  Interpreter stack overflow or underflow.
4  Jump address not even.
5  UNIBUS error.
6  Transmit buffer has an odd address; the driver tried to give the interpreter too many transmit buffers; or a `get` or `rtnxbuf` was executed while no transmit buffer was open, i.e., no `getxbuf` was executed prior to the `get` or `rtnxbuf`.
7  Receive buffer has an odd address; the driver tried to give the interpreter too many receive buffers; or a `put` or `rtnrbuf` was executed while no receive buffer was open, i.e., no `getrbuf` was executed prior to the `get` or `rtnxbuf`.
8  The script executed an `exit`.
9  A `crc16` was executed without a preceding `crcloc` execution.
10 Interpreter detected loss of modem-ready signal.
11 Transmit-buffer sequence-number error.
12 Command error; an invalid command or an improper sequence of commands was received from the driver.
13 Not used.
14 Invalid transmit state.
15 Invalid receive state.
16 Not used.
17 `Xmtctl` or `setctl` attempted while transmitter was still busy.
18 Not used.
19 Same as error code 6.
20 Same as error code 7.
21 Script too large.
22 Used for debugging the interpreter.
23 The driver’s OK-check has timed out.

**SEE ALSO**

`vpmc(1C)`, `vpmstart(1C)`, `trace(4)`.
NAME
intro — introduction to file formats

DESCRIPTION
This section outlines the formats of various files. The C struct declarations for the file formats are given where applicable. Usually, these structures can be found in the directories /usr/include or /usr/include/sys.
NAME

a.out — assembler and link editor output

DESCRIPTION

A.out is the output file of the assembler as and the link editor ld. Both programs will make a.out executable if there were no errors in assembling or linking, and no unresolved external references.

This file has four sections: a header, the program text and data segments, relocation information, and a symbol table (in that order). The last two sections may be missing if the program was linked with the -s option of ld(1) or if the symbol table and relocation bits were removed by strip(1). Also note that if there were no unresolved external references after linking, the relocation information will be removed.

The sizes of each segment (contained in the header, discussed below) are in bytes and are even. The size of the header is not included in any of the other sizes.

When an a.out file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment begins at location 0 in the core image; the header is not loaded. If the magic number (the first field in the header) is 407 (octal), it indicates that the text segment is not to be write-protected or shared, so the data segment will be contiguous with the text segment. If the magic number is 410 (octal), the data segment begins at the first 0 mod 8K byte boundary on the PDP-11, or the first 0 mod 512 byte boundary on the VAX-11/780 following the text segment, and the text segment is not writable by the program; if other processes are executing the same a.out file, they will share a single text segment. If the magic number is 411 (octal) (PDP-11 only), the text segment is again pure (write-protected and shared) and, moreover, the instruction and data spaces are separated; the text and data segment both begin at location 0. See the PDP-11/70 Processor Handbook for restrictions that apply to this situation.

The stack will occupy the highest possible locations in the core image: from 177776 (octal) on the PDP-11 or 80000000 (hexadecimal) on the VAX-11/780, and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by the brk(2) system call.

The start of the text segment in the a.out file is hsize; the start of the data segment is hsize+S, (the size of the text), where hsize is 20 (octal) on the PDP-11 and 20 (hexadecimal) on the VAX-11/780.

The value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information (discussed below) for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.
Header—PDP-11

The format of the a.out header for the PDP-11 is as follows:

```c
struct exec {
    short a_magic;    /* magic number */
    unsigned a_text; /* size of text segment */
    unsigned a_data; /* size of data segment */
    unsigned a_bss;  /* size of bss segment */
    unsigned a_syms; /* size of symbol table */
    unsigned a_entry; /* entry point of program */
    unsigned a_stamp; /* version stamp */
    unsigned a_flag; /* set if relocation info stripped */
};
```

Header—VAX-11/780

The format of the header on the VAX-11/780 is as follows:

```c
struct exec {
    short a_magic;    /* magic number */
    short a_stamp;   /* version stamp */
    unsigned a_text; /* size of text segment */
    unsigned a_data; /* size of data segment */
    unsigned a_bss;  /* size of bss segment */
    unsigned a_syms; /* size of symbol table */
    unsigned a_entry; /* entry point of program */
    unsigned a_trsize; /* size of text relocation info */
    unsigned a_drsize; /* size of data relocation info */
};
```

Relocation—PDP-11

If relocation information is present, it amounts to two bytes per relocatable datum. There is no relocation information if the "suppress relocation" flag (a...flag) in the header is on.

The format of the relocation data is:

```c
struct r_info {
    int r_symbolnum:11,
    r_segment:3,
    r_pcrel:1;
};
```

The r_pcrel field indicates, if on, that the reference is relative to the program counter (pc) register (e.g., clr x); if off, that the reference is to the actual symbol (e.g., clr *$x).

The r_segment field indicates the segment referred to by the text or data word associated with the relocation word:

- 00 indicates the reference is absolute;
- 02 indicates the reference is to the text segment;
- 04 indicates the reference is to initialized data;
- 06 indicates the reference is to bss (uninitialized data);
- 10 indicates the reference is to an undefined external symbol.

The field r_symbolnum contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.
Relocation—VAX-11/780

If relocation information is present, it amounts to eight bytes per relocatable datum. There are no relocation bits if \( a_{\text{rsize}} + a_{\text{drsize}} = 0 \). The format of the relocation information is:

```c
struct r_info {
    long r_address;
    int r_symbolnum:24,
        r_pcrel:1,
        r_length:2,
        r_extern:1,
        r_offset:1,
        r_pad:3;
};
```

The \( r_{\text{address}} \) field gives the location of the relocatable reference relative to the segment in which it is defined. The \( r_{\text{symbolnum}} \) field contains a symbol number in the case of an external; otherwise it contains a segment number (expressed in the same manner as the VAX-11/780 symbol types above). \( R_{\text{pcrel}} \) has the same meaning as on the PDP-11. \( R_{\text{length}} \) indicates the length of the relocatable reference:

- 0 byte
- 1 word
- 2 long

The start of the relocation information (on the PDP-11 and the VAX-11/780) is:

\[ hsize + a_{\text{text}} + a_{\text{data}} \]

Symbol Table—PDP-11

The symbol table on the PDP-11 consists of entries of the form:

```c
struct nlist {
    char n_name[8];
    int n_type;
    unsigned n_value;
};
```

The \( n_{\text{name}} \) field contains the ASCII name of the symbol, null-padded. The \( n_{\text{type}} \) field indicates the type of the symbol; the following values are possible:

- 00 undefined symbol
- 01 absolute symbol
- 02 text segment symbol
- 03 data segment symbol
- 04 bss segment symbol
- 37 file name symbol (produced by \( ld \))
- 40 undefined external symbol
- 41 absolute external symbol
- 42 text segment external symbol
- 43 data segment external symbol
- 44 bss segment external symbol

The start of the symbol table on the PDP-11 is:

\[ hsize + 2(a_{\text{text}} + a_{\text{data}}) \]

if relocation information is present, and

\[ hsize + a_{\text{text}} + a_{\text{data}} \]

if it is not.
Symbol Table—VAX-11/780

The symbol table on the VAX consists of entries of the form:

```c
struct nlist {
    char n_name[8];
    char n_type;
    char n_other;
    short n_desc;
    unsigned n_value;
};
```

The possible values for `n_type` are:

- 00 undefined symbol
- 02 absolute symbol
- 04 text segment symbol
- 06 data segment symbol
- 08 bss segment symbol
- 37 file name symbol (produced by `ld(1)`)  
- 40 undefined external symbol
- 42 absolute external symbol
- 44 text segment external symbol
- 46 data segment external symbol
- 48 bss segment external symbol

The start of the symbol table on the VAX is:

```
hsize + a_text + a_data + a_trsize + a_drsize
```

If a symbol's type (on either the PDP-11 or the VAX-11/780) is *undefined external* and the value field is non-zero, the symbol is interpreted by the link editor `ld(1)` as the name of a common region whose size is indicated by the value of the symbol.

**SEE ALSO**

`as(1), ld(1), nm(1), strip(1)`.
NAME
acct — per-process accounting file format

SYNOPSIS
#include <sys/acct.h>

DESCRIPTION
Files produced as a result of calling acct(2) have records in the form defined by <sys/acct.h>, whose contents are:

/* Accounting structures */

typedef ushort comp_t; /* "floating point" */
/* 13-bit fraction, 3-bit exponent */

struct acct
{
    char ac_flag; /* accounting flag */
    char ac_stat; /* exit status */
    ushort ac_uid; /* accounting user ID */
    ushort ac_gid; /* accounting group ID */
    dev_t ac_tty; /* control typewriter */
    time_t ac_btime; /* beginning time */
    comp_t ac_utime; /* accounting user time in clock ticks */
    comp_t ac_stime; /* accounting system time in clock ticks */
    comp_t ac_etime; /* accounting elapsed time in clock ticks */
    comp_t ac_mem; /* memory usage */
    comp_t ac_io; /* chars transferred */
    comp_t ac_rw; /* blocks read or written */
    char ac_comm[8]; /* command name */
};

extern struct acct acctbuf; /* acct type: 00 = acct */
extern struct inode *acctp; /* inode of accounting file */

#define AFORK 01 /* has executed fork, but no exec */
#define ASU 02 /* used super-user privileges */
#define ACCTF 0300 /* record type: 00 = acct */

In ac_flag, the AFORK flag is turned on by each fork(2) and turned off by an exec(2). The ac_comm field is inherited from the parent process and is reset by any exec. Each time the system charges the process with a clock tick, it also adds to ac_mem the current process size, computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of ac_mem/ac_stime can be viewed as an approximation to the mean process size, as modified by text-sharing.
The following structure represents the total accounting format used by the various accounting commands:

```c
/*
 * total accounting (for acct period), also for day
 */

struct tacct {
    uid_t    ta_uid;       /* userid */
    char     ta_name[8];   /* login name */
    float    ta_cpu[2];    /* cum. cpu time, p/np (mins) */
    float    ta_kcore[2];  /* cum. kcore-minutes, p/np */
    float    ta_con[2];    /* cum. conn. time, p/np, mins */
    float    ta_du;        /* cum. disk usage */
    long     ta_pc;        /* count of processes */
    unsigned short ta_sc;   /* count of login sessions */
    unsigned short ta_dc;   /* count of disk samples */
    unsigned short ta_fee;  /* fee for special services */
};
```

**SEE ALSO**
acct(1M), acctcom(1), acct(2).

**BUGS**
The `ac_mem` value for a short-lived command gives little information about the actual size of the command, because `ac_mem` may be incremented while a different command (e.g., the shell) is being executed by the process.
NAME
ar — archive file format

DESCRIPTION
The archive command \texttt{ar} is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor \texttt{ld(1)}.

A file produced by \texttt{ar} has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number is 0177545 (octal) (it was chosen to be unlikely to occur anywhere else). The header of each file is 26 bytes long:

```c
#define ARMAG 0177545
struct ar_hdr {
    char ar_name[14];
    long ar_date;
    char ar_uid;
    char ar_gid;
    int ar_mode;
    long ar_size;
};
```

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

SEE ALSO
\texttt{ar(1), arcv(1), ld(1)}.

BUGS
The archive header structure is not compatible between the PDP-11 and the VAX-11/780, due to the different word sizes. See \texttt{arcv(1)} to convert between machines.
NAME
checklist — list of file systems processed by fsck

DESCRIPTION
Checklist resides in directory /etc and contains a list of at most 15 special file names. Each special file name is contained on a separate line and corresponds to a file system. Each file system will then be automatically processed by the fsck(1M) command.

SEE ALSO
fsck(1M).
NAME

core — format of core image file

DESCRIPTION

UNIX writes out a core image of a terminated process when any of various
errors occur. See signal(2) for the list of reasons; the most common are
memory violations, illegal instructions, bus errors, and user-generated quit
signals. The core image is called core and is written in the process's work­ing
directory (provided it can be; normal access controls apply). A process
with an effective user ID different from the real user ID will not produce a
core image.

The first section of the core image is a copy of the system's per-user data
for the process, including the registers as they were at the time of the fault.
The size of this section depends on the parameter size, which is defined in
/usr/include/sys/param.h. The remainder represents the actual contents
of the user's core area when the core image was written. If the text seg­ment
is read-only and shared, or separated from data space, it is not dum­ped.

The format of the information in the first section is described by the user
structure of the system, defined in /usr/include/sys/user.h. The impor­tant stuff not detailed therein is the locations of the registers, which are
outlined in /usr/include/sys/reg.h.

SEE ALSO

adb(1), crash(1M), sdb(1), setuid(2), signal(2).
NAME
cpio — format of cpio archive

DESCRIPTION
The header structure, when the c option is not used, is:

```
struct {
  short  h_magic,
  h_dev,
  h_ino,
  h_mode,
  h_uid,
  h_gid,
  h_nlink,
  h_rdev,
  h_mtime[2],
  h_namesize,
  h_filesize[2];
}
```

When the c option is used, the header information is described by the statement below:
```
sscanf( Chdr, "%60%60%60%60%60%60%60%11l0%60%60%60%60s", 
    &Hdr.h_magic,&Hdr.h_dev,&Hdr.h_ino,&Hdr.h_mode,
    &Hdr.h_uid,&Hdr.h_gid,&Hdr.h_nlink,&Hdr.h_rdev,
    &Longtime,&Hdr.h_namesize,&Longfile,Hdr.h_name);
```

`Longtime` and `Longfile` are equivalent to `Hdr.h_mtime` and `Hdr.h_filesize`, respectively. The contents of each file is recorded in an element of the array of varying length structures, `archive`, together with other items describing the file. Every instance of `h_magic` contains the constant 070707 (octal). The items `h_dev` through `h_mtime` have meanings explained in `stat(2)`. The length of the null-terminated path name `h_name`, including the null byte, is given by `h_namesize`.

The last record of the `archive` always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with `h_filesize` equal to zero.

SEE ALSO
cpio(1), find(1), stat(2).
NAME
dir — format of directories

SYNOPSIS
#include <sys/dir.h>

DESCRIPTION
A directory behaves exactly like an ordinary file, save that no user may
write into a directory. The fact that a file is a directory is indicated by a bit
in the flag word of its i-node entry (see fs(5)). The structure of a directory
entry as given in the include file is:

    ifndef DIRSIZ
    #define DIRSIZ 14
    #endif
    struct direct
    {
        ino_t d_ino;
        char d_name(DIRSIZ);
    };

By convention, the first two entries in each directory are for . and ... The
first is an entry for the directory itself. The second is for the parent direc-
tory. The meaning of .. is modified for the root directory of the master file
system; there is no parent, so .. has the same meaning as ..

SEE ALSO
fs(5).
NAME
dump — incremental dump tape format

DESCRIPTION
The dump and restor commands are used to write and read incremental
dump magnetic tapes.

The dump tape consists of a header record, some bit mask records, a group
of records describing file system directories, a group of records describing
file system files, and some records describing a second bit mask.

The header record and the first record of each description have the format
described by the structure included by

#include <dumprestor.h>

This include file has the following contents:

```
#define NTREC 20
#define MLEN 16
#define MSIZ 4096

#define TS_TAPE 1
#define TS_INODE 2
#define TS_BITS 3
#define TS_ADDR 4
#define TS_END 5
#define TS_CLRI 6
#define MAGIC (int)60011
#define CHECKSUM (int)84446

struct spcl {
    int c_type;
    time_t c_date;
    time_t c_ddate;
    int c_volume;
    daddr_t c_tapea;
    ino_t c_inumber;
    int c_magic;
    int c_checksum;
    struct dinode c_dinode;
    int c_count;
    char c_addr[BSIZE];
} spcl;

struct idates {
    char id_name[16];
    char id_incno;
    time_t id_ddate;
};
```

NTREC is the number of 512 byte blocks in a physical tape record. MLEN is
the number of bits in a bit map word. MSIZ is the number of bit map
words.

The TS_ entries are used in the c_type field to indicate what sort of header
this is. The types and their meanings are as follows:

- **TS_TYPE** Tape volume label
- **TS_INODE** A file or directory follows. The `c_dinode` field is a copy of the disk inode and contains bits telling what sort of file this is.
- **TS_BITS** A bit mask follows. This bit mask has a one bit for each inode that was dumped.
- **TS_ADDR** A subblock to a file (`TS_INODE`). See the description of `c_count` below.
- **TS_END** End of tape record.
- **TS_CLR** A bit mask follows. This bit mask contains a one bit for all inodes that were empty on the file system when dumped.
- **MAGIC** All header blocks have this number in `c_magic`.
- **CHECKSUM** Header blocks checksum to this value.

The fields of the header structure are as follows:

- **c_type** The type of the header.
- **c_date** The date the dump was taken.
- **c_ddate** The date the file system was dumped from.
- **c_volume** The current volume number of the dump.
- **c_tapea** The current block number of this record. This is counting 512 byte blocks.
- **c_inumber** The number of the inode being dumped if this is of type `TS_INODE`.
- **c_magic** This contains the value `MAGIC` above, truncated as needed.
- **c_checksum** This contains whatever value is needed to make the block sum to `CHECKSUM`.
- **c_dinode** This is a copy of the inode as it appears on the file system.
- **c_count** This is the count of characters following that describe the file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system no block was dumped and it is replaced as a hole in the file. If there is not sufficient space in this block to describe all of the blocks in a file, `TS_ADDR` blocks will be scattered through the file, each one picking up where the last left off.
- **c_addr** This is the array of characters that is used as described above.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a `TS_END` block and then the tapemark.

The structure `idates` describes an entry of the file where dump history is kept.

**SEE ALSO**

dump(1M), restor(1M), fs(5).
NAME
errfile — error-log file format

DESCRIPTION
When hardware errors are detected by the system, an error record is generated and passed to the error-logging daemon for recording in the error log for later analysis. The default error log is /usr/adm/errfile.

The format of an error record depends on the type of error that was encountered. Every record, however, has a header with the following format:

```c
struct errhdr {
    int e_type;  /* record type */
    int e_len;   /* bytes in record (incl hdr) */
    time_t e_time; /* time of day */
};
```

The permissible record types are as follows:

```c
#define E_GOTS 010 /* Start for UNIX 3.0*/
#define E_GORT 011 /* Start for UNIX/RT */
#define E_STOP 012 /* Stop */
#define E_TCHG 013 /* Time change */
#define E_CCHG 014 /* Configuration change */
#define E_BLK  020 /* Block device error */
#define E_STRA 030 /* Stray interrupt */
#define E_PRTY 031 /* Memory parity */
```

Some records in the error file are of an administrative nature. These include the startup record that is entered into the file when logging is activated, the stop record that is written if the daemon is terminated "gracefully", and the time-change record that is used to account for changes in the system's time-of-day. These records have the following formats:

```c
struct estart {
    struct errhdr e_hdr;    /* record header */
    int e_cpu;              /* CPU type */
    int e_mmr3;             /* contents mem mgmt reg 3 */
    long e_syssize;         /* 11/70 system memory size */
    int e_bconf;            /* block dev configuration */
};
```

```c
struct eend {
    struct errhdr e_hdr;   /* record header */
};
```

```c
struct etimchg {
    struct errhdr e_hdr;    /* record header */
    time_t e_ntime;         /* new time */
};
```

Stray interrupts cause a record with the following format to be logged in the file:

```c
struct estray {
    struct errhdr e_hdr;    /* record header */
    physadr e_saddr;        /* stray loc or device addr */
    int e_sbacty;           /* active block devices */
};
```
Memory subsystem error on 11/70 processors cause the following record to be generated:

```c
struct eparity {
    struct errhdr e_hdr;  /* record header */
    int e_parreg[4];     /* memory subsys registers */
};
```

Error records for block devices have the following format:

```c
struct eblock {
    struct errhdr e_hdr;  /* record header */
    dev_t e_dev;          /* "true" major + minor dev no */
    physadr e_regloc;    /* controller address */
    int e_bacty;         /* other block I/O activity */
    struct iostat {
        long io_ops;       /* number read/writes */
        long io_misc;      /* number "other" operations */
        unsigned io_unlog; /* number unlogged errors */
    } e_stats;
    int e_bflags;        /* read/write, error, etc */
    int e_cyloff;        /* logical dev start cyl */
    daddr_t e_bnum;      /* logical block number */
    unsigned e_bytes;    /* number bytes to transfer */
    long e_memadd;       /* buffer memory address */
    unsigned e_rtry;     /* number retries */
    int e_nreg;          /* number device registers */
};
```

The following values are used in the `e_bflags` word:

```c
#define E_WRITE 0     /* write operation */
#define E_READ 1      /* read operation */
#define E_NOIO 02     /* no I/O pending */
#define E_PHYS 04     /* physical I/O */
#define E_MAP 010     /* Unibus map in use */
#define E_ERROR 020   /* I/O failed */
```

The "true" major device numbers that identify the failing device are as follows:

```c
#define RK0 0
#define RP0 1
#define RF0 2
#define TM0 3
#define TC0 4
#define HP0 5
#define HT0 6
#define HS0 7
```

SEE ALSO
errdemon(1M).
NAME
file system — format of system volume

SYNOPSIS
#include <sys/filsys.h>
#include <sys/types.h>
#include <sys/param.h>

DESCRIPTION
Every file system storage volume (e.g., RP04 disk) has a common format for certain vital information. Every such volume is divided into a certain number of 256 word (512 byte) blocks. Block 0 is unused and is available to contain a bootstrap program or other information.

Block 1 is the super-block. Starting from its first word, the format of a super-block is:

/*
* Structure of the super-block
*/
struct filsys
{
    ushort s_isize; /* size in blocks of i-list */
    daddr_t s_fsize; /* size in blocks of entire volume */
    short s_nfree; /* number of addresses in s_free */
    daddr_t s_free[NICFREE]; /* free block list */
    short s_ninode; /* number of i-nodes in s_inode */
    ino_t s_inode[NICINOD]; /* free i-node list */
    char s_flock; /* lock during free list manipulation */
    char s_ilock; /* lock during i-list manipulation */
    char s_fmod; /* super block modified flag */
    char s_ronly; /* mounted read-only flag */
    time_t s_time; /* last super block update */
    short s_dinfo[4]; /* device information */
    daddr_t s_tfree; /* total free blocks */
    ino_t s_tinode; /* total free inodes */
    char s_fname[6]; /* file system name */
    char s_fpack[6]; /* file system pack name */
};

S_isize is the address of the first data block after the i-list; the i-list starts just after the super-block, namely in block 2; thus the i-list is s_isize – 2 blocks long. S_fsize is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers; if an “impossible” block number is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The s_free array contains, in s_free[1], ..., s_free[s_nfree – 1], up to 49 numbers of free blocks. S_free[0] is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 longs of this chain member. The first of these 50 blocks is the link to the next member of the chain. To allocate a block: decrement s_nfree, and the new block is s_free[s_nfree]. If the new block number is 0, there are no blocks left, so give an error. If s_nfree became 0, read in the block named by the new block number, replace s_nfree by its first word, and copy the block numbers in the next 50 longs into the s_free array. To free a block, check if s_nfree
is 50; if so, copy \texttt{s_nfree} and the \texttt{s_free} array into it, write it out, and set \texttt{s_nfree} to 0. In any event set \texttt{s_free[s_nfree]} to the freed block's number and increment \texttt{s_nfree}.

\texttt{S_nfree} is the total free blocks available in the file system.

\texttt{S_ninode} is the number of free i-numbers in the \texttt{s_inode} array. To allocate an i-node: if \texttt{s_ninode} is greater than 0, decrement it and return \texttt{s_inode[s_ninode]}. If it was 0, read the i-list and place the numbers of all free inodes (up to 100) into the \texttt{s_inode} array, then try again. To free an i-node, provided \texttt{s_ninode} is less than 100, place its number into \texttt{s_inode[s_ninode]} and increment \texttt{s_ninode}. If \texttt{s_ninode} is already 100, do not bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the inode is really free or not is maintained in the inode itself.

\texttt{S_ninode} is the total free inodes available in the file system.

\texttt{S_flock} and \texttt{s_jlock} are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of \texttt{s_fmode} on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

\texttt{S_jonly} is a read-only flag to indicate write-protection.

\texttt{S_time} is the last time the super-block of the file system was changed, and is a double-precision representation of the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the \texttt{s_time} of the super-block for the root file system is used to set the system's idea of the time.

\texttt{S_name} is the name of the file system and \texttt{s_jpack} is the name of the pack.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 64 bytes long, so 8 of them fit into a block. Therefore, i-node \texttt{i} is located in block \((i+15)/8\), and begins \(64\times((i+15) \pmod 8)\) bytes from its start. I-node 1 is reserved for future use. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. For the format of an inode and its flags, see \texttt{inode(5)}.

\subsection*{FILES}

\begin{verbatim}
/usr/include/sys/filsys.h
/usr/include/sys/stat.h
\end{verbatim}

\subsection*{SEE ALSO}

\texttt{fsck(1M)}, \texttt{fsdb(1M)}, \texttt{mkfs(1M)}, \texttt{inode(5)}. 

\texttt{FS(5)}
NAME
fspec — format specification in text files

DESCRIPTION
It is sometimes convenient to maintain text files on UNIX with non-standard tabs, (i.e., tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets < : and : >. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

\texttt{ttabs} The \texttt{t} parameter specifies the tab settings for the file. The value of tabs must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns;
2. a \texttt{t} followed immediately by an integer \texttt{n}, indicating tabs at intervals of \texttt{n} columns;
3. a \texttt{t} followed by the name of a “canned” tab specification.

Standard tabs are specified by \texttt{t=8}, or equivalently, \texttt{t1,9,17,25,etc.} The canned tabs which are recognized are defined by the \texttt{tabs(1)} command.

\texttt{ssize} The \texttt{s} parameter specifies a maximum line size. The value of \texttt{size} must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

\texttt{mmargin} The \texttt{m} parameter specifies a number of spaces to be prepended to each line. The value of \texttt{margin} must be an integer.

\texttt{d} The \texttt{d} parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

\texttt{e} The \texttt{e} parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are \texttt{t=8} and \texttt{m0}. If the \texttt{s} parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

\texttt{<:t5,10,15 s72:>} *

If a format specification can be disguised as a comment, it is not necessary to code the \texttt{d} parameter.

Several UNIX commands correctly interpret the format specification for a file. Among them is \texttt{gath} (see \texttt{send(1C)}) which may be used to convert files to a standard format acceptable to other UNIX commands.

SEE ALSO
\texttt{ed(1), reform(1), send(1C), tabs(1)}. 

- 1 -
NAME

gps — graphical primitive string, format of graphical files

DESCRIPTION

GPS is a format used to store graphical data. Several routines have been
developed to edit and display GPS files on various devices. Also, higher
level graphics programs such as plot (in stat(1G)) and vtoc (in toc(1G))
produce GPS format output files.

A GPS is composed of five types of graphical data or primitives.

GPS PRIMITIVES

lines  The lines primitive has a variable number of points from which
zero or more connected line segments are produced. The first
point given produces a move to that location. (A move is a relo-
cation of the graphic cursor without drawing.) Successive points
produce line segments from the previous point. Parameters are
available to set color, weight, and style (see below).

arc    The arc primitive has a variable number of points to which a
curve is fit. The first point produces a move to that point. If
only two points are included a line connecting the points will
result, if three points a circular arc through the points is drawn,
and if more than three, lines connect the points. (In the future,
a spline will be fit to the points if they number greater than
three.) Parameters are available to set color, weight, and style.

text   The text primitive draws characters. It requires a single point
which locates the center of the first character to be drawn. Parameters are color, font, textsize,
and textangle.

hardware The hardware primitive draws hardware characters or gives
control commands to a hardware device. A single point locates the
beginning location of the hardware string.

comment A comment is an integer string that is included in a GPS file but
causes nothing to be displayed. All GPS files begin with a com-
ment of zero length.

GPS PARAMETERS

color  Color is an integer value set for arc, lines, and text primitives.

weight Weight is an integer value set for arc and lines primitives to indi-
cate line thickness. The value 0 is narrow weight, 1 is bold,
and 2 is medium weight.

style  Style is an integer value set for lines and arc primitives to give
one of the five different line styles that can be drawn on Tek-
tronix 4010 series storage tubes. They are:

0 solid
1 dotted
2 dot dashed
3 dashed
4 long dashed

font An integer value set for text primitives to designate the text font
to be used in drawing a character string. (Currently font is
expressed as a four-bit weight value followed by a four-bit style
value.)

textsize Textsize is an integer value used in text primitives to express the
size of the characters to be drawn. Textsize represents the height
of characters in absolute universe-units and is stored at one-fifth
this value in the size-orientation (so) word (see below).

textangle  Textangle is a signed integer value used in text primitives to express rotation of the character string around the beginning point. Textangle is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (so) word as a value 256/360 of its absolute value.

ORGANIZATION
GPS primitives are organized internally as follows:

<table>
<thead>
<tr>
<th>Lines</th>
<th>Cw points sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc</td>
<td>Cw points sw</td>
</tr>
<tr>
<td>Text</td>
<td>Cw point sw so [string]</td>
</tr>
<tr>
<td>Hardware</td>
<td>Cw point [string]</td>
</tr>
<tr>
<td>Comment</td>
<td>Cw [string]</td>
</tr>
</tbody>
</table>

cw  Cw is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word-count for that primitive.

point(s)  Point(s) is one or more pairs of integer coordinates. Text and hardware primitives only require a single point. Point(s) are values within a Cartesian plane or universe having 64K (−32K to +32K) points on each axis.

sw  Sw is the style-word and is used in lines, arc, and text primitives. The first eight bits contain color information. In arc and lines the last eight bits are divided as four bits weight and four bits style. In the text primitive the last eight bits of sw contain the font.

so  So is the size-orientation word used in text primitives. The first eight bits contain text size and the remaining eight bits contain text rotation.

string  String is a null-terminated character string. If the string does not end on a word boundary an additional null is added to the GPS file to insure word-boundary alignment.

SEE ALSO
graphics(1G).
NAME

group — group file

DESCRIPTION

*Group* contains for each group the following information:

- group name
- encrypted password
- numerical group ID
- comma-separated list of all user allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory `/etc`. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

`/etc/group`

SEE ALSO

`newgrp(1), passwd(1), crypt(3C), passwd(5)`.
NAME
initab — control information for init

DESCRIPTION
When a state is entered, init reads the file /etc/inittab. Lines in this file have the format:

state:id:flags:command

All lines in which the state field match init's current state are recognized. If a process is active under the same two character id as a recognized line, it may be terminated (signal 15), killed (signal 9), or both by including the flags t and k in the order desired. The signal is sent to all processes in the process group associated with the id. The command field is saved for later execution. The flag c requires the command to be continuously reinvoked whenever the process with that id dies. Otherwise the command is invoked a maximum of one time in the current state.

FILES
/etc/inittab
NAME
inode — format of an inode

SYNOPSIS
#include <sys/types.h>
#include <sys/ino.h>

DESCRIPTION
An i-node for a plain file or directory in a file system has the following
structure defined by <sys/ino.h>.

/* Inode structure as it appears on a disk block. */
struct dinode
{
  ushort di_mode;    /* mode and type of file */
  short  di_nlink;   /* number of links to file */
  ushort di_uid;     /* owner's user id */
  ushort di_gid;     /* owner's group id */
  off_t  di_size;    /* number of bytes in file */
  char   di_addr[40];/* disk block addresses */
  time_t di_atime;  /* time last accessed */
  time_t di_mtime;  /* time last modified */
  time_t di_ctime;  /* time created */
};
/*
* the 40 address bytes:
*  39 used; 13 addresses
*  of 3 bytes each.
*/
For the meaning of the defined types off_t and time_t see types(7).

FILES
/usr/include/sys/ino.h

SEE ALSO
stat(2), fs(5), types(7).
NAME
master — master device information table

DESCRIPTION
This file is used by the config(1M) program to obtain device information that enables it to generate the configuration files. The file consists of 3 parts, each separated by a line with a dollar sign ($) in column 1. Part 1 contains device information; part 2 contains names of devices that have aliases; part 3 contains tunable parameter information. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1 contains lines consisting of at least 10 fields and at most 13 fields, with the fields delimited by tabs and/or blanks:

Field 1: device name (8 chars. maximum).
Field 2: interrupt vector size (decimal, in bytes).
Field 3: device mask (octal) — each "on" bit indicates that the handler exists:
  000100 initialization handler
  000040 power-failure handler
  000020 open handler
  000010 close handler
  000004 read handler
  000002 write handler
  000001 ioctl handler.
Field 4: device type indicator (octal):
  000200 allow only one of these devices
  000100 suppress count field in the conf.e file
  000040 suppress interrupt vector
  000020 required device
  000010 block device
  000004 character device
  000002 floating vector
  000001 fixed vector.
Field 5: handler prefix (4 chars. maximum).
Field 6: device address size (decimal).
Field 7: major device number for block-type device.
Field 8: major device number for character-type device.
Field 9: maximum number of devices per controller (decimal).
Field 10: maximum bus request level (4 through 7).
Fields 11-13: optional configuration table structure declarations (8 chars. maximum).

Part 2 contains lines with 2 fields each:
Field 1: alias name of device (8 chars. maximum).
Field 2: reference name of device (8 chars. maximum; specified in part 1).

Part 3 contains lines with 2 or 3 fields each:
Field 1: parameter name (as it appears in description file; 20 chars. maximum)
Field 2: parameter name (as it appears in the conf.e file; 20 chars. maximum)
Field 3: default parameter value (20 chars. maximum; parameter specification is required if this field is omitted)
Devices that are not interrupt-driven have an interrupt vector size of zero. The 040 bit in Field 4 causes config(1M) to record the interrupt vector although the low.s (univec.c on the VAX-11/780) file will show no interrupt vector assignment at those locations (interrupts here will be treated as strays).

SEE ALSO
config(1M).
NAME
mnttab — mounted file system table

SYNOPSIS
struct mnttab {
    char mt_dev[10];
    char mt_filsys[10];
    short mt_ro_flg;
    time_t mt_time;
};

DESCRIPTION
Mnttab resides in directory /etc and contains a table of devices mounted by
the mount(1M) command.

Each entry is 26 bytes in length; the first 10 bytes are the null-padded name
of the place where the special file is mounted; the next 10 bytes represent
the null-padded root name of the mounted special file; the remaining 6
bytes contain the mounted special file's read/write permissions and the date
on which it was mounted.

The maximum number of entries in mnttab is based on the system
parameter NMOUNT located in /usr/src/uts/cf/conf.c, which defines the
number of allowable mounted special files.

SEE ALSO
mount(1M).
NAME
passwd — password file

DESCRIPTION
Passwd contains for each user the following information:

- login name
- encrypted password
- numerical user ID
- numerical group ID
- GCOS job number, box number, optional GCOS user ID
- initial working directory
- program to use as Shell

This is an ASCII file. Each field within each user's entry is separated from
the next by a colon. The GCOS field is used only when communicating
with that system, and in other installations can contain any desired informa-
tion. Each user is separated from the next by a new-line. If the
password field is null, no password is demanded; if the Shell field is null,
the Shell itself is used.

This file resides in directory /etc. Because of the encrypted passwords, it
can and does have general read permission and can be used, for example,
to map numerical user ID's to names.

The encrypted password consists of 13 characters chosen from a 64 charac-
ter alphabet (., /, 0-9, A-Z, a-z), except when the password is null in
which case the encrypted password is also null. Password aging is effected
for a particular user if his encrypted password in the password file is fol-
lowed by a comma and a non-null string of characters from the above
alphabet. (Such a string must be introduced in the first instance by the
super-user.) The first character of the age, M say, denotes the maximum
number of weeks for which a password is valid. A user who attempts to
login after his password has expired will be forced to supply a new one.
The next character, m say, denotes the minimum period in weeks which
must expire before the password may be changed. The remaining characters
define the week (counted from the beginning of 1970) when the password
was last changed. (A null string is equivalent to zero.) M and m have
numerical values in the range 0-63. If m = M = 0 (derived from the
string . or ..) the user will be forced to change his password the next time
he logs in (and the "age" will disappear from his entry in the password
file). If m > M (signified, e.g., by the string ./) only the super-user will be
able to change the password.

FILES
/etc/passwd

SEE ALSO
login(1), passwd(1), a64l(3C), crypt(3C), getpwent(3C), group(5).
NAME
plot — graphics interface

DESCRIPTION
Files of this format are produced by routines described in plot(3X) and are interpreted for various devices by commands described in tplot(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an l, m, n, or p instruction becomes the "current point" for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in plot(3X).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes. See tplot(1G).

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.

e erase: Start another frame of output.

f linemod: Take the following string, up to a new-line, as the style for drawing further lines. The styles are "dotted", "solid", "longdashed", "shortdashed", and "dotdashed". Effective only for the -T4014 and -Tver options of tplot(1G) (Tektronix 4014 terminal and Versatec plotter).

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of tplot(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300 space(0, 4096, 0, 4096);
DASI 300s space(0, 4096, 0, 4096);
DASI 450 space(0, 4096, 0, 4096);
Tektronix 4014 space(0, 3120, 0, 3120);
Versatec plotter space(0, 2048, 0, 2048);

SEE ALSO
graph(1G), tplot(1G), plot(3X), gps(5), term(7).
NAME
pnch — file format for card images

DESCRIPTION
The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.
NAME
profile — setting up an environment at login time

DESCRIPTION
If your login directory contains a file named .profile, that file will be executed (via the shell's exec .profile) before your session begins; .profiles are handy for setting exported environment variables and terminal modes. If the file /etc/profile exists, it will be executed for every user before the .profile. The following example is typical (except for the comments):

# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
# Tell me when new mail comes in
MAIL=/usr/mail/mynamex
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
echo "terminal: \c"
read TERM
case $TERM in
  300)
    stty cr2 nl0 tabs; tabs;;
  300s)
    stty cr2 nl0 tabs; tabs;;
  450)
    stty cr0 nl0 tabs; tabs;;
  hp)
    stty cr0 nl0 tabs; tabs;;
  745|735)
    stty cr1 nl1 -tabs; TERM=745;;
  43)
    stty cr1 nl0 -tabs;;
  4014|tek)
    stty cr0 nl0 -tabs ff1; TERM=4014; echo \"33;\"
  *)
    echo "$TERM unknown";;
esac

FILES
$HOME/.profile
/etc/profile

SEE ALSO
env(1), login(1), mail(1), sh(1), stty(1), su(1), environ(7), term(7).
NAME  
sccsfile — format of SCCS file

DESCRIPTION  
An SCCS file is an ASCII file. It consists of six logical parts: the checksum, the delta table (contains information about each delta), user names (contains login names and/or numerical group IDs of users who may add deltas), flags (contains definitions of internal keywords), comments (contains arbitrary descriptive information about the file), and the body (contains the actual text lines intermixed with control lines).

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the control character and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form DDDDD represent a five digit string (a number between 00000 and 99999).

Each logical part of an SCCS file is described in detail below.

Checksum  
The checksum is the first line of an SCCS file. The form of the line is:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The @h provides a magic number of (octal) 064001.

Delta table  
The delta table consists of a variable number of entries of the form:

@s DDDDD/DDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
.
@e <comments> ...
.
.
.
@e

The first line (@s) contains the number of lines inserted/deleted/unchanged respectively. The second line (@d) contains the type of the delta (currently, normal: D, and removed: R), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The @i, @x, and @g lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.
The @m lines (optional) each contain one MR number associated with the delta; the @c lines contain comments associated with the delta.

The @e line ends the delta table entry.

User names
The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

Flags
Keywords used internally (see admin(1) for more information on their use). Each flag line takes the form:

@f <flag> <optional text>

The following flags are defined:
@f t <type of program>
@f v <program name>
@f i
@f b
@f m <module name>
@f f <floor>
@f c <ceiling>
@f d <default-sid>
@f n
@f j
@f l <lock-releases>
@f q <user defined>

The t flag defines the replacement for the %Y% identification keyword. The v flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The i flag controls the warning/error aspect of the "No id keywords" message. When the i flag is not present, this message is only a warning; when the i flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the b flag is present the −b keyletter may be used on the get command to cause a branch in the delta tree. The m flag defines the first choice for the replacement text of the %M% identification keyword. The f flag defines the "floor" release; the release below which no deltas may be added. The c flag defines the "ceiling" release; the release above which no deltas may be added. The d flag defines the default SID to be used when none is specified on a get command. The n flag causes delta to insert a "null" delta (a delta that applies no changes) in those releases that are skipped when a delta is made in a new release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the n flag causes skipped releases to be completely empty. The j flag causes get to allow concurrent edits of the same base SID. The l flag defines a list of releases that are locked against editing (get(1) with the −e keyletter). The q flag defines the replacement for the %Q% identification keyword.
Comments
Arbitrary text surrounded by the bracketing lines @t and @T. The comments section typically will contain a description of the file's purpose.

Body
The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: insert, delete, and end, represented by:

@I DDDDD
@D DDDDD
@E DDDDD

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO
admin(1), delta(1), get(1), prs(1).
NAME

tp — magnetic tape format

DESCRIPTION

The command tp(1) dumps files to and extracts files from magtape.

Block zero contains a copy of a stand-alone bootstrap program; see tapeboot(8).

Blocks 1 through 62 contain a directory of the tape. There are 496 entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

```c
struct tpent {
    char pathnam[32];
    short mode;
    char uid;
    char uid;
    char gid;
    char spare;
    char size0;
    long time;
    short tapea; /* tape address */
    short unused[8]; /* check sum */
    short cksum;
}
```

The pathnam entry is the path name of the file when put on the tape. If the path name starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null byte. Mode, uid, gid, the sizes and time modified are the same as described under i-nodes (fs(5)). The tape address is the tape block number of the start of the contents of the file. Every file starts on a block boundary. The file occupies \((size+511)/512\) blocks of continuous tape. The checksum entry has a value such that the sum of the 32 words of the directory entry is zero.

Blocks 63 on are available for file storage.

A fake entry has a size of zero. See tp(1).

SEE ALSO

cpio(1), tp(1), fs(5), tapeboot(8).
NAME
utmp, wtmp — utmp and wtmp entry format

DESCRIPTION
The files utmp and wtmp hold user and accounting information for use by
commands such as who(1), acctcon(1M) (see acctcon(1M)), and login(1).
They have the following structure, as defined by <utmp.h>:

```c
struct utmp
{
    char  ut_line[8];  /* tty name */
    char  ut_name[8];  /* login name */
    long  ut_time;    /* time on */
};
```

FILES
/etc/utmp
/usr/adm/wtmp
/usr/include/utmp.h

SEE ALSO
acctcon(1M), login(1), who(1), write(1).
NAME
intro — introduction to games

DESCRIPTION
This section describes the recreational and educational programs found in the directory /usr/games. The availability of these programs may vary from system to system. A suggested procedure is to disallow their use during business hours by means of cron(1M).
NAME
arithmetic — provide drill in number facts

SYNOPSIS
/usr/games/arithmetic [ +/-x/ ] [ range ]

DESCRIPTION
Arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back “Right!”, and a new problem. If the answer is wrong, it replies “What?”, and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; +, -, x, and / respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +/-.

Range is a decimal number; all addends, subtrahends, differences, multipliers, divisors, and quotients will be less than or equal to the value of range. Default range is 10.

At the start, all numbers less than or equal to range are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts de novo. For almost all users, the relevant statistic should be time per problem, not percent correct.
NAME
back — the game of backgammon

SYNOPSIS
/usr/games/back

DESCRIPTION
Back is a program which provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select. In addition to selecting the opponent’s level, you may also indicate that you would like to roll your own dice during your turns (for the superstitious players). You will also be given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated.

The points are numbered 1—24, with 1 being white’s extreme inner table, 24 being brown’s inner table, 0 being the bar for removed white pieces and 25 the bar for brown. For details on how moves are expressed, type y when back asks “Instructions?” at the beginning of the game. When back first asks “Move?”, type ? to see a list of move options other than entering your numerical move.

When the game is finished, back will ask you if you want the log. If you respond with y, back will attempt to append to or create a file back.log in the current directory.

FILES
/usr/games/lib/backrules  rules file
/tmp/back*  log temp file
back.log  log file

BUGS
The only level really worth playing is “expert”, and it only plays the forward game. Back will complain loudly if you attempt to make too many moves in a turn, but will become very silent if you make too few. Doubling is not implemented.
NAME
bj - the game of black jack

SYNOPSIS
/usr/games/bj

DESCRIPTION

*bj* is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

The bet is $2 every hand.

A player “natural” (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a “push” (no money exchange).

If the dealer has an ace up, the player is allowed to make an “insurance” bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.

If the player is dealt two cards of the same value, he is allowed to “double”. He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)

If a dealt hand has a total of ten or eleven, the player may “double down”. He may double the bet ($2 to $4) and receive exactly one more card on that hand.

Under normal play, the player may “hit” (draw a card) as long as his total is not over twenty-one. If the player “busts” (goes over twenty-one), the dealer wins the bet.

When the player “stands” (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.

If both player and dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions will be asked at appropriate times. Each question is answered by *y* followed by a new-line for “yes”, or just new-line for “no”.

? (means, “do you want a hit?”)

Insurance?
Double down?

Every time the deck is shuffled, the dealer so states and the “action” (total bet) and “standing” (total won or lost) is printed. To exit, hit the interrupt key (DEL) and the action and standing will be printed.
NAME
chess — the game of chess

SYNOPSIS
/usr/games/chess

DESCRIPTION
Chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol + must be placed at the end of a line when the move on that line places the opponent’s king in check. o-o and o-o-o specify castling, king side or queen side, respectively.

The user is prompted for a move or command by a *. To play black, type first at the onset of the game. To print a copy of the board in play, type a carriage return only. Each move is echoed in the appropriate notation, followed by the program’s reply. Near the middle and end games, the program can take considerable time in computing its moves.

A ? or help may be typed to get a help message that briefly describes the possible commands.

Execute /usr/games/chessrules for further explanation.

FILES
/usr/games/chessrules executable “rules” file

DIAGNOSTICS
The most cryptic diagnostic is “eh?” which means that the input was syntactically incorrect.

BUGS
Pawns may be promoted only to queens.
NAME
  craps — the game of craps

SYNOPSIS
  /usr/games/craps

DESCRIPTION
  Craps is a form of the game of craps that is played in Las Vegas. The program simulates the roller, while the user (the player) places bets. The player may choose, at any time, to bet with the roller or with the House. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a "bankroll" of $2,000.

The program prompts with:

  bet?

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with "bet?" until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The first roll is the roll immediately following a bet.

1. On the first roll:
   7 or 11 wins for the roller;
   2, 3, or 12 wins for the House;
   any other number is the point, roll again (Rule 2 applies).

2. On subsequent rolls:
   point roller wins;
   7 House wins;
   any other number roll again.

If a player loses the entire bankroll, the House will offer to lend the player an additional $2,000. The program will prompt:

  marker?

A "yes" (or "y") consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

  Repay marker?

A reply of "yes" (or "y") indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

  How many?

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program
will prompt with “How many?” until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the total amount of money borrowed will be automatically repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

New game?

to give the House a chance to win back its money.

Any reply other than “yes” is considered “no” (except in the case of “bet?” or “How many?”). To exit, send an interrupt (break), DEL, or control-D. The program will indicate whether the player won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.
NAME
  hangman — guess the word

SYNOPSIS
  /usr/games/hangman [ arg ]

DESCRIPTION
  *Hangman* chooses a word at least seven letters long from a dictionary. The user is to guess letters one at a time.
  The optional argument *arg* names an alternate dictionary.

FILES
  /usr/lib/w2006

BUGS
  Hyphenated compounds are run together.
NAME
maze — generate a maze

SYNOPSIS
/usr/games/maze

DESCRIPTION
Maze asks a few questions and then prints a maze.

BUGS
Some mazes (especially small ones) have no solutions.
NAME
moo — guessing game

SYNOPSIS
/usr/games/moo

DESCRIPTION
Moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A "cow" is a correct digit in an incorrect position. A "bull" is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).
NAME
  quiz — test your knowledge

SYNOPSIS
  /usr/games/quiz [ -i file ] [ -t ] [ category1 category2 ]

DESCRIPTION
  Quiz gives associative knowledge tests on various subjects. It asks items
  chosen from category1 and expects answers from category2, or vice versa.
  If no categories are specified, quiz gives instructions and lists the available
  categories.

  Quiz tells a correct answer whenever you type a bare new-line. At the end
  of input, upon interrupt, or when questions run out, quiz reports a score
  and terminates.

  The -t flag specifies "tutorial" mode, where missed questions are repeated
  later, and material is gradually introduced as you learn.

  The -i flag causes the named file to be substituted for the default index
  file. The lines of these files have the syntax:

  line = category new-line | category : line
  category = alternate | category | alternate
  alternate = empty | alternate primary
  primary = character | [ category ] | option
  option = { category }

  The first category on each line of an index file names an information file.
  The remaining categories specify the order and contents of the data in each
  line of the information file. Information files have the same syntax.
  Backslash \ is used as with sh(1) to quote syntactically significant characters
  or to insert transparent new-lines into a line. When either a question or its
  answer is empty, quiz will refrain from asking it.

FILES
  /usr/games/lib/quiz/index
  /usr/games/lib/quiz/*

BUGS
  The construct "a|ab" doesn't work in an information file. Use "a{b}".
NAME
reversi — a game of dramatic reversals

SYNOPSIS
/usr/games/reversi [ [ -r ] file ]

DESCRIPTION
Reversi (also known as "friends", "Chinese friends" and "Othello") is played on an 8 by 8 board using two-sided tokens. Each player takes his turn by placing a token with his side up in an empty square. During the first four turns, players may only place tokens in the four central squares of the board. Subsequently, with each turn, a player must capture one or more of his opponent's tokens. He does this by placing one of his tokens such that it and another of his tokens embrace a solid line of his opponent's horizontally, vertically or diagonally. Captured tokens are flipped over and thus can be re-captured. If a player cannot outflank his opponent he forfeits his turn. The play continues until the board is filled or until no more outflanking is possible.

In this game, your tokens are asterisks (*) and the machine's are at-signs (@). You move by typing in the row and column at which you want to place your token as two digits (1-8), optionally separated by blanks or tabs. You can also type in:
c to continue the game after hitting break (this is only necessary if you interrupt the machine while it is deliberating),
g n to start reversi playing against itself for the next n moves (or until the break key is hit),
n to stop printing the board after each move,
o to start it up again,
p to print the board regardless,
q to quit (without dishonor),
s to print the score, and, as always,
! to escape to the shell. Control-d gets you back.

Reversi also recognizes several commands which are valid only at the start of the game, before any moves have been made. They are:
f to let the machine go first.
h n to ask for a handicap of from one to four corner squares. If you're really good, you can give the machine a handicap by typing a negative number.
l n to set the amount of look-ahead used by the machine in searching for moves. Zero means none at all. Four is the default. Greater than six means you may fall asleep waiting for the machine to move.
t n to tell reversi that you will only need n seconds to consider each move. If you fail to respond in the allotted time, you forfeit your turn.

If reversi is given a file name as an argument, it will checkpoint the game, move by move, by dumping the board onto file. The -r option will cause reversi to restart the game from file and continue logging.

DIAGNOSTICS
"Illegal!" for an illegal move, and "Huh?" for a move that even the machine cannot understand.
NAME
sky — obtain ephemerides

SYNOPSIS
/usr/games/sky [ -I ]

DESCRIPTION
Sky predicts the apparent locations of the Sun, the Moon, the planets out to Saturn, stars of magnitude at least 2.5, and certain other celestial objects. Sky reads the standard input to obtain a GMT time typed on one line with blanks separating year, month number, day, hour, and minute; if the year is missing the current year is used. If a blank line is typed the current time is used. The program prints the azimuth, elevation, and magnitude of objects which are above the horizon at the ephemeris location of Murray Hill at the indicated time. The -I flag causes it to ask for another location.

Placing a "1" input after the minute entry causes the program to print out the Greenwich Sidereal Time at the indicated moment and to print for each body its topographic right ascension and declination as well as its azimuth and elevation. Also, instead of the magnitude, the semidiameter of the body, in seconds of arc, is reported.

A "2" after the minute entry makes the coordinate system geocentric.

The effects of atmospheric extinction on magnitudes are not included; the brightest magnitudes of variable stars are marked with *.

For all bodies, the program takes into account precession and nutation of the equinox, annual (but not diurnal) aberration, diurnal parallax, and the proper motion of stars. In no case is refraction included.

The program takes into account perturbations of the Earth due to the Moon, Venus, Mars, and Jupiter. The expected accuracies are: for the Sun and other stellar bodies a few tenths of seconds of arc; for the Moon (on which particular care is lavished) likewise a few tenths of seconds. For the Sun, Moon and stars the accuracy is sufficient to predict the circumstances of eclipses and occultations to within a few seconds of time. The planets may be off by several minutes of arc.

There are lots of special options not described here, which do things like substituting named star catalogs, smoothing nutation and aberration to aid generation of mean places of stars, and making conventional adjustments to the Moon to improve eclipse predictions.

For the most accurate use of the program it is necessary to know that it actually runs in Ephemeris time.

FILES
/usr/lib/startab, /usr/lib/moontab

SEE ALSO
American Ephemeris and Nautical Almanac, for the appropriate years; also, the Explanatory Supplement to the American Ephemeris and Nautical Almanac.
NAME
    ttt, cubic — tic-tac-toe

SYNOPSIS
    /usr/games/ttt
    /usr/games/cubic

DESCRIPTION
    Ttt is the X and O game popular in the first grade. This is a learning pro-
    gram that never makes the same mistake twice.

    Although it learns, it learns slowly. It must lose nearly 80 games to com-
    pletely know the game.

    Cubic plays three-dimensional tic-tac-toe on a 4×4×4 board. Moves are
    specified as a sequence of three coordinate numbers in the range 1-4.

FILES
    /usr/games/ttt.k learning file
NAME
wump — the game of hunt-the-wumpus

SYNOPSIS
/usr/games/wump

DESCRIPTION
Wump plays the game of "Hunt the Wumpus." A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in People's Computer Company, 2, 2 (November 1973).

BUGS
It will never replace Adventure.
NAME
intro — introduction to miscellany

DESCRIPTION
This section describes miscellaneous facilities such as macro packages, character set tables, etc.
NAME
 ascii - map of ASCII character set

SYNOPSIS
 cat /usr/pub/ascii

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

```
| 000 nul | 001 soh | 002 stx | 003 etx | 004 eot | 005 enq | 006 ack | 007 bel |
| 010 bs  | 011 ht  | 012 nl  | 013 vt  | 014 sp  | 015 gr  | 016 so  | 017 si  |
| 020 dle  | 021 dc1 | 022 dc2 | 023 dc3 | 024 dc4 | 025 nak | 026 syn | 027 etb |
| 030 can  | 031 em  | 032 sub | 033 esc | 034 fs  | 035 gd  | 036 rs  | 037 us  |
| 040 sp  | 041 !  | 042 *  | 043 #  | 044 $  | 045 %  | 046 &  | 047 '  |
| 050 (    | 052 *  | 053 +  | 054 ,  | 055 -  | 056 .  | 057 /  |
| 060 0   | 062 2  | 063 3  | 064 4  | 065 5  | 066 6  | 067 7  |
| 070 8   | 072 :) | 073 ;  | 074 <  | 075 =  | 076 >  |
| 100 @   | 101 A  | 102 B  | 103 C  | 104 D  | 105 E  | 106 F  | 107 G  |
| 110 H   | 111 I  | 112 J  | 113 K  | 114 L  | 115 M  | 116 N  | 117 O  |
| 120 P   | 121 Q  | 122 R  | 123 S  | 124 T  | 125 U  | 126 V  | 127 W  |
| 130 X   | 131 Y  | 132 Z  | 133 [  | 134 \  | 135 ]  |
| 140 `   | 141 a  | 142 b  | 143 c  | 144 d  | 145 e  | 146 f  | 147 g  |
| 150 h   | 151 i  | 152 j  | 153 k  | 154 l  | 155 m  | 156 n  | 157 o  |
| 160 p   | 161 q  | 162 r  | 163 s  | 164 t  | 165 u  | 166 v  | 167 w  |
| 170 x   | 171 y  | 172 z  | 173 {  | 174 | 175 }  | 176 ^  | 177 del |
```

FILES
 /usr/pub/ascii
NAME

environ — user environment

DESCRIPTION

An array of strings called the "environment" is made available by exec(2) when a process begins. By convention, these strings have the form "name=value". The following names are used by various commands:

PATH
The sequence of directory prefixes that sh(1), time(1), nice(1), nohup(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). Login(1) sets PATH=:/bin:/usr/bin.

HOME
Name of the user's login directory, set by login(1) from the password file passwd(5).

TERM
The kind of terminal for which output is to be prepared. This information is used by commands, such as mm(1) or tplot(1G), which may exploit special capabilities of that terminal.

TZ
Time zone information. The format is xxxnzzz where xxx is standard local time zone abbreviation, n is the difference in hours from GMT, and zzz is the abbreviation for the daylight-saving local time zone, if any; for example, EST5EDT.

Further names may be placed in the environment by the export command and "name=value" arguments in sh(1), or by exec(2). It is unwise to conflict with certain shell variables that are frequently exported by .profile files: MAIL, PS1, PS2, IFS.

SEE ALSO

env(1), login(1), sh(1), exec(2), getenv(3C), profile(5), term(7).
NAME
eqnchar — special character definitions for eqn and neqn

SYNOPSIS
eqn /usr/pub/eqnchar [ files ] | troff [ options ]
neqn /usr/pub/eqnchar [ files ] | nroff [ options ]

DESCRIPTION
Eqnchar contains troff(1) and nroff(1) character definitions for constructing characters that are not available on the Wang Laboratories, Inc. C/A/T phototypesetter. These definitions are primarily intended for use with eqn(1) and neqn(1); eqnchar contains definitions for the following characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ciplus</td>
<td>⊕</td>
</tr>
<tr>
<td>citimes</td>
<td>✕</td>
</tr>
<tr>
<td>wig</td>
<td>~</td>
</tr>
<tr>
<td>−wig</td>
<td>≈</td>
</tr>
<tr>
<td>&gt; wig</td>
<td>∨</td>
</tr>
<tr>
<td>&lt; wig</td>
<td>≤</td>
</tr>
<tr>
<td>−wig</td>
<td>≥</td>
</tr>
<tr>
<td>star</td>
<td>*</td>
</tr>
<tr>
<td>bigstar</td>
<td>⋆</td>
</tr>
<tr>
<td>−dot</td>
<td>-</td>
</tr>
<tr>
<td>orsign</td>
<td>∨</td>
</tr>
<tr>
<td>andsign</td>
<td>∧</td>
</tr>
<tr>
<td>−del</td>
<td>Δ</td>
</tr>
<tr>
<td>oppA</td>
<td>∀</td>
</tr>
<tr>
<td>oppE</td>
<td>∃</td>
</tr>
<tr>
<td>angstrom</td>
<td>Å</td>
</tr>
<tr>
<td>square</td>
<td>□</td>
</tr>
<tr>
<td>circle</td>
<td>○</td>
</tr>
<tr>
<td>blot</td>
<td>■</td>
</tr>
<tr>
<td>empty</td>
<td>Ø</td>
</tr>
<tr>
<td>member</td>
<td>∈</td>
</tr>
<tr>
<td>nomem</td>
<td>ℓ</td>
</tr>
<tr>
<td>cup</td>
<td>⊙</td>
</tr>
<tr>
<td>cap</td>
<td>⊙</td>
</tr>
<tr>
<td>incl</td>
<td>⊙</td>
</tr>
<tr>
<td>subset</td>
<td>⊆</td>
</tr>
<tr>
<td>supset</td>
<td>⊇</td>
</tr>
<tr>
<td>subset</td>
<td>⊆</td>
</tr>
<tr>
<td>supset</td>
<td>⊇</td>
</tr>
</tbody>
</table>

FILES
/usr/pub/eqnchar

SEE ALSO
eqn(1), troff(1).
NAME
fcntl — file control options

SYNOPSIS
#include <fcntl.h>

DESCRIPTION
The fcntl(2) function provides for control over open files. This include file
describes requests and arguments to fcntl and open(2).
/* Flag values accessible to open(2) and fcntl(2) */
/* (The first three can only be set by open) */
#define O_RDONLY 0    /* Non-blocking I/O */
#define O_WRONLY 1    /* append (writes guaranteed at the end) */
#define O_RDWR 2
#define O_NDELAY 4
#define O_APPEND 10
#define O_CREAT 20
#define O_TRUNC 40
#define O_EXCL 60
#define _F_DUPFD 80
#define _GETFD 160
#define _SETFD 240
#define _GETFL 320
#define _SETFL 400

/* fcntl(2) requests */
#define DUPFD 0 /* Duplicate fildes */
#define GETFD 1 /* Get fildes flags */
#define SETFD 2 /* Set fildes flags */
#define GETFL 3 /* Get file flags */
#define SETFL 4 /* Set file flags */

SEE ALSO
fcntl(2), open(2).
NAME
greek — graphics for the extended TTY-37 type-box

SYNOPSIS
cat /usr/pub/greek [ | greek -Tterminal ]

DESCRIPTION
Greek gives the mapping from ASCII to the "shift-out" graphics in effect between SO and SI on TELETYPEn Model 37 terminals equipped with a 128-character type-box. These are the default greek characters produced by nroff(1). The filters of greek(1) attempt to print them on various other terminals. The file contains:

<table>
<thead>
<tr>
<th>Character</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>α A</td>
</tr>
<tr>
<td>Gamma</td>
<td>Γ G</td>
</tr>
<tr>
<td>Epsilon</td>
<td>ε S</td>
</tr>
<tr>
<td>Theta</td>
<td>Θ T</td>
</tr>
<tr>
<td>Lambda</td>
<td>Λ E</td>
</tr>
<tr>
<td>Xi</td>
<td>ξ X</td>
</tr>
<tr>
<td>Rho</td>
<td>ρ K</td>
</tr>
<tr>
<td>Tau</td>
<td>τ I</td>
</tr>
<tr>
<td>Psi</td>
<td>ψ V</td>
</tr>
<tr>
<td>Omega</td>
<td>Ω Z</td>
</tr>
<tr>
<td>Partial</td>
<td>∂ ]</td>
</tr>
<tr>
<td>Integral</td>
<td>∫ ]</td>
</tr>
</tbody>
</table>

FILES
/usr/pub/greek

SEE ALSO
300(1), 4014(1), 450(1), greek(1), hp(1), tc(1), troff(1).
NAME
man — macros for formatting entries in this manual

SYNOPSIS
nroff — man files
troff — man [-rs1] files

DESCRIPTION
These troff(1) macros are used to lay out the format of the entries of this manual. A skeleton entry may be found in the file /usr/man/man0/skeleton. These macros are used by the man(1) command.

The default page size is 8.5" × 11", with a 6.5" × 10" text area; the -rs1 option reduces these dimensions to 6" × 9" and 4.75" × 8.375", respectively; this option (which is not effective in nroff(1)) also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The -rV2 option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining; this option should not be confused with the -Tvp option of the man(1) command, which is available at some UNIX sites.

Any text argument below may be one to six "words".

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., .I, .RB, .SM. Tab stops are neither used nor set by any macro except .DT and .TH.

Default units for indents in are ens. When in is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in troff, 5 ens in nroff—which corresponds to 0.5" in the default page size) by .TH, .PP, and .RS, and restored by .RE.

.TH t s c n Set the title and entry heading; t is the title, s is the section number, c is extra commentary, e.g., "local", n is new manual name. Invokes .DT (see below).

.SH text Place subhead text, e.g., SYNOPSIS, here.

.SS text Place sub-subhead text, e.g., Options, here.

.B text Make text bold.

.I text Make text italic.

.SM text Make text 1 point smaller than default point size.

.RI a b Concatenate roman a with italic b, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of roman, italic, and bold:

.IP t in Same as .TP in with tag t; often used to get an indented paragraph without a tag.
.RS in  Increase relative indent (initially zero). Indent all output an extra in units from the current left margin.
.RE k  Return to the kth relative indent level (initially, k=1; k=0 is equivalent to k=1); if k is omitted, return to the most recent lower indent level.
.PM m  Produces proprietary markings; where m may be P for PRIVATE, N for NOTICE, BP for BELL LABORATORIES PROPRIETARY, or BR for BELL LABORATORIES RESTRICTED.
.DT    Restore default tab settings (every 7.2 ens in troff, 5 ens in nroff).
.PD v  Set the interparagraph distance to v vertical spaces. If v is omitted, set the interparagraph distance to the default value (0.4v in troff, 1v in nroff).

The following strings are defined:
\*R    * in troff(1), (Reg.) in nroff(1).
\*S    Change to default type size.

The following number registers are given default values by .TH:
IN    Left margin indent relative to subheads (default is 7.2 ens in troff, 5 ens in nroff).
LL    Line length including IN.
PD    Current interparagraph distance.

CAVEATS
In addition to the macros, strings, and number registers mentioned above, there are defined a number of internal macros, strings, and number registers. Except for names predefined by troff(1) and number registers d, m, and y, all such internal names are of the form XA, where X is one of ), ], and }, and A stands for any alphanumeric character.

If a manual entry needs to be preprocessed by cw(1), eqn(1) (or neqn), and/or tbl(1), it must begin with a special line (described in man(1)), causing the man command to invoke the appropriate preprocessor(s).

The programs that prepare the Table of Contents and the Permuted Index for this Manual assume the NAME section of each entry consists of a single line of input that has the following format:

name[, name, name ...] \ - explanatory text

The macro package increases the inter-word spaces (to eliminate ambiguity) in the SYNOPSIS section of each entry.

The macro package itself uses only the roman font (so that one can replace, for example, the bold font by the constant-width font—see cw(1)). Of course, if the input text of an entry contains requests for other fonts (e.g., .I, .RB, \fI), the corresponding fonts must be mounted.

FILES
/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.[nt].[dt].an
/usr/lib/macros/ucmp.[nt].an
/usr/man/man0/skeleton

SEE ALSO
man(1), troff(1).

BUGS
If the argument to .TH contains any blanks and is not enclosed by double quotes (""), there will be bird-dropping-like things on the output.
NAME

mm — the MM macro package for formatting documents

SYNOPSIS

```
mm [ options ] [ files ]
nroff -mm [ options ] [ files ]
nroff -cm [ options ] [ files ]
mmt [ options ] [ files ]
troff -mm [ options ] [ files ]
troff -cm [ options ] [ files ]
```

DESCRIPTION

This package provides a formatting capability for a very wide variety of
documents. It is the standard package used by the BTL typing pools and
documentation centers. The manner in which a document is typed in and
edited is essentially independent of whether the document is to be eventu­
ally formatted at a terminal or is to be phototypeset. See the references
below for further details.

The -mm option causes nroff(1) and troff(1) to use the non-compacted
version of the macro package, while the -cm option results in the use of
the compacted version, thus speeding up the process of loading the macro
package.

FILES

```
/usr/lib/tmac/tmac.m       pointer to the non-compacted version of
/usr/lib/macros/mm[nt]    non-compacted version of the package
/usr/lib/macros/cmp.[nt].[dt].m  compacted version of the package
/usr/lib/macros/ucmp.[nt].m  initializers for the compacted version of
```

the package

SEE ALSO

mm(1), mmt(1), troff(1).

MM—Memorandum Macros by D. W. Smith and J. R. Mashey.
Typing Documents with MM by D. W. Smith and E. M. Piskorik.
NAME
mv — a macro package for making view graphs

SYNOPSIS
mvt [ options ] [ files ]
troff -mv [ options ] [ files ]

DESCRIPTION
This package provides an easy-to-use facility for making view graphs and projection slides in a variety of formats. A dozen or so macros are provided that accomplish most of the formatting tasks needed in making transparencies. All of the facilities of troff(1), eqn(1), and tbl(1) are available for more difficult tasks. The output can be previewed on most terminals, and, in particular, on the Tektronix 4014 and on the Versatec printer. See the reference below for further details.

FILES
/usr/lib/tmac/tmac.v

SEE ALSO
eqn(1), mvt(1), tbl(1), troff(1).
NAME
regexp — regular expression compile and match routines

SYNOPSIS
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>
#include "regexp.h">

cbar .compile(instring, expbuf, endbuf, eof)
cbar .instring, .expbuf, .endbuf;
int step(string, expbuf)
cbar .string, .expbuf;
char *string, *expbuf;

DESCRIPTION
This page describes general purpose regular expression matching routines in the form of ed(1), defined in /usr/include/regexp.h. Programs such as ed(1), sed(1), grep(1), bs(1), expr(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the compile routine.

GETC() Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.

PEEKc() Return the next character in the regular expression. Successive calls to PEEKC() should return the same character (which should also be the next character returned by GETC()).

UNGETC(c) Cause the argument c to be returned by the next call to GETC() (and PEEKC()). No more than one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC(c) is always ignored.

RETURN(pointer) This macro is used on normal exit of the compile routine. The value of the argument pointer is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(val) This is the abnormal return from the compile routine. The argument val is an error number (see table below for meanings). This call should never return.
The syntax of the compile routine is as follows:

```
compile(instring, expbuf, endbuf, eof)
```

The first parameter instring is never used explicitly by the compile routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *)0) for this parameter.

The next parameter expbuf is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter endbuf is one more that the highest address that the compiled regular expression may be placed. If the compiled expression cannot fit in (endbuf - expbuf) bytes, a call to ERROR(50) is made.

The parameter eof is the character which marks the end of the regular expression. For example, in ed(1), this character is usually a '/.

Each programs that includes this file must have a #define statement for INIT. This definition will be placed right after the declaration for the function compile and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEEKC() and UNGETC(). Otherwise it can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the example below of the declarations taken from grep(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function step. The call to step is as follows:

```
step(string, expbuf)
```

The first parameter to step is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter expbuf is the compiled regular expression which was obtained by a call of the function compile.

The function step returns one, if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to step. The variable set in step is loc1. This is a pointer to the first character that matched the regular expression. The variable loc2, which is set by the function advance, points the character after the last character that matches the regular expression. Thus if the regular expression matches the entire
line, loc1 will point to the first character of \textit{string} and loc2 will point to the null at the end of \textit{string}.

\textit{Step} uses the external variable \texttt{cire!} which is set by \texttt{compile} if the regular expression begins with `\textbf{\textperiodcentered}`. If this is set then \textit{step} will only try to match the regular expression to the beginning of the string. If more than one regular expression is to be compiled before the the first is executed the value of \texttt{cire!} should be saved for each compiled expression and \texttt{cire!} should be set to that saved value before each call to \textit{step}.

The function \textit{advance} is called from \textit{step} with the same arguments as \textit{step}. The purpose of \textit{step} is to step through the \textit{string} argument and call \textit{advance} until \textit{advance} returns a one indicating a match or until the end of \textit{string} is reached. If one wants to constrain \textit{string} to the beginning of the line in all cases, \textit{step} need not be called, simply call \textit{advance}.

When \textit{advance} encounters a `\textstar` or `\{ \} sequence in the regular expression it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, \textit{advance} will back up along the string until it finds a match or reaches the point in the string that initially matched the `\textstar` or `\{ \}`. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer \texttt{loes} is equal to the point in the string at sometime during the backing up process, \textit{advance} will break out of the loop that backs up and will return zero. This is used be \texttt{ed(1)} and \texttt{sed(1)} for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like \texttt{s/y*/g} do not loop forever.

The routines \texttt{ecmp} and \texttt{getrange} are trivial and are called by the routines previously mentioned.

\textbf{EXAMPLES}

The following is an example of how the regular expression macros and calls look from \texttt{grep(1)}:

\begin{verbatim}
#define INIT register char *sp = instring;
#define GETC( ) (*sp++)
#define PEEKC( ) (*sp)
#define UNGETC(c) (--sp)
#define RETURN(c) return;
#define ERROR(c) regerr()
#include <regexp.h>
...
compile(*argv, expbuf, &expbuf[ESIZE], ^0');
...
if(step(linebuf, expbuf))
    succeed();
\end{verbatim}

\textbf{FILES}

\texttt{/usr/include/regexp.h}

\textbf{SEE ALSO}

\texttt{ed(1), grep(1), sed(1)}.

\textbf{BUGS}

The handling of \texttt{cire!} is kludgy.

The routine \texttt{ecmp} is equivalent to the Standard I/O routine \texttt{strncmp} and should be replaced by that routine.

The actual code is probably easier to understand than this manual page.
NAME
stat — data returned by stat system call

SYNOPSIS

```c
#include <sys/types.h>
#include <sys/stat.h>
```

DESCRIPTION
The system calls `stat` and `fstat(2)` return data whose structure is defined by this include file. The encoding of the field `st_mode` is defined in this file also.

```c
struct stat
{
    dev_t     st_dev;
    ino_t     st_ino;
    ushort    st_mode;
    short     st_nlink;
    ushort    st_uid;
    ushort    st_gid;
    dev_t     st_rdev;
    off_t     st_size;
    time_t    st_atime;
    time_t    st_mtime;
    time_t    st_ctime;
};
```

```c
#define S_IFMT          0170000 /* type of file */
#define S_IFDIR         0040000 /* directory */
#define S_IFCHR         0020000 /* character special */
#define S_IFBLK         0060000 /* block special */
#define S_IFREG         0100000 /* regular */
#define S_IFIFO         0010000 /* fifo */
#define S_ISUID         04000 /* set user id on execution */
#define S_ISGID         02000 /* set group id on execution */
#define S_ISVTX         01000 /* save swapped text even after use */
#define S_IREAD         004000 /* read permission, owner */
#define S_IWRITE        002000 /* write permission, owner */
#define S_IEXEC         001000 /* execute/search permission, owner */
```

FILES
`/usr/include/sys/types.h`
`/usr/include/sys/stat.h`

SEE ALSO
stat(2).
NAME

term — conventional names

DESCRIPTION

These names are used by certain commands (e.g., nroff(1), mm(1), man(1), tabs(1)) and are maintained as part of the shell environment (see sh(1), profile(5), and environ(7)) in the variable STERM:

1520  Datamedia 1520
1620  Diablo 1620 and others using the HyType II printer
1620-12  same, in 12-pitch mode
2621  Hewlett-Packard HP2621 series
2631  Hewlett-Packard 2631 line printer
2631-c  Hewlett-Packard 2631 line printer - compressed mode
2631-e  Hewlett-Packard 2631 line printer - expanded mode
2640  Hewlett-Packard HP2640 series
2645  Hewlett-Packard HP264n series (other than the 2640 series)
300  DASI/DTC/GSI 300 and others using the HyType I printer
300-12  same, in 12-pitch mode
300s  DASI/DTC/GSI 300s
382  DTC 382
300s-12  same, in 12-pitch mode
3045  Datamedia 3045
33  TELETYPER Model 33 KSR
37  TELETYPER Model 37 KSR
40-2  TELETYPER Model 40/2
4000A  Trendata 4000A
4014  Tektronix 4014
43  TELETYPER Model 43 KSR
450  DASI 450 (same as Diablo 1620)
450-12  same, in 12-pitch mode
735  Texas Instruments TI735 and TI725
745  Texas Instruments TI745
dumb  generic name for terminals that lack reverse line-feed and other special escape sequences
hp  Hewlett-Packard (same as 2645)
lp  generic name for a line printer
tn1200  General Electric TermiNet 1200
tn300  General Electric TermiNet 300

Up to 8 characters, chosen from [−a−z0−9], make up a basic terminal name. Terminal sub-models and operational modes are distinguished by suffixes beginning with a -. 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.

Commands whose behavior depends on the type of terminal should accept arguments of the form -Tterm 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 STERM, which, in turn, should contain term.

SEE ALSO

mm(1), nroff(1), tplot(1G), sh(1), stty(1), tabs(1), profile(5), environ(7).

BUGS

This is a small candle trying to illuminate a large, dark problem. Programs that ought to adhere to this nomenclature do so somewhat fitfully.
NAME
  types — primitive system data types

SYNOPSIS
  #include <sys/types.h>

DESCRIPTION
  The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```c
struct { int r[1]; } *  physadr;
typedef long   daddr_t;
typedef char *  caddr_t;
typedef unsigned short ushort;
typedef ushort  ino_t;
#ifdef vax
  typedef short   cnt_t;
#else
  typedef char    cnt_t;
#endif
typedef long    time_t;
#ifdef vax
  typedef int     label_t[10];
#else
  typedef int     label_t[6];
#endif
typedef short   dev_t;
typedef long    off_t;
typedef long    paddr_t;
```

The form `daddr_t` is used for disk addresses except in an i-node on disk, see `fs(5)`. Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The `label_t` variables are used to save the processor state while another process is running.

SEE ALSO
  `fs(5)`.
INTRO(8)

NAME
intro — introduction to system maintenance procedures

DESCRIPTION
This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions on such topics as boot procedures, recovery from crashes, file backups, etc.

BUGS
No manual can take the place of good, solid experience.
NAME
70boot — 11/70 bootstrap procedures

DESCRIPTION
To bootstrap programs from a wide range of storage media, the PDP-11/70 has a dedicated diagnostic bootstrap loader called the M9301-YC. The M9301-YC contains two 256 word ROMs (17 765 000 to 17 765 776 and 17 773 000 to 17 773 776) which contain hardware verification diagnostic routines and bootstrap loader routines.

The diagnostic portion tests the basic CPU to verify correct operation. The branches, registers, all addressing modes, and most of the instructions are checked. If requested, memory management and the UNIBUS map are turned on. Then memory is tested from virtual address 001 000 to 157 776 with the cache disabled. Next the cache is enabled and tested.

The physical memory tested is determined by the console switches. Console switches <15:12> are used to set physical address bits <19:16>. If console switches <15:12> are zero, memory management and the UNIBUS map will not be enabled, so that physical memory 0 to 157 776 will be used. If console switches <15:12> are non-zero, then memory management, the UNIBUS map, and 22-bit mapping will be enabled. Table I describes the physical address ranges for each switch setting. In all cases, virtual addresses 160 000 to 177 776 are mapped to the peripheral page, physical addresses 17 600 000 to 17 777 776. Note that physical memory above 512K words is not accessible by this program even though the physical memory maximum is 1920K words.

The bootstrap portion of the M9301-YC attempts to BOOT from the device and drive number specified in the console switches. Console switches <7:3> select the device and console switches <2:0> select the drive number. Table II describes the devices selected for each switch setting. If console switches <7:0> are zero, the program will read a set of switches on the M9301-YC, set by field service, to determine a default boot device and drive number. These switches appear at location 17 773 024, however bits <8:4> select the device and bits <3:1> select the drive number.

Having selected a boot device, the program will read a block of data into memory starting at virtual address 0, and then jump to virtual address 0. Table III describes the details of booting for each device. Note that the physical address selection is the same as described above for the diagnostic portion. Excluding the RX11/RX01 floppy disk, bootstrap programs must fit in one block of 256 words, even though this program may read in more.

To start operation of the bootstrap loader, halt the CPU by depressing the HALT switch, set the Address Display select switch to Console Physical, set the Console Switch Register to 165 000, and depress the Load Address switch. Then reset the console switches to 0 and set switches <15:12> for the desired physical memory (normally 0) and switches <7:0> for the desired device (normally 0 for the default boot). Put the HALT switch in the ENABLE position and depress the START switch. The diagnostic portion will then run followed by the boot from the selected media. This takes approximately three seconds.

Any error during the diagnostic portion will cause the CPU to halt. Table IV lists the addresses and error indications. Only cache errors are recoverable in that by pressing the CONTINUE switch the program will disable the cache by forcing misses and proceed to the bootstrap section. If there is an error in reading the boot block, the program will do a RESET instruction and jump back to the memory test section (test 24) and then attempt to
boot again.

**SEE ALSO**

romboot(8), unixboot(8).

**Table I — Physical Memory Selection**

<table>
<thead>
<tr>
<th>Console switches &lt;15:12&gt;</th>
<th>Physical addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00 000 000 - 00 157 776</td>
</tr>
<tr>
<td>01</td>
<td>00 200 000 - 00 357 776</td>
</tr>
<tr>
<td>02</td>
<td>00 400 000 - 00 557 776</td>
</tr>
<tr>
<td>03</td>
<td>00 600 000 - 00 757 776</td>
</tr>
<tr>
<td>04</td>
<td>01 000 000 - 01 157 776</td>
</tr>
<tr>
<td>05</td>
<td>01 200 000 - 01 357 776</td>
</tr>
<tr>
<td>06</td>
<td>01 400 000 - 01 557 776</td>
</tr>
<tr>
<td>07</td>
<td>01 600 000 - 01 757 776</td>
</tr>
<tr>
<td>10</td>
<td>02 000 000 - 02 157 776</td>
</tr>
<tr>
<td>11</td>
<td>02 200 000 - 02 357 776</td>
</tr>
<tr>
<td>12</td>
<td>02 400 000 - 02 557 776</td>
</tr>
<tr>
<td>13</td>
<td>02 600 000 - 02 757 776</td>
</tr>
<tr>
<td>14</td>
<td>03 000 000 - 03 157 776</td>
</tr>
<tr>
<td>15</td>
<td>03 200 000 - 03 357 776</td>
</tr>
<tr>
<td>16</td>
<td>03 400 000 - 03 557 776</td>
</tr>
<tr>
<td>17</td>
<td>03 600 000 - 03 757 776</td>
</tr>
</tbody>
</table>

**Table II — Device selection**

<table>
<thead>
<tr>
<th>Console switches &lt;7:3&gt;</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>illegal</td>
</tr>
<tr>
<td>01</td>
<td>TM11/TU10 Tape</td>
</tr>
<tr>
<td>02</td>
<td>TC11/TU56 Tape</td>
</tr>
<tr>
<td>03</td>
<td>RK11/RK05 Disk</td>
</tr>
<tr>
<td>04</td>
<td>RP11/RP03 Disk</td>
</tr>
<tr>
<td>05</td>
<td>reserved</td>
</tr>
<tr>
<td>06</td>
<td>RH70/TU16 Tape</td>
</tr>
<tr>
<td>07</td>
<td>RH70/RP04 Disk</td>
</tr>
<tr>
<td>10</td>
<td>RH70/RS04 Disk</td>
</tr>
<tr>
<td>11</td>
<td>RX11/RX01 Disk</td>
</tr>
<tr>
<td>12-37</td>
<td>illegal</td>
</tr>
</tbody>
</table>
Table III — Boot procedures

TU10: Select drive, wait until online,
      set to 800 bpi, rewind,
      space forward 1 record,
      read 1 record (maximum of 256 words).
TU56: Select drive, rewind, read 512 words.
RK05 or RP03: Select drive, start at block 0, read 512 words.
TU16: Select drive on first TM02, wait until online,
      set to 800 bpi, PDP format, rewind,
      space forward 1 record,
      read 1 record (maximum of 512 words).
RP04: Select drive, read-in preset,
      set to 16-bits/word, ECC inhibit,
      start at block 0, read 512 words.
RS04: Select drive, start at block 0, read 512 words.
RX01: Select drive 0 or 1,
      start at track 1, sector 1 (IBM standard),
      read 64 words.
Table IV — Error halts

<table>
<thead>
<tr>
<th>Address displayed</th>
<th>Test</th>
<th>Subsystem under test</th>
</tr>
</thead>
<tbody>
<tr>
<td>17 765 004</td>
<td>1</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 020</td>
<td>2</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 036</td>
<td>3</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 052</td>
<td>4</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 066</td>
<td>5</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 076</td>
<td>6</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 134</td>
<td>7</td>
<td>Register data path</td>
</tr>
<tr>
<td>17 765 146</td>
<td>10</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 166</td>
<td>11</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 204</td>
<td>12</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 214</td>
<td>13</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 222</td>
<td>14</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 236</td>
<td>14</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 260</td>
<td>15</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 270</td>
<td>16</td>
<td>Branch</td>
</tr>
<tr>
<td>17 765 312</td>
<td>16</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 346</td>
<td>17</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 360</td>
<td>20</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 374</td>
<td>20</td>
<td>CPU instruction</td>
</tr>
<tr>
<td>17 765 450</td>
<td>21</td>
<td>Kernel PAR</td>
</tr>
<tr>
<td>17 765 474</td>
<td>22</td>
<td>Kernel PDR</td>
</tr>
<tr>
<td>17 765 510</td>
<td>23</td>
<td>JSR</td>
</tr>
<tr>
<td>17 765 520</td>
<td>23</td>
<td>JSR</td>
</tr>
<tr>
<td>17 765 530</td>
<td>23</td>
<td>RTS</td>
</tr>
<tr>
<td>17 765 542</td>
<td>23</td>
<td>RTI</td>
</tr>
<tr>
<td>17 765 550</td>
<td>23</td>
<td>JMP</td>
</tr>
<tr>
<td>17 765 742</td>
<td>25</td>
<td>Main memory data compare error</td>
</tr>
<tr>
<td>17 765 760</td>
<td>25</td>
<td>Main memory data compare error</td>
</tr>
<tr>
<td>17 776 000</td>
<td>25</td>
<td>Main memory parity error; no recovery possible from this error</td>
</tr>
<tr>
<td>17 773 644</td>
<td>26</td>
<td>Cache memory data compare error</td>
</tr>
<tr>
<td>17 773 654</td>
<td>26</td>
<td>Cache memory no hit, recoverable</td>
</tr>
<tr>
<td>17 773 736</td>
<td>27</td>
<td>Cache memory data compare error</td>
</tr>
<tr>
<td>17 773 746</td>
<td>27</td>
<td>Cache memory data compare error</td>
</tr>
<tr>
<td>17 773 764</td>
<td>25/26</td>
<td>Cache memory parity error, recoverable</td>
</tr>
</tbody>
</table>
NAME
crash — what to do when the system crashes

DESCRIPTION
This entry gives at least a few clues about how to proceed if the system crashes. It can’t pretend to be complete.

How to bring it back up. If the reason for the crash is not evident (see below for guidance on "evident") you may want to try to dump the system if you feel up to debugging. At the moment a dump can be taken only on magtape. With a tape mounted and ready, stop the machine, load address 44(8) (on the PDP-11), 400(16) (on the VAX-11/780; see vaxops(8)), and start. This should write a copy of all of core on the tape with an EOF mark. Be sure the ring is in, the tape is ready, and the tape is clean and new.

In restarting after a crash, always bring up the system single-user, as specified in unixboot(8) as modified for your particular installation. Then perform an fsck(1M) on all file systems which could have been in use at the time of the crash. If any serious file system problems are found, they should be repaired. When you are satisfied with the health of your disks, check and set the date if necessary, then come up multi-user.

To even boot UNIX at all, three files (and the directories leading to them) must be intact. First, the initialization program /etc/init must be present and executable. If it is not, the CPU will loop in user mode at location 6(8) (PDP-11), 13(16) (VAX-11/780). For init to work correctly, /dev/console and /bin/sh must be present. If either does not exist, the symptom is best described as thrashing. Init will go into a fork/exec loop trying to create a Shell with proper standard input and output.

If you cannot get the system to boot, a runnable system must be obtained from a backup medium. The root file system may then be doctored as a mounted file system as described below. If there are any problems with the root file system, it is probably prudent to go to a backup system to avoid working on a mounted file system.

Repairing disks. The first rule to keep in mind is that an addled disk should be treated gently; it shouldn’t be mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be copied before trying surgery on it. This is an area where experience and informed courage count for much.

Fsck(1M) is adept at diagnosing and repairing file system problems. It first identifies all of the files that contain bad (out of range) blocks or blocks that appear in more than one file. Any such files are then identified by name and fsck requests permission to remove them from the file system. Files with bad blocks should be removed. In the case of duplicate blocks, all of the files except the most recently modified should be removed. The contents of the survivor should be checked after the file system is repaired to ensure that it contains the proper data. (Note that running fsck with the -n option will cause it to report all problems without attempting any repair.)

Fsck will also report on incorrect link counts and will request permission to adjust any that are erroneous. In addition, it will reconnect any files or directories that are allocated but have no file system references to a "lost+found" directory. Finally, if the free list is bad (out of range, missing, or duplicate blocks) fsck will, with the operators concurrence, construct a new one.
Why did it crash? UNIX types a message on the console typewriter when it voluntarily crashes. Here is the current list of such messages, with enough information to provide a hope at least of the remedy. The message has the form “panic: ...”, possibly accompanied by other information. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

blkdev
The getblk routine was called with a nonexistent major device as argument. Definitely hardware or software error.

devtab
Null device table entry for the major device used as argument to getblk. Definitely hardware or software error.

iinit
An I/O error reading the super-block for the root file system during initialization.

no fs
A device has disappeared from the mounted-device table. Definitely hardware or software error.

no imt
Like “no fs”, but produced elsewhere.

no clock
During initialization, neither the line nor programmable clock was found to exist.

I/O error in swap
An unrecoverable I/O error during a swap. Really shouldn’t be a panic, but it is hard to fix.

out of swap space
A program needs to be swapped out, and there is no more swap space. It has to be increased. This really shouldn’t be a panic, but there is no easy fix.

trap
An unexpected trap has occurred within the system. This is accompanied by three numbers: a “ka6”, which is the contents of the segmentation register for the area in which the system’s stack is kept; “aps”, which is the location where the hardware stored the program status word during the trap; and a “trap type” which encodes which trap occurred. The trap types are:

PDP-11:
0  bus error
1  illegal instruction
2  BPT/trace
3  IOT
4  power fail
5  EMT
6  recursive system call (TRAP instruction)
7  11/70 cache parity, or programmed interrupt
8  floating point trap
9  segmentation violation

VAX-11/780:
0  reserved addressing fault
1  illegal instruction
2  BPT instruction trap
3  XFC instruction trap
reserved operand fault
recursive system call (CHMK instruction)
floating point trap
software level 1 (reschedule) trap
segmentation violation
protection fault
compatibility mode fault

In some of these cases it is possible for octal 40 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred. If you wish to examine the stack after such a trap, either dump the system, or use the console switches to examine core; the required address mapping is described below.

Interpreting dumps. All file system problems should be taken care of before attempting to look at dumps. The dump should be read into the file /usr/tmp/core; cp(1) will do. At this point, you should execute ps -el -c /usr/tmp/core and who to print the process table and the users who were on at the time of the crash.

Additional information for the PDP-11. You should dump (adb(1)) the first 30 bytes of /usr/tmp/core. Starting at location 4, the registers R0, R1, R2, R3, R4, R5, SP and KDSA6 (KISA6 for 11/40s) are stored. If the dump had to be restarted, R0 will not be correct. Next, take the value of KA6 (location 22(8) in the dump) multiplied by 100(8) and dump 2000(8) bytes starting from there. This is the per-process data associated with the process running at the time of the crash. Relabel the addresses 140000 to 141776. R5 is C's frame or display pointer. Stored at (R5) is the old R5 pointing to the previous stack frame. At (R5)+2 is the saved PC of the calling procedure. Trace this calling chain until you obtain an R5 value of 141756, which is where the user's R5 is stored. If the chain is broken, you have to look for a plausible R5, PC pair and continue from there. Each PC should be looked up in the system's name list using adb(1) and its : command, to get a reverse calling order. In most cases this procedure will give an idea of what is wrong. A more complete discussion of system debugging is impossible.

SEE ALSO
adb(1), fsck(1M), unixboot(8), vaxops(8).
NAME
diskboot — disk bootstrap programs

DESCRIPTION
There are several programs available to accomplish bootstraps off of a
variety of disks. These programs reside in the directory /stand.

The program must be located in block 0 of the disk pack. The space availa-
able for the program is thus only one block (256 words) which severely con-
strains the amount of error handling. Block 0 is unused by the UNIX file
system, so this does not affect normal file system operation. To boot, the
program must be read into memory starting at address 0 and started at
address 0. This may be accomplished by standard DEC ROM bootstraps,
special ROM bootstraps, or manual procedures.

After initial load, the program relocates itself to high core as specified when
assembled (typically 24K words, maximum of 28K). Next, memory below
the program is cleared and the prompt #$ is typed on the console. A one
digit field specifying the disk drive is expected. For example, 2 would
 correspond to drive 2, starting at cylinder 0. The last word in the boot
block contains a cylinder offset, initially zero, which may be changed to
access another section of the disk pack. No error checking is done on this
field; invalid data will cause unpredictable results. Also, there is no error
checking on disk reads.

After the file system select, the program prompts with =. The user must
then enter the UNIX path name of the desired file. The #$ character will
erase the last character typed, the @ character will kill the entire line, and
A through Z is translated to a through z. Also, carriage return (CR) is
mapped into line-feed (LF) on input, and LF is output as CR-LF. The
upper-case to lower-case conversion is used to handle upper-case-only ter-
minals such as the TELETYPE® Model 33 or the DEC LA30. Therefore, a
file name with upper case characters cannot be booted using this procedure.

After the name has been completely entered by typing CR or LF, the pro-
gram searches the file system specified for the path name. Note, the path
name may be any valid UNIX file system path name. If the file does not
exist, or if the file is a directory or special file, the bootstrap starts over and
prompts with #$ . Otherwise, the file is read into memory starting at address
0. If address 0 contains 000 407, a UNIX a.out program is assumed and the
first 8 words are stripped off by relocating the loaded program toward
address 0. Finally, a jump to address 0 is done by executing jsr pc, &$0.

FILES
/usr/src/stand  source directory

SEE ALSO
a.out(5), fs(5), tapeboot(8), unixboot(8).
NAME
etp — Equipment Test Package

DESCRIPTION
Etp is a stand-alone program that exercises the PDP-11 or VAX-11/780 hardware in a manner that simulates the load imposed by a UNIX system. Its output consists of reports that can be formatted to resemble the output of DEC diagnostic programs.

SEE ALSO
errpt(1M).
NAME
  filesave, tapesave — daily/weekly UNIX file system backup

SYNOPSIS
  /etc/filesave.?
  /etc/tapesave

DESCRIPTION
  These shell scripts are provided as models. They are designed to provide a
  simple, interactive operator environment for file backup. Filesave.? is for
daily disk-to-disk backup and tapesave is for weekly disk-to-tape.

  The suffix .? can be used to name another system where two (or more)
machines share disk drives (or tape drives) and one or the other of the sys-
tems is used to perform backup on both.

SEE ALSO
  shutdown(1M), volcopy(1M).
NAME
getty — set the modes of a terminal

SYNOPSIS
/etc/getty name type delay

DESCRIPTION
*Getty* is normally invoked by *init*(8) as the first step in allowing users to login to the system. Lines in /etc/inittab tell *init* to invoke *getty* with the proper arguments.

*Name* should be the name of a terminal in /dev (e.g., tty03); *type* should be a single character chosen from —, 0, 1, 2, 3, 4, 5, or 6 (may vary locally) which selects a speed table in *getty*, or !, which tells *getty* to update /etc/utmp and exit; *delay* is relevant for dial-up ports only. It specifies the time in seconds that should elapse before the port is disconnected if the user does not respond to the *login*: request.

First, *getty* types the *login*: message. The *login*: message depends on the speed table being used, and may include the characters that put the GE Terminet 300 terminal into full-duplex, take the DASI terminals out of the plot mode, or put a TELETYPE® Model 37 into full-duplex. Then the user’s login name is read, a character at a time.

While reading, *getty* tries to adapt to the terminal, speed, and mode that is being used. If a null character is received, it is assumed to be the result of a “break” (“interrupt”). The speed is then changed based on the speed table that *getty* is using, and *login*: is typed again. Subsequent breaks cause a cycling through the speeds in the speed table being used.

The user’s login name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately. If the login name contains only upper-case alphabetic characters, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, *login*(1) is called with the user’s login name as argument.

Speed sequences for the speed tables:

- B110; for 110 baud console TTY.
- B300—B150—B110—B1200; normal dial-up sequence starting at B300.
- B150; no sequence.
- B2400; no sequence.
- B1200—B300—B150—B110; normal dial-up sequence starting at B1200.
- B300; for console DECwriter.
- B9600; no sequence.
- B4800—B9600; for Tektronix 4014.

SEE ALSO
login(1), tty(4), inittab(5), utmp(5), init(8).

BUGS
Ideally, the speed tables would be read from a file, not compiled into *getty*. 
NAME

hasp — RJE (Remote Job Entry) to IBM

SYNOPSIS

/usr/hasp/haspinit
/usr/hasp/hasphalt

DESCRIPTION

*HASP* is the communal name for a collection of programs and a file organization that allow a UNIX system, equipped with an appropriate driver for the DQSII-B, to communicate with IBM's Job Entry Subsystems by mimicking an IBM 2770 remote station.

*HASP* is initiated by the command *haspinit* and is terminated gracefully by the command *hasphalt*. While active, *hasp* runs in background and requires no human supervision. It quietly transmits, to the IBM system, jobs that have been queued by the command *send*(1C) and messages that have been entered by the command *rjestat*(1C). It receives, from the IBM system, print and punch data sets and message output. It enters the data sets into the proper UNIX directory and notifies the appropriate user of their arrival. It scans the message output to maintain a record on each of its jobs. It also makes these messages available for public inspection, so that *rjestat*(1C), in particular, may extract responses.

Unless otherwise specified, all files and commands described below live in directory /usr/hasp (first exceptions: *send* and *rjestat*).

There are two sources of data that is to be transmitted by *hasp* from UNIX to an IBM System/370. In both cases, the data is organized as files in *pnch*(5) format. The first is a single file *haspmesg* that is reserved for message input. It is written by the enquiry command *rjestat*(1C) and is assigned a priority for transmission. The second source, containing the bulk of the data, consists of jobs that have been entered into the *xmit* queue by the program *haspqer*. On completion of processing, *send* invokes *haspqer*. As each file is queued, a subordinate *info/logx* file is created to save the name, user ID, login directory, and terminal ID of the user who is doing the queuing. Upon successful transmission of the data to the IBM system, *haspdisp* will move this information into the *jobsout* file and delete the *info/logx* file.

Each time *haspinit* is invoked, the *xmit* queue is compacted, along with the associated *info/logx* files, and its beginning and end are calculated. A three-digit sequence number specifying the first free slot at the end of the queue is written to file *haspstat*. This number is subsequently updated by *haspqer* each time that a new job is entered into the queue. A pointer to the beginning of the queue is maintained by *haspmain*. It is periodically compared to the current end of the queue to determine whether any jobs are waiting to be transmitted. A null lock-file *hasplock* is created with mode zero to prevent simultaneous updating of *haspstat*.

In anticipation of receiving output, *hasp* always maintains a vacant file *tmp* in its own directory. Output from the IBM system is initially written into this file and is classified as either a print data set, a punch data set, or message output. Print output is converted to an ASCII text file, with standard tabs. Form feeds are suppressed, but the last line of each page is distinguished by the presence of an extraneous trailing space. Punch output is converted to EBCDIC format. This classification and both conversions occur as the output is received; *tmp* files are moved or copied into the appropriate user's directory and assigned the name *prnt* or *pncb*, respectively, or placed into user directories under user-specified names, or used as input to
programs to be automatically executed, as specified by the user. This process is driven by the "usr=..." specification. Hasp retains ownership of these files and permits read-only access to them. Files of message output are digested by hasp immediately and are not retained.

A record is maintained for each job that passes through hasp. Identifying information is extracted contextually from files transmitted to and received from the IBM system. From each file transmitted, hasp extracts the job name, the programmer's name, the user name, the destination directory name, and the message level. This information is temporarily stored, in the order of submission of jobs, in file jobsout. It is retrieved, by job name and programmer's name, when the IBM system acknowledges the job and assigns a number to it.

The IBM system automatically returns an acknowledgement message for each job it receives. Other status messages are returned in response to enquiries entered by users and in response to enquiries that hasp itself generates every ten minutes. All messages received by hasp are appended to the resp file. The resp file is automatically truncated when it reaches 32,000 bytes. Each sequence of enquiries written to the message file haspmesg should be preceded by an identification card image of the form /<SUX><process id>. The IBM system will echo back the first portion of this card image, as this is an illegal command. The appearance of process ids in the response stream permits responses to be passed on to the proper users. Hasp enters process id zero on all enquiries it generates on its own behalf.

While it is active, hasp occupies at least the two process slots that are appropriated by haspinit. These slots are used to run haspmain, that supervises data transfers, as well as haspdisp, that performs dispatching functions; these two processes are connected by a pipe. The function of haspmain is to cycle repetitively, looking for data to transfer either to or from the IBM system. When it finds some, it spawns a child process, either haspxmit or hasprecv, to effect the transfer. It waits for its child to complete its task and then passes an event notice to haspdisp. Haspmain exits normally as soon as it detects the file haspstop (created by haspinit), and exits reluctantly whenever it encounters a run of errors. An attempt is made to manage the null file haspdead so that it exists precisely when haspmain is not executing. Haspinit has the capability of dialing any remote IBM system with the proper hardware and software configuration. A file haspoff is created by haspinit to signal that the phone should be hung up by haspmain.

Ordinarily, haspdisp waits for event completion notices from haspmain. Haspdisp follows up the events described by directing output files, updating records, and notifying users. It may spawn the program haspcopy to copy output across file systems. Haspdisp references the system files /etc/passwd and /etc/utmp to correlate user names, numeric ids, and terminals. Normal termination of haspmain causes haspdisp to exit also. In the case of error termination, haspdisp delays about one minute and then reboots RJE by executing haspinit again.

Event notices begin with a one-digit code. The code "0" alone signals normal termination. Other event notices consist of a code in the range 1 to 6 followed by the name of a file in the /usr/hasp directory. Notices are issued as each file in the xmit queue is transmitted and as each tmp file is filled with output. These files are moved to new temporary names before the event notice is composed. Transmitted files (code 1) are renamed zmite and output files (codes 3-5) are renamed prte, pche, or msge,
depending on their type. When haspdisp gets around to following up on the events described, the files will either be deleted or moved to a permanent destination.

Event notices are written to the log file at the time they are received by haspdisp. A typical section of the log looks as follows:

```
1zmit283
5msg61
1zmit284
5msg62
3prt63
```

Additional lines are written to the log by haspinit. Each reboot of haspinit is marked by a time stamp. If the previous execution of haspmain ended in error, an exception notice precedes the time stamp. Exception notices are formatted by haspmain and consist of a sequence of capital letters. The most common is AAAAA, that indicates five successive failures to acquire the line for a transmission to the host. A sequence of time stamps alternating with AAAAA indicates that the host is not responding to RJE. Each time the RJE facility is booted via the haspinit program, the log file is cleaned out. A copy of its last contents is placed in a file named slog.

Most hasp files and directories are protected from unauthorized tampering. The exception is the pool directory, that is provided so that send(1C) can create temporary files in the correct file system. Hasqpgr and rjestat(1C), the user’s interfaces to hasp, operate in setuid mode to contribute the necessary permission modes. Rjestat(1C), incidentally, extends to anyone who can login as rje complete freedom to enter console commands. When invoked with a + argument, it suppresses the d that begins a display command and allows one to cancel or re-route jobs.

Some minimal oversight of each hasp subsystem is required. The hasp mailbox should be inspected and cleaned out periodically. The job directory should also be checked. The only files placed there are output files whose destination file systems are out of space. Users should be given a short period of time (say, a day or two), and then these files should be removed.

Usage statistics are recorded in the directory /usr/hasp/usage, if it exists. Six files will be created and updated. Each will contain data on a per-user ID basis. File hasp.in.sum accumulates the number of blocks transmitted by hasp; file hasp.in.cnt records the number of transmissions; file hasp.in.max records the size, in blocks, of the largest job sent. Files hasp.out.sum, hasp.out.cnt and hasp.out.max contain the same statistics for output received by hasp. The program usage may be used to print these statistics; “usage file [user ID1 ...]” will print out the statistics gathered in file. If the optional user ID list is present, only the statistics for these user IDs will be printed.

The configuration table /usr/rje/lines is accessed by all components of RJE. Its six columns may be labeled “host”, “system”, “directory”, “prefix”, “device”, and “parameters”. Each line of the table maximum of eight) defines an RJE connection. “Host” is the name of a remote computer: A, B, C, U2, or U3. “System” is a string of capital letters identifying UNIX systems. The first specifies where the RJE connection is normally terminated; the remainder specify where it may be backed-up to if the primary RJE system goes down. “Directory” is the directory name of the servicing RJE subsystem. “Prefix” is the string prefixed (redundantly) to several crucial files and programs in the directory: hasp, hasp2, uvac. “Device” is the name of the controlling DQS-11B, with /dev/ excised. “Parameters”
contains information on the type of connection to make. Each subfield is separated by the delimiter :. Any or all fields may be omitted; however, the fields are positional. All but trailing delimiters must be present. For example, in

1200:512:::9-555-1212

subfields 3, 4, and 5 are missing, but the delimiters are present.

The first subfield specifies the amount of space (S) in blocks that RJE tries to maintain on file systems it touches. The default is 0 blocks. Several RJE programs, including the send(1C) command, use the ustat(2) system call to determine the remaining capacity of the file systems they use. Send shuts down and haspinit issues a warning when no more than 1.5S blocks are available; haspmain stops accepting output from the host when the capacity falls to 1.2S blocks; RJE becomes dormant, until conditions improve, when the capacity falls to S blocks. If the space on the file system specified by the user on the “usr=” card would be depleted to a point below S, the file will be put in the “job” subdirectory of the connection’s home directory (e.g., /usr/hasp2/job), rather than in the place that the user requested.

The second subfield specifies the size in blocks of the largest file that can be accepted from the host without truncation. The default is no truncation.

The third subfield specifies burst page removal. If this subfield contains the letter y, RJE will not try to remove any burst pages from returned output. Any other value in this subfield will cause RJE to scan for and remove the leading burst pages. For UNIVAC hosts this flag is inoperative and no burst pages are ever removed. Embedded and trailing burst pages are never removed.

The fourth subfield specifies what to do with undeliverable returning jobs. If an output file is undeliverable for any reason other than file system space limitations (e.g., missing or invalid “usr=” card) and this subfield contains the letter y, the output will be retained in the “job” subdirectory of the home directory (e.g., /usr/hasp/job). If this subfield has any other value, undeliverable output will be discarded. The default is n. The fifth subfield specifies the status of the interactive status terminal for this line. If the subfield contains an i, all console status facilities are inhibited (e.g., rjestat(1C) will not behave like a status terminal, and the ten-minute automatic status inquiry is inhibited). This subfield must contain an i for UNIVAC configurations.

Subfield six contains a telephone number to be used to call a host machine. The telephone number may contain the digits 0 thru 9 and the character - which denotes a pause. If the telephone number is not present, no dialing is attempted and a leased line is assumed.

Sign-on is controlled by the existence of a signon file in the controlling directory (e.g., /usr/hasp/signon). If this file is present its contents are sent as a sign-on message to the host system.

The file /usr/rje/sys contains the single-letter name of the current UNIX system. An RJE connection will be considered available if this is its primary system or if this is one of its backup systems and the associated directory is mounted. Send(1C) and rjestat(1C) select an available connection by indexing on the “host” field of the configuration table. Hasp programs index on the “prefix” field. A subordinate directory, sque, exists in /usr/rje for use by haspdisp and shqer programs. This directory holds those output files that have been designated as standard input to some executable file. This designation is done via the “usr=...” specification. Haspdisp places the output files here and updates the file log to specify the order of execution, arguments to be passed, etc. Shqer executes the
appropriate files. The *shqer* must be started in */etc/rc*. A program called *compact* compacts the *log* file. It should be executed before *shqer* and RJE have been started.

All HASP programs are reentrant; therefore, if more than one HASP is to be run on a given UNIX system, simply link (via *ln*(1)) HASP2 program names to HASP names in */usr*.

**FILES**

Configuration-dependent and general-purpose RJE files:

```
/dev/rjei          DQS11-B
/dev/tty?         terminals
/etc/utmp         list of active users
/etc/passwd       user population
/usr/rje/sys      UNIX system name, e.g., "A"
/usr/rje/lines    UNIX RJE lines configuration table
/usr/rje/sque/log log information for *shqer*
```

User files:

```
/usr/mail/*      a user's mailbox
* /print*        a user's print data set
* /punch*        a user's punch data set
```

*Hasp* files (relative to the *directory* entry in the RJE configuration table):

```
hasp*      mostly programs
haspdead    inactive flag
haspsoff    dial-up hang-up signal
haspsstop   halt signal
haspmesg    message slot
haspstat    queue end record
hasplock    lockout file
xmit*      jobs queued
info/logx*  haspqer loginfo
job/*       output from jobs whose file systems are out of space
jobsout     fifo job store
tmp*        output files
log         event log
resp        concatenated responses from the IBM system
status      RJE message of the day
pool/stm*   *send*(1C) temporaries
usg/*       usage statistics
signon      contains card image for signon
```

**SEE ALSO**

`rjestat(1C), send(1C), dqs(4), pnch(5), mk(8)`.


**DIAGNOSTICS**

*Haspinit* provides brief error messages describing obstacles to bringing up *hasp*. They can best be understood in the context of the RJE source code. The most frequently occurring one is “cannot open */dev/rjei*”. This may occur if the DQS-11B status register shows something other than READY (octal 200). It will also occur if another process already has the DQS-11B open, or if the exclusive use flag (_dqsx+3, _dqsx+73, etc.) has remained set after a close of the DQS-11B.
Once *hasp* has been started, users should assist in monitoring its performance, and should notify operations personnel of any perceived need for remedial action. *Rjestar(1C)* will aid in diagnosing the current state of RJE. It can detect, with some reliability, when the far end of the communications line has gone dead, and will report in this case that the host computer is not responding to RJE. It will also attempt to reboot *hasp* if it detects a prolonged period of inactivity on the DQS-11B.

**BUGS**

The name *hasp* is an anachronism. It is used only as a collective name and could represent HASP, JES2, ASP, etc.
NAME
init — process control initialization

SYNOPSIS
/etc/init [ state ]

DESCRIPTION
init is invoked inside UNIX as the last step in the boot procedure. It is pro-
cess number one, and is the ancestor of every other process in the system.
As such, it can be used to control the process structure of the system. If
init is invoked with an argument by the super-user, it will cause a change in
state of process one.

init has 9 states, 1 through 9; it is invoked by the system in state 1, and it
performs the same functions on entering each state. When a state is
entered, init reads the file /etc/inittab. Lines in this file have the format:

state:id:flags:command

All lines in which the state field matches init’s current state are recognized.
If a process is active under the same two character id as a recognized line,
it may be terminated (signal 15), killed (signal 9), or both by including the
flags t and k in the order desired. The signal is sent to all processes in the
process group associated with the id. The command field is saved for later
execution.

After reading /etc/inittab and signaling running processes as required, but
before invoking any processes under the new state, /etc/rc is invoked with
three arguments. This command file performs housekeeping such as remo-
ving temporary files, mounting file systems, and starting daemons. The
three arguments are the current state, the number of times this state has
been entered previously, and the prior state. Init will also execute /etc/rc
at the request of the operating system (e.g., when recovering from power
failure). In this last case, the first argument has an x appended to it.

When /etc/rc has finished executing, init invokes all commands waiting to
be executed. (A command is waiting to be executed if there is no process
currently running that has the same id as the command.) The flag c (con-
tinuous) requires the command to be continuously reinvoked whenever the
process with that id dies. The flag o (off) causes the command to be
ignored. This is useful for turning lines off without extensive editing. Oth-
erwise, the command is invoked a maximum of one time in the current
state.

Init invokes the command field read from /etc/inittab by opening / for
reading and writing on file descriptors 0, 1, and 2, resetting all signals to
system default, setting up a new process group (setpgrp(2)), and execing:

/bin/sh -c exec command

DIAGNOSTICS
When init can do nothing else because of a missing /etc/inittab or when it
has no children left, it will try to execute a shell on /dev/console. When
the problem has been fixed, it is necessary to change states, and terminate
the shell.

BUGS
init does not complain if the state—id pairs in /etc/inittab are not unique.
For any given pair, the last one in the file is valid.

FILES
/etc/inittab
/etc/rc
INIT(8)

/bin/sh
/dev/console

SEE ALSO
login(1), sh(1), exec(2), setpgroup(2), inittab(5), getty(8).
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 lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the output key.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 output key bits in the result.

Makekey is intended for programs that perform encryption (e.g., ed(1) and crypt(1)). Usually, its input and output will be pipes.

SEE ALSO
crypt(1), ed(1), passwd(5).
NAME
mk — how to remake the system and commands

DESCRIPTION
All source for UNIX is in a source tree distributed in the directory /usr/src. This includes source for the operating system, libraries, commands, miscellaneous files necessary to the running system, and procedures to create everything from this source.

The top level consists of the directories cmd, lib, uts, head, and stand as well as commands to remake each of these "directories". These commands are named :mk, which remakes everything, and :mkdir where dir is the directory to be recreated. Each recreation command will make all or part of the piece; over which it has control. :mk will run each of these commands and thus recreate the whole system.

The lib directory contains libraries used when loading user programs. The largest and most important of these is the C library. All libraries are in sub-directories and are created by a makefile or runcom. A runcom is a Shell command procedure used specifically to remake a piece of the system. :mklib will rebuild the libraries that are given as arguments. The argument \* will cause it to remake all libraries.

The head directory contains the header files, usually found in /usr/include on the running system. :mkhead will install those header files that are given as arguments. The argument \* will cause it to install all header files.

The uts directory contains the source for the UNIX operating system. :mkuts (no arguments) invokes a series of makefiles that will recreate the operating system.

The stand directory contains stand-alone commands and boot programs. :mkstand will rebuild and install these programs.

The cmd directory contains files and directories. :mkcmd transforms source into a command based upon its suffix (.1, .y, .c, .s, .sh), or its makefile (see make(1)) or runcom. A directory is assumed to have a makefile or a runcom that will take care of creating everything associated with that directory and its sub-directories. Makefiles and runcoms are named command.mk and command.rc respectively.

:mkcmd will recreate commands based upon a makefile or runcom if one of them exists; alternatively commands are recreated in a standard way based on the suffix of the source file. All commands requiring more than one file of source are grouped in sub-directories, and must have a makefile or a runcom. C programs (.c) are compiled by the C compiler and loaded stripped with shared text. Assembly language programs (.s) are assembled with /usr/include/sys.s which contains the system call definitions. Yacc programs (.y) and lex programs (.l) are processed by yacc(1) and lex(1) respectively before C compilation. Shell programs (.sh) are copied to create the command. Each of these operations leaves a command in ./cmd which is then installed by using /etc/install.

The arguments to :mkcmd are either command names, or subsystem names. The subsystems distributed with UNIX are: acct, graf, rje, sces, and text. Prefacing the :mkcmd instruction with an assignment to the Shell variable SARGS will cause the indicated components of the subsystem to be rebuilt.

The entire sces subsystem can be rebuilt by:

/usr/src/:mkcmd sces
while the delta component of sees can be rebuilt by:

ARGS="delta" /usr/src/:mkcmd sees

The log command, which is a part of the stat package, which is itself a part of the graf package, can be rebuilt by:

ARGS="stat log" /usr/src/:mkcmd graf

The argument \* will cause all commands and subsystems to be rebuilt.

Makefiles, both in ./emd and in subdirectories, have a standard format. In particular :mkcmd depends on there being entries for install and clobber. Install should cause everything over which the makefile has jurisdiction to be made and installed by /etc/install. Clobber should cause a complete cleanup of all unnecessary files resulting from the previous invocation.

Most of the runcoms in ./cmd (as opposed to subdirectories) relate in particular to a need for separated instruction and data (I and D) space.

In the past, dependency on the C library routine ctime(3C) was also important. Ctime had to be modified for all systems located outside of the eastern time zone, and all commands that referenced it had to be recompiled. Ctime has been rewritten to check the environment (see environ(7)) for the time zone. This results in time zone conversions possible on a per-process basis. /etc/profile sets the initial environment for each user, and /etc/rc sets it for certain system daemons. These two programs are the only ones which must be modified outside of the eastern time zone.

An effort has been made to separate the creation of a command from source, and its installation on the running system. The command /etc/install is used by :mkcmd and most makefiles to install commands in the proper place on the running system. The use of install allows maximum flexibility in the administration of the system. Install makes very few assumptions about where a command is located, who owns it, and what modes are in effect. All assumptions may be overridden on invocation of the command, or more permanently by redefining a few variables in install. The object is to install a new version of a command in the same place, with the same attributes as the prior version.

In addition, the use of a separate command to perform installation allows for the creation of test systems in other than standard places, easy movement of commands to balance load, and independent maintenance of makefiles. The minimization of makefiles in most cases, and the site independence of the others should greatly reduce the necessary maintenance, and allow makefiles to be considered part of the standard source.

SEE ALSO
install(1M), make(1).
NAME
rc — system initialization shell script

SYNOPSIS
/etc/rc

DESCRIPTION
The /etc/rc file is executed by init(8) whenever the init state is changed.

SEE ALSO
init(8).
NAME
rje — RJE (Remote Job Entry) to IBM

SYNOPSIS
/usr/rje/rjeinit
/usr/rje/rjehalt

DESCRIPTION
RJE is the communal name for a collection of programs and a file organization that allows a UNIX system, equipped with a KMC11-B, KMC11 driver, and associated Virtual Protocol Machine (VPM) software, to communicate with IBM's Job Entry Subsystems by mimicking an IBM 360 remote multitasking work station.

Implementation.
RJE is initiated by the command rjeinit and is terminated gracefully by the command rjehalt. While active, RJE runs in the background and requires no human supervision. It quietly transmits, to the IBM system, jobs that have been queued by the send(1C) command, and operator requests that have been entered by the rjestat(1C) command. It receives, from the IBM system, print and punch data sets and message output. It enters the data sets into the proper UNIX directory and notifies the appropriate user of their arrival. It scans the message output to maintain a record on each of its jobs. It also makes these messages available for public inspection, so that rjestat(1C), in particular, may extract responses.

Unless otherwise specified, all files and commands described below reside in directory /usr/rje (first exceptions: send and rjestat).

There are two sources of data to be transmitted by RJE from UNIX to an IBM System/370. In both cases, the data is organized as files in the /usr/rje/squeue directory. The first are files named CO* which are created by the enquiry command rjestat(1C). The second source, containing the bulk of the data, are files named rd* or sq* which have been created by send and queued, by the program rjeqer. On completion of processing send invokes rjeqer. Rjeqer and rjestat inform the program rjexmit that a file has been queued via the file joblog. Upon successful transmission of the data to the IBM machine, rjexmit removes the queued file. As files are transmitted and received, the program rjedisp writes an entry containing the date, time, file name, logname, and number of records in the file acctlog, if it exists. This file can be used for local logging or accounting information, but is not used elsewhere by RJE. The use of this information is up to the RJE administrator.

Each time rjeinit is invoked, the joblog file is truncated and recreated from the contents of the /usr/rje/squeue directory. During this time, rjeinit prevents simultaneous updating of the joblog file.

Output from the IBM system is classified as either a print data set, a punch data set, or message output. Print output is converted to an ASCII text file, with standard tabs. Form feeds are suppressed, but the last line of each page is distinguished by the presence of an extraneous trailing space. Punch output is converted to pncch(5) format. This classification and both conversions occur as the output is received. Files are moved or copied into the appropriate user's directory and assigned the name prnt* or pncch*, respectively, or placed into user directories under user-specified names, or used as input to programs to be automatically executed, as specified by the user. This process is driven by the "usr==..." specification. RJE retains ownership of these files and permits read-only access to them. Message output is digested by RJE immediately and is not retained.
A record is maintained for each job that passes through RJE. Identifying information is extracted contextually from files transmitted to and received from the IBM system. This information is stored and used by the \texttt{rjedisp} program for IBM job acknowledgements and delivery of output files.

The IBM system automatically returns an acknowledgement message for each job it receives. Other status messages are returned in response to enquiries entered by users. All messages received by RJE are appended to the \texttt{resp} file. The \texttt{resp} file is automatically truncated when it reaches 70,000 bytes. Each enquiry is preceded and followed by an identification card image of the form "$UX<process id>$". The IBM system will echo this back as an illegal command. The appearance of process ids in the response stream permits responses to be passed on to the proper users.

While it is active, RJE occupies at least the three process slots that are appropriated by \texttt{rjeinit}. These slots are used to run \texttt{rjexmit}, the transmitter, \texttt{rjerecv}, the receiver, and \texttt{rjedisp}, the dispatcher. These three processes are connected by pipes. The function of each is as follows:

\texttt{rjexmit} Cycles repetitively, looking for data to transmit to the IBM system. After transmission, \texttt{rjexmit} passes an event notice to \texttt{rjedisp}. If \texttt{rjexmit} encounters a stop file, (created by \texttt{rjehalt}), it exits normally. In the case of error termination, \texttt{rjexmit} reboots RJE by executing \texttt{rjeinit}.

\texttt{rjerecv} Cycles repetitively, looking for data returning from the IBM machine. Upon receipt of data, \texttt{rjerecv} notifies either \texttt{rjexmit} or \texttt{rjedisp} of the event (transfer information is sometimes passed to \texttt{rjexmit}). \texttt{RJerecv} exits normally at the first appropriate moment when it encounters the file stop, or exits reluctantly when it encounters a run of errors.

\texttt{rjedisp} Follows up event notices by directing output files, updating records, and notifying users. \texttt{Rjedisp} references the system files \texttt{/etc/passwd} and \texttt{/etc/atmp} to correlate user names, numeric ids, and terminals. Termination of \texttt{rjerecv} causes \texttt{rjedisp} to exit also.

\texttt{Rjeinit} has the capability of dialing any remote IBM system with the proper hardware and software configuration.

Most RJE files and directories are protected from unauthorized tampering. The exception is the spool directory. It is used by \texttt{send(1C)} to create temporary files in the correct file system. \texttt{Rjeqer} and \texttt{rjestat(1C)}, the user's interfaces to RJE, operate in setuid mode to contribute the necessary permission modes.

\textbf{Administration.}

Some minimal oversight of each RJE subsystem is required. The RJE mailbox should be inspected and cleaned out periodically. The job directory should also be checked. The only files placed there are output files whose destination file systems are out of space. Users should be given a short period of time (say, a day or two), and then these files should be removed.

The configuration table \texttt{/usr/rje/lines} is accessed by all components of RJE. Each line of the table (maximum of 8) defines an RJE connection. Its seven columns may be labeled host, system, directory, prefix, device, peripherals and parameters. These columns are described as follows:

\texttt{host}

The name of a remote IBM computer (e.g., A B C). This string can be up to 5 characters.
The name of a UNIX system. This name should be the same as the system name from `uname(1)`.

directory
This is the directory name of the servicing RJE subsystem (e.g., `/usr/rje1`).

prefix
This is the string prefixed (redundantly) to several crucial files and programs in directory (e.g., `rje1`, `rje2`, `rje3`).

device
This is the name of the controlling VPM device, with `/dev/` excised.

peripherals
This field contains information on the logical devices (readers, printers, punches) used by RJE. Each subfield is separated by `:` and is described as follows:

1. Number of logical readers.
2. Number of logical printers.
3. Number of logical punches.

Note: the number of peripherals specified for an RJE subsystem must agree with the number of peripherals which have been described on the remote machine for that line.

parameters
This field contains information on the type of connection to make. Each subfield is separated by `:`. Any or all fields may be omitted; however, the fields are positional. All but trailing delimiters must be present. For example, in

```
1200:512::9-555-1212
```

subfields 3 and 4 are missing, but the delimiters are present. Each subfield is defined as follows:

1. space
   This subfield specifies the amount of space (`S`) in blocks that RJE tries to maintain on file systems it touches. The default is 0 blocks. `Send` will not submit jobs and `rjeinit` issues a warning when less than 1.5S blocks are available; `rjerecv` stops accepting output from the host when the capacity falls to S blocks; RJE becomes dormant, until conditions improve. If the space on the file system specified by the user on the `"usr="` card would be depleted to a point below S, the file will be put in the `job` subdirectory of the connection's home directory, rather than in the place that the user requested.

2. size
   This subfield specifies the size in blocks of the largest file that can be accepted from the host without truncation taking place. The default is no truncation.

3. badjobs
   This subfield specifies what to do with undeliverable returning jobs. If an output file is undeliverable for any reason other than file system space limitations (e.g., missing or invalid `"usr="` card) and this subfield contains the letter `y`, the output will be retained in the `job` subdirectory of the
home directory, and login rje is notified. If this subfield contains an n or has any other value, undeliverable output will be discarded. The default is n.

(4) console
This subfield specifies the status of the interactive status terminal for this line. If the subfield contains an i, all console status facilities are inhibited (e.g., rjestat(1C) will not behave like a status terminal). In all cases, the normal non-interactive uses of rjestat(1C) will continue to function. The default is y.

(5) dial-up
This subfield contains a telephone number to be used to call a host machine. The telephone number may contain the digits 0 thru 9 and the character — which denotes a pause. If the telephone number is not present, no dialing is attempted and a leased line is assumed.

Sign-on is controlled by the existence of a signon file in the home directory. If this file is present, its contents are sent as a sign-on message to the host system. If this file does not exist, a blank card is sent. Sign-off is controlled in the same way, except that the signoff file is sent by rjehalt if it exists. If the signoff file does not exist, a "/osiIDOft'' card is sent. These files should be ASCII text and no more than 80 characters.

Send(1C) and rjestat(1C) select an available connection by indexing on the host field of the configuration table. RJE programs index on the prefix field. A subordinate directory, sqwe, exists in /usr/rje for use by rjedis p and shqer programs. This directory holds those output files that have been designated as standard input to some executable file. This designation is done via the "usr..." specification. Rjedis p places the output files here and updates the file log to specify the order of execution, arguments to be passed, etc. Shqer executes the appropriate files.

All RJE programs are shared text; therefore, if more than one RJE is to be run on a given UNIX system, simply link (via ln(1)) RJE2 program names to RJE names in /usr.

SEE ALSO
rjestat(1C), send(1C), vpm(4), pnch(5), mk(8).
Setting Up UNIX.

DIAGNOSTICS
Rjeinit provides brief error messages describing obstacles encountered while bringing up RJE. They can best be understood in the context of the RJE source code. The most frequently occurring one is "cannot open /dev/vpm?". This may occur if the VPM script has not been started, or if another process already has the VPM device open.

Once RJE has been started, users should assist in monitoring its performance, and should notify operations personnel of any perceived need for remedial action. Rjestat(1C) will aid in diagnosing the current state of RJE. It can detect, with some reliability, when the far end of the communications line has gone dead, and will report in this case that the host computer is not responding to RJE. It will also attempt to reboot RJE if it detects a prolonged period of inactivity on the KMC-11B.
NAME
romboot — special ROM bootstrap loaders

DESCRIPTION
To bootstrap programs from various storage media, standard DEC ROM bootstrap loaders are often used. However, such standard loaders may not be compatible with UNIX bootstrap programs or may not exist on a particular system. Thus, special bootstrap loaders were designed that may be cut into a programmable ROM (M792 read-only-memory) or manually toggled into memory.

Each program is position-independent, that is, it may be located anywhere in memory. Normally, it is loaded into high core to avoid being overwritten. Each reads one block from drive 0 into memory starting at address 0 and then jumps to address 0. To minimize the size, each assumes that a system INIT was generated prior to execution. Also, the address of one of the device registers is used to set the byte count register or word count register. In each case, this will read in at least 256 words, which is the maximum size of bootstrap programs.

On disk devices, block 0 is read; on tape devices, one block from the current position. Thus, the tape should be positioned at the load point (endzone if DECtape) prior to booting. Also, the standard DEC bootstrap loader for magnetic tape may be emulated by positioning the tape at the load point and executing the bootstrap loader twice.

By convention, on PDP 11/45 systems, address 773 000 is the start of a tape bootstrap loader, and 773 020 the start of a disk bootstrap loader. The actual loaders used depend on the particular hardware configuration.

SEE ALSO
70boot(8), unixboot(8).

CODE
TC11 — DECtape
012700  mov $tcbar0
177346
010040  mov r0,−(r0) /use tc addr for wc
012740  mov $3,−(r0) /read bn forward
000003
105710  l: tstb (r0) /wait for ready
002376  bge 1b
112710  movb $5,(r0) /read forward
000005
105710  l: tstb (r0) /wait for ready
002376  bge 1b
005007  clr pc /transfer to zero

TU10 — Magnetic Tape
012700  mov $mtcma,r0
172526
010040  mov r0,−(r0) /use mt addr for bc
012740  mov $60003,−(r0) /read, 800 bpi, 9 track
060003
105710  l: tstb (r0) /wait for ready
002376  bge 1b
005007  clr pc /transfer to zero
ROM BOOT(8) (PDP-11 only)  ROM BOOT(8)

TU16 — Magnetic Tape

012700 mov $mtwc, r0
172442
012760 mov $1300,30(r0) /set 800 bpi, PDP format
001300
000030
010010 mov r0,(r0) /use mt addr for wc
012740 mov $71, -(r0) /read
000071
105710 1: tstb (r0) /wait for ready
002376 bge 1b
005007 clr pc /transfer to zero

RK05 — Disk Pack

012700 mov $rkda, r0
177412
005040 clr -(r0)
010040 mov r0, -(r0) /use rk addr for wc
012740 mov $5, -(r0) /read
000005
105710 1: tstb (r0) /wait for ready
002376 bge 1b
005007 clr pc /transfer to zero

RP03 — Disk Pack

012700 mov $rpmr, r0
176726
005040 clr -(r0)
005040 clr -(r0)
005040 clr -(r0)
010040 mov r0, -(r0) /use rp addr for wc
012740 mov $5, -(r0) /read
000005
105710 1: tstb (r0) /wait for ready
002376 bge 1b
005007 clr pc /transfer to zero

RP04 — Disk Pack

012700 mov $rpcs1, r0
176700
012720 mov $21,(r0)+ /read—in preset
000021
012760 mov $10000,30(r0) /set to 16—bits/word
010000
000030
010010 mov r0,(r0) /use rp addr for wc
012740 mov $71, -(r0) /read
000071
105710 1: tstb (r0) /wait for ready
002376 bge 1b
005007 clr pc /transfer to zero
NAME
   rp6fmt — format and/or check RP06 disk packs

DESCRIPTION
   rp6fmt will format new RP06 packs and check used packs (with write inhibited). The program reports the location and type of errors encountered, including ECC correctable error burst sizes.

EXECUTION
   The following example shows how to load rp6fmt on a VAX-11/780 with a UNIX 3.0 updated floppy disc:

   >>> H<cr>
   HALTED AT nnnnnnnn
   >>> B<cr>
   CPU HALTED
   INIT SEQ DONE
   HALT INST EXECUTED
   HALTED AT nnnnnnnn
   LOAD DONE, nnnnnnnn BYTES LOADED

   To execute rp6fmt, type /stand/rp6fmt after the standalone shell prompt. The formatter will print out its command vocabulary, and proceed inter-actively. If one wishes to format a pack on disk drive 1, for example, the command is df1. The program will double check format requests, as pack contents will be destroyed.

COMMANDS
   m n           MBA with drive doing the format is n. (defaults to 0)
   d n           drive with the pack to be formatted or checked is n. (drive number must be between 1 and 7)
   f             format pack
   c             check pack format
   q             quit
   v             print vocabulary
   R n           set the error report level to n.
   X             will tell you about the available report levels.

   The X command will explain the Report Level options the first time it is executed. Subsequent execution by the operator or by the program during error logging, will merely print the information defined by the current report level.

FILES
   /stand/rp6fmt

SEE ALSO
   vaxops(8).
NAME
sar — system activity report package

DESCRIPTION
Sar is the first (tentative) piece of an overall UNIX measurement and statistics package; the data that are collected and the output formats are not yet final.

The operating system contains a number of counters that are incremented as various system actions occur. These include several time counters (that are incremented each 60th of a second depending on the CPU mode), I/O activity counters, switching and system-call counters, and file-access counters. The system activity package writes system activity parameters periodically on a binary file. It also generates a daily system activity report that covers the prime period (from 8:00 to 18:00).

The data collection and report generation are controlled by entries in crontab (see cron(1M)). The data collection program is normally activated every hour on the hour; the report generation once a day.

Every time the system is booted, a special record is written to the daily data file, since all the system activity counters restart from zero at that time. This process is done while executing /etc/rc (see init(8)) during UNIX initialization. It produces an entry on the daily report showing the restart time.

The daily reports are deposited in /usr/adm/sa/sar where dd are digits representing the day of the month. A report can be printed (e.g., cat /usr/adm/sa/sar05) any time before it is removed the following week.

The structure of the binary daily data file is:

```
struct sa {
    struct sysinfo si; /* defined in /usr/include/sys/sysinfo.h */
    long d0; /* number of reads and writes of disk 0 */
    long d1; /* number of reads and writes of disk 1 */
    long d2; /* number of reads and writes of disk 2 */
    long ts; /* time stamp in time_t format */
};
```

FILES
/usr/adm/sa/sadd daily data file
/usr/adm/sa/sar dd daily report file
/tmp/sa.adrfl address file
TAPE BOOT(8) (PDP-11 only) TAPE BOOT(8)

NAME

tapeboot — magnetic tape bootstrap program

DESCRIPTION

Tapeboot handles the problem of booting a PDP-11/45 or PDP-11/70 from a TU10 or TU16 tape transport. In both cases, the tape density is 800 bpi. The complete program fits in one 512 byte block, but is duplicated so that one copy resides in block 0 and another in block 1. Thus, both the standard DEC ROM bootstrap loaders and the special ROM loaders will work. For example, to create a boot tape, execute:

```
    cat /stand/tapeboot program-to-boot >/dev/mt0
```

To boot from magnetic tape, read the first record of the tape into memory starting at address 0 and then jump to address 0, using a special ROM or some manual procedure (toggle in the program). The bootstrap program relocates itself to high core as specified when assembled (typically 24K words, maximum of 28K). It then determines whether to use the TU10 code or the TU16 code. The TU10 is used if the TM11 command register (772 522) exists and the function (bits <3:1>) is non-zero, otherwise the TU16 is used. It then types on the console UNIX tape boot loader, rewinds the tape, reads two blocks to skip past itself on the tape, clears memory, and reads the rest of the tape, to the tape mark, into memory starting at address 0. If address 0 contains 000 407, a UNIX a.out program is assumed and the first 8 words are stripped off by relocating the loaded program toward address 0. Finally, a jump to address 0 is done by executing jsr pc,$0.

If there is an error while reading the tape, the bootstrap program will type tape error and attempt to read the record again.

FILES

/stand/tapeboot TU10/TU16 magtape bootstrap
/usr/src/stand source directory

SEE ALSO

unixboot(8).
NAME
unixboot — UNIX startup and boot procedures

DESCRIPTION
How to start UNIX. UNIX is started by placing it in core at location zero and transferring to zero. Since the system is not reenterable, it is necessary to read it in from disk or tape. See diskboot(8) or tapeboot(8).

The switches. On systems with console switches, the switches are examined 60 times per second, and the contents of the address specified by the switches are displayed in the display register. If the switch address is even, the address is interpreted in kernel (system) space; if odd, the rounded-down address is interpreted in the current user space.

Init. The operating system invokes init(8) as process number 1. It comes up in state one which is conventionally single-user.

FILES
/ unix UNIX code

SEE ALSO
70boot(8), diskboot(8), init(8), romboot(8), tapeboot(8).
NAME
uvac — RJE (Remote Job Entry) to UNIVAC

SYNOPSIS
/usr/uvac/uvacinit
/usr/uvac/uvachalt

DESCRIPTION

Uvac is the communal name for a collection of programs and a file organization that allow a UNIX System, equipped with an appropriate driver for the DQS11-A, to communicate with a UNIVAC 1100 Series processor. This facility includes code that must run on the UNIVAC processor, under any Level 32 (or later) UNIVAC 1100 Executive that supports the Remote Symbiont Interface (RSI).

Uvac is initiated by the command uvacinit and is terminated gracefully by the command uvachalt. While active, uvac runs in background and requires no human supervision. It quietly transmits to the UNIVAC system jobs that have been queued by the command send(1C). It receives from the UNIVAC system print data sets. It enters the data sets into the proper UNIX directory and notifies the appropriate user of their arrival.

Other than name changes (uvac in place of hasp), non-existence of transparent mode (no punch files), non-existence of interactive rjestat(1C) capabilities, and use of ASCII format in place of EBCDIC format, hasp(8) should be referenced for information on this facility.
NAME

vaxops — VAX-11/780 console operations

DESCRIPTION

The procedures described here include the major operational sequences involved in running UNIX on the VAX-11/780 system. The following notation is used:

1. Special characters are enclosed in <> (e.g., <ctl> represents the "control" key, and <cr> stands for the "carriage return" key).

2. Items within {}s are mandatory substitutions.

DAILY PROCEDURES

Disk Boot

This procedure can be used only on a system with a floppy disk updated for use with UNIX. If the floppy disk has not been so updated, the sequences shown below under UNIX Floppy Update must be performed.

When the system is first turned on, the console prompt >>> is printed. If UNIX has been shut down, but not halted (see Bringing the System Down), the operator must type <ctl>p to get into console mode. After the prompt, type H<cr> to halt the system.

With the system halted, any of the console commands may be executed as described below under Console Operation.

To boot the stand-alone shell (sash) the operator types B<cr>. The following is an example of this operation as seen on the console, picking up after the <ctl>p:

```plaintext
>>>H<cr>
HALTED AT nnnnnnnnn

>>>B<cr>
CPU HALTED
INIT SEQ DONE
HALT INST EXECUTED
HALTED AT nnnnnnnnn
LOAD DONE, nnnnnnnn BYTES LOADED
```

The $S prompt indicates that the stand-alone shell (sash) is ready to accept commands. If it is desired to run stand-alone fsck(1M) (or other stand-alone functions), this is the time to do it. The commands have the form /stand/program where program can be any name from a limited list of UNIX commands found in the directory /stand. To perform a file system consistency check, type:

```plaintext
$S /stand/fsck /dev/rp0
```

To bring up UNIX, the operator must type unix<cr>. The system will come up through init 1 (see init(8)).

This is the appropriate time to do file system backups, and fsck(1M) should be executed if it was not executed in the stand-alone section of the boot. One must never operate the system with a defective file system.

After successful completion of fsck(1M) and setting the date and time (see date(1)), the operator can bring the system to multi-user operation by executing init 2.
Bringing the System Down
The shutdown procedure is designed to gracefully turn off all processes and bring the system back to single user state with all buffers flushed. To do this the operator can execute `shutdown(1M)` or the following sequence of commands:

- `killall`
- `sync`
- `init 1`
- `fsck` (optional)

The system may then be halted by typing the `<ctl>p` and `H<cr>` sequence.

System Dumps
After a system crash, the following procedure should be used to get a system dump on tape.

1. Mount a tape with write ring and bring it on-line.
2. Enter console mode with `<ctl>p`.
3. After the `>>>` prompt, halt the system with `H<cr>`.
4. Issue the following command sequence, each command followed by `<cr>`:
   - `E RO/N:F` (Examine RO thru R15)
   - `E SP` (Get the stack pointer for the next command)
   - `E/V @/N:3F` (Examine virtual memory beginning at the address from the previous instruction, and continuing for the next 63 locations; i.e., examine the stack)
   - `ST 400` (Start execution at 400, i.e., dump to tape)
5. Before returning to UNIX, execute the stand-alone `fsck(1M)`.

System Faults
On occasion, the UNIBUS or its devices fail in such a manner as to flood the console with error messages and suspend operations on UNIBUS devices. It may be possible under these conditions to bring the system down gracefully from an internal point-of-view, by inhibiting UNIBUS interrupts and running a normal shutdown. The following sequence can be executed:

- `<ctl>p`
- `>>> H`
- `>>> E 20006004` (Look at UBA control register)
- `>>> D * 1` (Clear the UBA)
- `>>> C` (Return to UNIX)

You should now be able to login as root and run a normal shutdown sequence. Reboot the system by normal means, ensuring `fsck(1M)` is performed.

INSTALLATION BOOT PROCEDURES
Tape Boot
The floppy disk delivered with the VAX-11/780 does not have tape-boot capability. The user must type in the following program to read the first record on tape drive 0. Type `<cr>` at the end of each input line:

- `>>> H`
- `>>> U`
- `>>> I`

INIT SEQ DONE

- `>>> D 20000 20008FD0`
- `>>> D + D0502001`
- `>>> D + 3204A001`
Disk Boot

The floppy disk delivered with the VAX-11/780 does not have UNIX disk-boot capability. The user must type in the following program to read the first block on disk drive 0. Type <cr> at the end of each line.

```plaintext
>>> H
>>> LINK
>>> SAVE
>>> H
>>> U
>>> I
<<< D 20000 00009FDE (Boot program for MBA 0, drive 0)
<<< D + D0512001
<<< D + D004A101
<<< D + 0400C113
<<< D + 10008F32
<<< D + D40424C1
<<< D + 8FD00CA1
<<< D + 80000000
<<< D + 320800C1
<<< D + A1FE008F
<<< D + 28C1D410
<<< D + 14C1D404
<<< D + C139D004
<<< D + 0000400
<<< S 20000
<<< S 2
<<< <ctl>C (Exit LINK mode)
```
Update prints commentary during the update operation indicating the files that are being replaced or added. Finally, a new table of contents is printed and the available space is indicated.

CONSOLE OPERATION
The following is condensed from Chapter 2 of the VAX-11/780 Hardware Handbook, DEC, 1978.

The following are the standard console commands. The most abbreviated form is shown in parentheses.

<ctl>P Causes console to exit Program I/O mode (talking to the VAX-11/780 program). This does not halt the VAX CPU.

<ctl>U Deletes the current input line.

<del> Deletes the previous character.

<ctl>C Interrupts printout.

(HEL)P Prints "help" file of which this is a part.

EXAMINE {address} Displays 8-digit hexadecimal address and its contents. See "help" file for qualifiers.

DEPOSIT {address} {data} Enters data to address.

INITIALIZE Initializes CPU.

NJAM Unjams the SBI.

SHOW Displays console and CPU state.

HALT Halts execution of VAX CPU instructions.

INITIALIZE Initializes CPU, enters address to PC, issues CONTINUE to CPU, and puts console into Program I/O mode.

CONTINUE Starts execution of VAX CPU instructions.

SET TERMINAL PROGRAM Puts console into Program I/O mode.

Causes the named floppy file to be printed and executed.

WARNINGS
Only <ctl>p can be executed from Program I/O mode. It does not stop the VAX CPU from running. Only HALT can be executed while the VAX CPU is running and not in Program I/O mode; therefore, the sequence to stop the VAX-11/780 while running UNIX (Program I/O mode) is:

<ctl>p

FILES

FILES

/etc/shutdown
/statd/*

SEE ALSO

fsck(1M), shutdown(1M), filesave(8), init(1M), tapeboot(8).