CTIX™ OPERATING SYSTEM MANUAL

Version B
Volume 1

Specifications Subject to Change.

This manual was prepared on a Convergent Technologies MegaFrame Computer System and was printed on an Imagen 8/300 Laser Printer.

First Edition (November 1985) B-09-00634-01

The *CTIX Operating System Manual, Version B,* describes the commands, system calls, libraries, data files, and device interfaces that make up the CTIX Operating System on MiniFrame Computer Systems and MightyFrame Computer Systems. Only internal-use and unbundled software products are excluded. This manual should always be your starting point when you need to find the documentation for a CTIX feature with which you are unfamiliar.

The manual consists of a large number of short entries, sometimes called "the *man* pages," after the command which accesses the entries when they are kept online. Each entry briefly documents some feature of CTIX. Some features require longer documentation than an entry in this manual; such features have an entry that outlines the feature and cross-references the manual that documents the feature fully. Entries that do not refer to other manuals are self-contained and are the final word on the features they describe.

Organization of the manual. The entries are organized into seven sections in two volumes:

Volume 1:

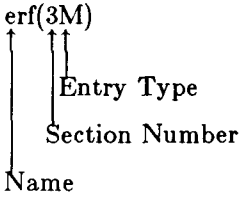
1. Commands and Application Programs.

Volume 2:

2. System Calls.
3. Subroutines and Libraries.
4. File Formats.
5. Miscellaneous Facilities.
6. Games.
7. Special files.

Within each section, entries are alphabetical by title, except for an *intro* entry at the beginning of each section.

Entry Title Conventions. An entry title looks like this example:



Name is the name of the entry. *Section Number* indicates the section that contains the entry. In this case, the entry is in Section 3, which is in Volume 2. *Entry Type* is only on entries that belong to special categories; refer to the section's *intro* entry for an explanation. In this case, a reference to *intro*(3) would tell you that *erf*(3M) describes functions from the Math Library, which the C compiler does not load by default.

Finding the entry you need. To find out which entry you need, refer to the following guides:

- The Permuted Index. This indexes each significant word in each entry's description. It is useful when you only have a general notion what you're looking for. It is also useful when you know the name of the command, function, etc., that you are interested in, but there is no entry by that name. To simplify its use, a complete Permuted Index for both volumes is in each volume.
- The Table of Contents. This is a simple list of entries, by section, together with the entry descriptions. Volume 1 has a Table of Contents for Section 1. Volume 2 has a Table of Contents for Sections 2 through 7.
- The Table of Related Entries. For Volume 1 only. A table of entries organized so that related entries are grouped together.

Section organization. Each section begins with an *intro* entry, which provides important general information for that section.

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 **bin**ary 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*. Commands that were not transported from UNIX System V reside in **/usr/local/bin**; this directory is recommended for locally implemented programs. Some administrative commands reside in **/etc** and various other places. The **/etc** directory is searched automatically if you are logged in as root; otherwise type out the full path name given under **SYNOPSIS** or change the **PATH** environment variable to include the command's directory.

Section 2, System Calls, describes the entries into the CTIX kernel, including the C language interfaces.

Section 3, Subroutines and Libraries, describes the available library functions or 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, 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*(4). 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 5, Miscellaneous Facilities, contains a variety of things. Included are descriptions of character sets, macro packages, etc.

Section 6, Games, describes the games and educational programs that reside in the directory **/usr/games**.

Section 7, Special Files, discusses the characteristics of files that actually refer to input/output devices.

Entry organization. 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.

If the entries are online, they are available via the *catman*(1) command.

HOW TO GET STARTED

This discussion provides the basic information you need to get started on CTIX: how to log in and log out, how to communicate through your terminal, and how to run a program. (See the *CTIX Programmer's Guide* for a more complete introduction to the system.)

Logging in. Most MightyFrame and MiniFrame terminals are either cluster or 9600 baud asynchronous terminals. An unused terminal prompts **login:**.

Most asynchronous 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, CTIX 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), which discusses the login sequence in more detail, and *stty*(1), which tells you how to describe the characteristics of your terminal to the system. *Profile*(4) tells how to have the shell automatically perform startup tasks when you log in. To log out, type an end-of-file indication to the shell (ASCII **EOT** character; the FINISH key on a Convergent Programmable Terminal, control-d on most others). The shell terminates and the **login:** message appears again.

How to communicate through your terminal. When you type, 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.*

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 BACKSPACE character (control-h if your terminal lacks a backspace key) erases the last character typed. Successive uses of BACKSPACE will erase characters back to, but not beyond, the beginning of the line; **@** and BACKSPACE can be typed as themselves by preceding them with \ (thus, to erase a \, you need two BACKSPACEs). 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", Programmable Terminal DELETE key) 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 (Programmable Terminal CODE-CANCEL, Control-\ on other terminals). 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.

The system 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 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 on, 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. The CTIX file system is 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 definition of *path name*.

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (*without* a prefixed /). 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*, 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 file, use *ed*(1), *ex*(1), or *vi*(1). After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file in the current directory named **a.out** (if that output is precious, use *mv*(1) to give it a less vulnerable name).

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.

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

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 $.

Changes from UNIX System V. The *CTIX Operating System Manual, Version B,* documents CTIX for MightyFrame and MiniFrame systems, which is derived from UNIX System V, Release 2.2.

The *CTIX Operating System Manual, Version B,* also includes descriptions of the CTIX Internetworking programs and tools.

These are the important changes in UNIX software in CTIX:

The language support provided by the *bs*, *efl*, *ratfor*, *sno*, and *f77* programs. In their place, Convergent Technologies can provide the following CTIX languages: GSA high level COBOL; GSA-certified FORTRAN 77; Pascal; BASIC, with both a compiler and interpreter and compatible with Convergent Technologies workstation BASIC.

A terminal name is of the form **tty***xxx* instead of **tty***xx*. RS-232 terminal numbers range from tty000 to tty255; RS-422 terminal numbers range from tty256 to tty511.

There are two changes in terminal defaults. The default speed for RS-232 terminals is 9600 baud instead of 300 baud. The default erase character for all terminals is BACKSPACE (control-h if your terminal lacks a BACKSPACE key) instead of #.

Ls columnizes its output by default if the standard output is a terminal, making *ls* easier to use on video terminals. This convention and the associated **–C** option are borrowed from the Berkeley Software Distribution.

Many Berkeley Software Distribution programs, libraries, and networking programs are included. See especially the indispensible *head*(1), *mklost+found*(1), *more*(1), *renice*(1), and *ul*(1). In addition to the AT&T *curses* (based on *terminfo*(4)), the Berkeley *ocurse* library (based on *termcap*(4)) is supported.

PERMUTED INDEX

vc: version control. . . . . vc(1)
interface. tty: controlling terminal . . . . tty(7)
local RS-232/ tp: controlling terminal's . . . tp(7)
terminals. term: conventional names for . . . term(5)
units: conversion program. . . . . units(1)
dd: convert and copy a file. . . dd(1)
to English. number: convert Arabic numerals . . number(6)
floating-point/ atof: convert ASCII string to . . atof(3C)
integers/ l3tol, ltol3: convert between 3-byte . . l3tol(3C)
integer and/ a64l, l64a: convert between long . . . a64l(3C)
and archive files to/ convert: convert object . . . convert(1)
/gmtime, asctime, tzset: convert date and time to/ . ctime(3C)
ecvt, fcvt, gcvt: convert floating-point/ . . . ecvt(3C)
scanf, fscanf, sscanf: convert formatted input. . . scanf(3S)
archive files/ convert: convert object and . . . . convert(1)
strtod, atof: convert string to/ . . . . . strtod(3C)
strtol, atol, atoi: convert string to/ . . . . . strtol(3C)
/htons, ntohl, ntohs: convert values between/ . . byteorder(3N)
dd: convert and copy a file. . . . . dd(1)
bcopy: interactive block copy. . . . . bcopy(1M)
and out. cpio: copy file archives in . . . . cpio(1)
optimal access/ dcopy: copy file systems for . . . . dcopy(1M)
label/ volcopy, labelit: copy file systems with . . . volcopy(1M)
files. cp, ln, mv: copy, link or move . . . . . cp(1)
rcp: remote file copy. . . . . rcp(1N)
system to CTIX system copy. /uuname: CTIX . . . uucp(1C)
CTIX-to-CTIX system file copy. /uupick: public . . . uuto(1C)
image file. core: format of core . . . . core(4)
core: format of core image file. . . . . core(4)
atan, atan2:/ sin, cos, tan, asin, acos, ... trig(3M)
functions. sinh, cosh, tanh: hyperbolic . . . sinh(3M)
print checksum and block count of a file. sum: . . . . sum(1)
wc: word count. . . . . wc(1)
or move files. cp, ln, mv: copy, link . . . cp(1)
cpio: format of cpio archive. . . . . cpio(4)
in and out. cpio: copy file archives . . . cpio(1)
archive. cpio: format of cpio . . . . cpio(4)
preprocessor. cpp: the C language . . . . cpp(1)
shell environment at/ cprofile: setting up a C . . . cprofile(4)
files in binary/ cpset: install object . . . . cpset(1M)
clock: report CPU time used. . . . . clock(3C)
craps: the game of craps. . . . . craps(6)
craps. craps: the game of . . . . . craps(6)
images. crash: examine system . . . crash(1M)
or rewrite an existing/ creat: create a new file . . . creat(2)
tmpnam, tempnam: create a name for a/ . . . . tmpnam(3S)
rewrite an/ creat: create a new file or . . . . creat(2)
fork: create a new process. . . . fork(2)
ctags: create a tags file. . . . . ctags(1)
tmpfile: create a temporary file. . . tmpfile(3S)
communication. socket: create an endpoint for . . . socket(2N)
channel. pipe: create an interprocess . . . pipe(2)
SCCS files. admin: create and administer . . . admin(1)
the manual. catman: create the cat files for . . . catman(1)
umask: set and get file creation mask. . . . . umask(2)
cron: clock demon. . . . . cron(1M)

process or a group of/ kill: send a signal to a . . . kill(2)
process. kill: terminate a . . . . . kill(1)
processes. killall: kill all active . . . . killall(1M)
interface. mem, kmem: system memory . . . mem(7)
quiz: test your knowledge. . . . . quiz(6)
between 3-byte integers/ l3tol, ltol3: convert . . . l3tol(3C)
long integer and/ a64l, l64a: convert between . . . a64l(3C)
/copy file systems with label checking. . . . . volcopy(1M)
systems with/ volcopy, labelit: copy file . . . . . volcopy(1M)
scanning and processing language. awk: pattern . . awk(1)
/arithmetic language. . . . . bc(1)
cpp: the C language preprocessor. . . . cpp(1)
includes: determine C language preprocessor/ . . . includes(1)
/command programming language. . . . . sh(1)
/ckpacct, dodisk, lastlogin, monacct,/ . . . . acctsh(1M)
shl: shell layer manager. . . . . shl(1)
/srand48, seed48, lcong48: generate/ . . . . . drand48(3C)
common object files. ld: link editor for . . . . . ld(1)
object file. ldclose, ldaclose: close a common . . ldclose(3X)
archive header of a/ ldahread: read the . . . . . ldahread(3X)
object file for/ ldopen, ldaopen: open a common . . ldopen(3X)
a common object file. ldclose, ldaclose: close . . . ldclose(3X)
ldeeprom: load EEPROM. . ldeeprom(1M)
parts of/ frexp, ldexp, modf: manipulate . . frexp(3C)
file access routines. ldfcn: common object . . . ldfcn(4)
header of a common/ ldfhread: read the file . . . ldfhread(3X)
symbol name for common/ ldgetname: retrieve . . . . ldgetname(3X)
manipulate/ ldlread, ldlinit, ldlitem: . . . . . ldlread(3X)
ldlread, ldlinit, ldlitem: manipulate line/ . . ldlread(3X)
ldlitem: manipulate/ ldlread, ldlinit, ... ldlread(3X)
to line number entries/ ldlseek, ldnlseek: seek . . . ldlseek(3X)
number entries/ ldlseek, ldnlseek: seek to line . . . ldlseek(3X)
relocation/ ldrseek, ldnrseek: seek to . . . . . ldrseek(3X)
ldshread, ldnshread: read an/ . . . . ldshread(3X)
indexed/named/ ldsseek, ldnsseek: seek to an . . . . ldsseek(3X)
optional file header of/ ldohseek: seek to the . . . . ldohseek(3X)
common object file for/ *ldopen, ldaopen: open a* . . *ldopen*(3X)
to relocation entries/ ldrseek, ldnrseek: seek . . . ldrseek(3X)
read an indexed/named/ ldshread, ldnshread: . . . . ldshread(3X)
to an indexed/named/ ldsseek, ldnsseek: seek . . . ldsseek(3X)
index of a symbol table/ ldtbindex: compute the . . ldtbindex(3X)
indexed symbol table/ ldtbread: read an . . . . . ldtbread(3X)
symbol table of a/ ldtbseek: seek to the . . . . ldtbseek(3X)
getopt: get option *letter from argument*/ . . . *getopt*(3C)
for simple lexical/ lex: generate programs . . . lex(1)
programs for simple lexical tasks. /generate . . lex(1)
update. lsearch, lfind: linear search and . . . lsearch(3C)
Volume Home Blocks/ libdev: manipulate . . . . . libdev(3X)
to subroutines and libraries. /introduction . . intro(3)
relation for an object library. /find ordering . . . lorder(1)
ar: archive and *library maintainer for*/ . . . *ar*(1)
ulimit: get and set user limits. . . . . ulimit(2)
/an out-going terminal line connection. . . . . dial(3C)
/type, modes, speed, and line discipline. . . . . getty(1M)
line: read one line. . . . . line(1)
common object/ linenum: line number entries in a . . linenum(4)

accept a connection on a socket. accept: . . . . . accept(2N)
bind: bind a name to a socket. . . . . bind(2N)
a connection on a socket. /initiate . . . . . connect(2N)
endpoint for/ socket: create an . . . . . socket(2N)
for connections on a socket. listen: listen . . . . listen(2N)
getsockname: get socket name. . . . . getsockname(2N)
receive a message from a socket. recv, recvfrom: . . . recv(2N)
send a message to a socket. send, sendto: . . . send(2N)
get and set options on sockets. /setsockopt: . . . getsockopt(2N)
/file perusal filter for soft-copy terminals. . . . . pg(1)
ctinstall: install software. . . . . ctinstall(1)
ssignal, gsignal: software signals. . . . . ssignal(3C)
/install and verify software using the/ . . . . qinstall(1)
sort: sort and/or merge files. . . sort(1)
qsort: quicker sort. . . . . qsort(3C)
files. sort: sort and/or merge . . sort(1)
tsort: topological sort. . . . . tsort(1)
lines common to two sorted files. /or reject . . . comm(1)
bsearch: binary search a sorted table. . . . . bsearch(3C)
change data segment space allocation. /sbrk: . . brk(2)
/unexpand: expand tabs to spaces, and vice versa. . . . expand(1)
terminal. ct: spawn getty to a remote . . ct(1C)
files. fspec: format specification in text . . . . fspec(4)
openi: open a file specified by i-node. . . . . openi(2)
receipt of a/ signal: specify what to do upon . . signal(2)
terminal type, modes, speed, and line/ /set . . . getty(1M)
settings/ gettydefs: speed and terminal . . . . gettydefs(4)
spellin, hashcheck:/ spell, hashmake, ... spell(1)
spell, hashmake, spellin, hashcheck: find/ . . spell(1)
/spellin, hashcheck: find spelling errors. . . . . spell(1)
smooth curve. spline: interpolate . . . . . spline(1G)
pieces. split: split a file into . . . . . split(1)
csplit: context split. . . . . csplit(1)
efl files. fsplit: split f77, ratfor, or . . . . fsplit(1)
pieces. split: split a file into . . . . split(1)
clean-up. uuclean: uucp spool directory . . . . . uuclean(1M)
lpr: line printer spooler. . . . . lpr(1)
/configure the LP spooling system. . . . . lpadmin(1M)
printf, fprintf, sprintf: print formatted/ . . printf(3S)
long integer data in a/ sputl, sgetl: access . . . . . sputl(3X)
exp, log, log10, pow, sqrt: exponential,/ . . . . . exp(3M)
/logarithm, power, square root functions. . . . exp(3M)
random-number/ rand, srand: simple . . . . . rand(3C)
/mrand48, jrand48, srand48, seed48,/ . . . . . drand48(3C)
scanf, fscanf, sscanf: convert/ . . . . . scanf(3S)
software signals. ssignal, gsignal: . . . . . ssignal(3C)
input/output/ stdio: standard buffered . . . . . stdio(3S)
communication/ ftok: standard interprocess . . . stdipc(3C)
sh, rsh: shell, the standard/restricted/ . . . . sh(1)
input/ rsterm: manually start and stop terminal . . rsterm(1M)
lpsched, lpshut, lpmove: start/stop the LP/ . . . . lpsched(1M)
/runacct, shutacct, startup, turnacct: shell/ . . acctsh(1M)
stat system call. stat: data returned by . . . stat(5)
status. stat, fstat: get file . . . . . stat(2)
network useful with/ stat: statistical . . . . . stat(1G)
stat: data returned by stat system call. . . . . stat(5)

TABLE OF CONTENTS

1. Commands and Application Programs

config . . . . . configure a CTIX system
conlocate . . locate a terminal to use as the virtual system console
convert . . . . convert object and archive files to common formats
cp . . . . . copy, link or move files
cpio . . . . . copy file archives in and out
cpp . . . . . the C language preprocessor
cpset . . . . . install object files in binary directories
crash . . . . . examine system images
cron . . . . . clock demon
crontab . . . . . crontab - user crontab file
csh . . . . . a shell (command interpreter) with C-like syntax
csplit . . . . . context split
ct . . . . . spawn getty to a remote terminal
ctags . . . . . create a tags file
ctinstall . . . . . install software
ctrace . . . . . C program debugger
cu . . . . . call another computer system
cut . . . . . cut out selected fields of each line of a file
cw . . . . . prepare constant-width text for troff
cxref . . . . . generate C program cross-reference
date . . . . . print and set the date
dc . . . . . desk calculator
dcopy . . . . . copy file systems for optimal access time
dd . . . . . convert and copy a file
delta . . . . . make a delta (change) to an SCCS file
deroff . . . . . remove nroff/troff, tbl, and eqn constructs
devnm . . . . . device name
df . . . . . report number of free disk blocks
diff . . . . . differential file comparator
diff3 . . . . . 3-way differential file comparison
diffmk . . . . . mark differences between files
dircmp . . . . . directory comparison
diskusg . . . . diskusg - generate disk accounting data by user ID
dismount . . . . . remove exchangeable disk
du . . . . . summarize disk usage
dump . . . . . dump selected parts of an object file
echo . . . . . echo arguments
ed . . . . . text editor
edit . . . . . text editor (variant of ex for casual users)
enable . . . . . enable/disable LP printers
env . . . . . set environment for command execution
eqn . . . . . format mathematical text for nroff or troff
errdead . . extract error records and status information from dump
errdemon . . . . . error-logging demon
errpt . . . . . process a report of logged errors
errstop . . . . . terminate the error-logging demon
ex . . . . . text editor
expand . . . . . expand tabs to spaces, and vice versa
expr . . . . . evaluate arguments as an expression
factor . . . . . factor a number

ff . . . . . list file names and statistics for a file system
file . . . . . determine file type
finc . . . . . fast incremental backup
find . . . . . find files
fold . . . . . fold long lines for finite width output device
frec . . . . . recover files from a backup tape
fsck . . . . . file system consistency check and interactive repair
fsdb . . . . . file system debugger
fsplit . . . . . split f77, ratfor, or efl files
ftp . . . . . file transfer program
ftpd . . . . . DARPA Internet File Transfer Protocol server
fuser . . . . . identify processes using a file or file structure
fwtmp . . . . . manipulate connect accounting records
gdev . . . . . graphical device routines and filters
ged . . . . . graphical editor
get . . . . . get a version of an SCCS file
getopt . . . . . parse command options
getty . . . . . set terminal type, modes, speed, and line discipline
graph . . . . . draw a graph
graphics . . . . . access graphical and numerical commands
greek . . . . . select terminal filter
grep . . . . . search a file for a pattern
gutil . . . . . graphical utilities
hd . . . . . hexadecimal and ascii file dump
head . . . . . give first few lines
help . . . . . ask for help
hinv . . . . . hardware inventory
hp . handle special functions of HP 2640 and 2621-series terminals
hyphen . . . . . find hyphenated words
id . . . . . print user and group IDs and names
ifconfig . . . . . configure network interface parameters
includes . . . . . determine C language preprocessor include files
init . . . . . process control initialization
install . . . . . install commands
ipcrm . remove message queue, semaphore set or shared memory id
ipcs . . . . . report inter-process communication facilities status
iv . . . . . initialize and maintain volume
join . . . . . relational database operator
kill . . . . . terminate a process
killall . . . . . kill all active processes
ld . . . . . link editor for common object files
lddrv . . . . . manage loadable drivers
ldeeprom . . . . . load EEPROM
lex . . . . . generate programs for simple lexical tasks
line . . . . . read one line
link . . . . . exercise link and unlink system calls
lint . . . . . a C program checker
login . . . . . sign on
logname . . . . . get login name
lorder . . . . . find ordering relation for an object library

lp . . . . . send/cancel requests to an LP line printer
lpadmin . . . . . configure the LP spooling system
lpr . . . . . line printer spooler
lpsched . . start/stop the LP request scheduler and move requests
lpset . . . . . set parallel line printer options
lpstat . . . . . print LP status information
ls . . . . . list contents of directory
m4 . . . . . macro processor
machid . . . . . provide truth value about your processor type
mail . . . . . send mail to users or read mail
mailx . . . . . interactive message processing system
make . . . . maintain, update, and regenerate groups of programs
man . . . . . print entries in this manual
mesg . . . . . permit or deny messages
mkdir . . . . . make a directory
mkfs . . . . . construct a file system
mkifile . . . . . make an ifile from an object file
mklost+found . . . . . make a lost+found directory for fsck
mknod . . . . . build special file
mktpy . . . . . install or relocate a PT or GT local printer
mm . . . . print/check documents formatted with the MM macros
mmt . . . . . typeset documents, view graphs, and slides
more . . . . . text perusal
mount . . . . . mount and dismount file system
mvdir . . . . . move a directory
ncheck . . . . . generate names from i-numbers
netman . . . . . form-based network management
newform . . . . . change the format of a text file
newgrp . . . . . log in to a new group
news . . . . . print news items
nice . . . . . run a command at low priority
nl . . . . . line numbering filter
nm . . . . . print name list of common object file
nohup . . . . . run a command immune to hangups and quits
nroff . . . . . format text
od . . . . . octal dump
pack . . . . . compress and expand files
passwd . . . . . change login password
paste merge same lines of several files or subsequent lines of one file
path . . . . . locate executable file for command
pg . . . . . file perusal filter for soft-copy terminals
pr . . . . . print files
prof . . . . . display profile data
profiler . . . . . operating system profiler
prs . . . . . print an SCCS file
ps . . . . . report process status
ptx . . . . . permuted index
pwck . . . . . password/group file checkers
pwd . . . . . working directory name
qinstall . . . . . install and verify software using the mkfs(1)

qlist . . . . . print out file lists from proto file; set links based on
rcmd . . . . . remote shell command execution
rcp . . . . . remote file copy
reboot . . . . . reboot the system
regcmp . . . . . regular expression compile
renice . . . . . alter priority of running process by changing nice
rexecd . . . . . remote execution server
rm . . . . . remove files or directories
rmdel . . . . . remove a delta from an SCCS file
route . . . . . manually manipulate the routing tables
rsterm . . . . manually start and stop terminal input and output
runacct . . . . . run daily accounting
rwho . . . . . who is logged in on local network
rwhod . . . . . node status server
sact . . . . . print current SCCS file editing activity
sadp . . . . . disk access profiler
sag . . . . . system activity graph
sar . . . . . system activity reporter
sar . . . . . system activity report package
sccsdiff . . . . . compare two versions of an SCCS file
script . . . . . make typescript of terminal session
sdb . . . . . symbolic debugger
sdiff . . . . . side-by-side difference program
sed . . . . . stream editor
setaddr . . . . . set DARPA Internet address from node name
setenet . . . . . write Ethernet address on disk
setmnt . . . . . establish mount table
setuname . . . . . set name of system
sh . shell, the standard/restricted command programming language
shl . . . . . shell layer manager
shutdown . . . . . terminate all processing
size . . . . . print section sizes of common object files
slattach . . . . attach and detach serial lines as network interfaces
sleep . . . . . suspend execution for an interval
sort . . . . . sort and/or merge files
spell . . . . . find spelling errors
spline . . . . . interpolate smooth curve
split . . . . . split a file into pieces
stat . . . . . statistical network useful with graphical commands
strings . . . . . extract the ASCII text strings in a file
strip . . . . . strip symbol and line number information
stty . . . . . set the options for a terminal
su . . . . . become super-user or another user
sum . . . . . print checksum and block count of a file
swap . . . . . swap administrative interface
sync . . . . . update the super block
tabs . . . . . set tabs on a terminal
tail . . . . . deliver the last part of a file
tar . . . . . tape file archiver
tbl . . . . . format tables for nroff or troff

tdl . . . . . RS-232 terminal download
tee . . . . . pipe fitting
telnet . . . . . user interface to TELNET protocol
telnetd . . . . . DARPA TELNET protocol server
test . . . . . condition evaluation command
tftp . . . . . user interface to the DARPA TFTP protocol
tftpd . . . . . DARPA Trivial File Transfer Protocol server
tic . . . . . terminfo compiler
time . . . . . time a command
timex . . time a command; report process data and system activity
toc . . . . . graphical table of contents routines
touch . . . . . update access and modification times of a file
tplot . . . . . graphics filters
tput . . . . . query terminfo database
tr . . . . . translate characters
troff . . . . . typeset text
trpt . . . . . print protocol trace
true . . . . . provide truth values
tset . . . set terminal, terminal inteface, and terminal environment
tsort . . . . . topological sort
tty . . . . . get the name of the terminal
ul . . . . . do underlining
umask . . . . . set file-creation mode mask
uname . . . . . print name of current CTIX system
unget . . . . . undo a previous get of an SCCS file
uniq . . . . . report repeated lines in a file
units . . . . . conversion program
update . . . . . provide disk synchronization
uuclean . . . . . uucp spool directory clean-up
uucp . . . . . CTIX system to CTIX system copy
uucpd . . . . . network uucp server
uustat . . . . . uucp status inquiry and job control
uusub . . . . . monitor uucp network
uuto . . . . . public CTIX-to-CTIX system file copy
uux . . . . . CTIX-to-CTIX system command execution
val . . . . . validate SCCS file
vc . . . . . version control
vi . . . . . screen-oriented (visual) display editor based on ex
volcopy . . . . . copy file systems with label checking
wait . . . . . await completion of process
wall . . . . . write to all users
wc . . . . . word count
what . . . . . identify SCCS files
who . . . . . who is on the system
whodo . . . . . who is doing what
wm . . . . . window management
write . . . . . write to another user
xargs . . . . . construct argument list(s) and execute command
xstr . . . . . extract and share strings in C programs
yacc . . . . . yet another compiler-compiler

TABLE OF RELATED ENTRIES

Administration

Accounting and Profiling

acct . . . overview of accounting and miscellaneous accounting commands
acctcms . . . . . command summary from per-process accounting records
acctcom . . . . . search and print process accounting file(s)
acctcon . . . . . connect-time accounting
acctmerg . . . . . merge or add total accounting files
acctprc . . . . . process accounting
acctsh . . . . . shell procedures for accounting
fwtmp . . . . . manipulate connect accounting records
prof . . . . . display profile data
runacct . . . . . run daily accounting
sar . . . . . system activity reporter
sar . . . . . system activity report package

Backups

ff . . . . . list file names and statistics for a file system
finc . . . . . fast incremental backup
frec . . . . . recover files from a backup tape
volcopy . . . . . copy file systems with label checking

Controlling System State

brc . . . . . system initialization shell scripts
crash . . . . . examine system images
getty . . . . . set terminal type, modes, speed, and line discipline
init . . . . . process control initialization
killall . . . . . kill all active processes
login . . . . . sign on
shutdown . . . . . terminate all processing
wall . . . . . write to all users

Disk Management

bcopy . . . . . interactive block copy
clri . . . . . clear i-node
dcopy . . . . . copy file systems for optimal access time
devnm . . . . . device name
df . . . . . report number of free disk blocks
fsck . . . . . file system consistency check and interactive repair
fsdb . . . . . file system debugger
fuser . . . . . identify processes using a file or file structure
link . . . . . exercise link and unlink system calls
mkfs . . . . . construct a file system
mklost+found . . . . . make a lost+found directory for fsck
mount . . . . . mount and dismount file system
mvdir . . . . . move a directory
ncheck . . . . . generate names from i-numbers
setmnt . . . . . establish mount table
sync . . . . . update the super block

General

conlocate . . . . . locate a terminal to use as the virtual system console
config . . . . . configure a CTIX system
cpset . . . . . install object files in binary directories
cron . . . . . clock daemon
dismount . . . . . remove floppy or cartridge disk
errdead . . . . . extract error records and status information from dump
errdemon . . . . . error-logging demon
errpt . . . . . process a report of logged errors
errstop . . . . . terminate the error-logging daemon
install . . . . . install commands
iv . . . . . initialize and maintain volume
mknod . . . . . build special file
path . . . . . locate executable file for command
pwck . . . . . password/group file checkers
rsterm . . . . . manually start and stop terminal input and output
setuname . . . . . set name of system
update . . . . . provide disk synchronization
whodo . . . . . who is doing what

Interprocess Communication

ipcrm . . . . remove a message queue, semaphore set or shared memory id
ipcs . . . . . report inter-process communication facilities status

Basic File Commands

cat . . . . . concatenate and print files
chmod . . . . . change mode
chown . . . . . change owner or group
dircmp . . . . . directory comparison
cp . . . . . copy, link or move files
dd . . . . . convert and copy a file
file . . . . . determine file type
find . . . . . find files
ls . . . . . list contents of directory
pwd . . . . . working directory name
mkdir . . . . . make a directory
rm . . . . . remove files or directories
umask . . . . . set file-creation mode mask

Basic General Commands

calendar . . . . . reminder service
date . . . . . print and set the date
id . . . . . print user and group IDs and names
kill . . . . . terminate a process
logname . . . . . get login name
newgrp . . . . . log in to a new group
news . . . . . print news items
passwd . . . . . change login password
ps . . . . . report process status
uname . . . . . print name of system
who . . . . . who is on the system

Communication Between Systems (uucp)

ct . . . . . spawn getty to a remote terminal
cu . . . . . call another computer system

uuclean . . . . . uucp spool directory clean-up
uucp . . . . . copy data between computer systems
uustat . . . . . uucp status inquiry and job control
uusub . . . . . monitor uucp network
uuto . . . . . public computer system-to-computer system file copy
uux . . . . . remote system command execution

Communication Between Users

mail . . . . . send mail to users or read mail
mailx . . . . . interactive message processing system
mesg . . . . . permit or deny messages
write . . . . . write to another user

Document Formatting and Checking

col . . . . . filter reverse line-feeds
cw . . . . . prepare constant-width text for troff
deroff . . . . . remove nroff/troff, tbl, and eqn constructs
du . . . . . summarize disk usage
eqn . . . . . format mathematical text for nroff or troff
greek . . . . . select terminal filter
hyphen . . . . . find hyphenated words
mm . . . . . print/check documents formatted with the MM macros
mmt . . . . . typeset documents, view graphs, and slides
nroff . . . . . format text
ptx . . . . . permuted index
spell . . . . . find spelling errors
tbl . . . . . format tables for nroff or troff
troff . . . . . typeset text

Internetworking Tools

ftp . . . . . file transfer program
ftpd . . . . . DARPA Internet File Transfer Protocol server
ifconfig . . . . . configure network interface parameters
mkhosts . . . . . make node name commands
netman . . . . . form-based network management
netstat . . . . . show network status
rcmd . . . . . remote shell command execution
rcp . . . . . remote file copy
rexecd . . . . . remote execution server
rlogin . . . . . remote login
rlogind . . . . . remote login server
route . . . . . remove files or directories
rshd . . . . . remote shell server
ruptime . . . . . display status of notes on local network
rwho . . . . . who is loggin in on local network
rwhod . . . . . node status server
setaddr . . . . . set DARPA Internet address from nodename
setenet . . . . . write Ethernet address on disk
slattach . . . . . attach serial lines as network interfaces
sldetach . . . . . detach serial lines as network interfaces
telnet . . . . . user interface to TELNET protocol
telnetd . . . . . DARPA TELNET protocol server
tftp . . . . . user interface to the DARPA TFTP protocol
tftpd . . . . . DARPA Trivial File Transfer Protocol server
trpt . . . . . print protocol trace

Mathematics Tools

bc . . . . . arbitrary-precision arithmetic language
dc . . . . . desk calculator
factor . . . . . factor a number
spline . . . . . interpolate smooth curve
units . . . . . conversion program

Miscellaneous

man . . . . . print entries in this manual
nl . . . . . line numbering filter
pack . . . . . compress and expand files
script . . . . . make typescript of terminal session
su . . . . . become super-user or another user
wc . . . . . word count

Offline Storage

cpio . . . . . copy file archives in and out
tar . . . . . tape file archiver

Printer Spooling

accept . . . . . allow/prevent LP requests
enable . . . . . enable/disable LP printers
lp . . . . . send/cancel requests to an LP line printer
lpadmin . . . . . configure the LP spooling system
lpr . . . . . line printer spooler
lpsched . . . . . start/stop the LP request scheduler and move requests
lpset . . . . . set parallel line printer options
lpstat . . . . . print LP status information

Program Development

adb . . . . . absolute debugger
ar . . . . . archive and library maintainer for portable archives
as . . . . . mc68010 assembler
cb . . . . . C program beautifier
cc . . . . . C compiler
cflow . . . . . generate C flow graph
cpp . . . . . the C language preprocessor
ctags . . . . . create a tags file
cxref . . . . . generate C program cross reference
dump . . . . . dump selected parts of an object file
fsplit . . . . . split fortran, ratfor, or efl files
hd . . . . . hexadecimal and ascii file dump
ld . . . . . link editor for common object files
lint . . . . . a C program checker
lorder . . . . . find ordering relation for an object library
m4 . . . . . macro processor
make . . . . . maintain, update, and regenerate groups of programs
nm . . . . . print name list of common object file
od . . . . . octal dump
regcmp . . . . . regular expression compile
size . . . . . print section sizes of common object files
strings . . . . . extract the ASCII text strings in a file
strip . strip symbol and line number information from a common object file
time . . . . . time a command

timex . . . . . time a command; report process data and system activity
touch . . . . . update access and modification times of a file
tsort . . . . . topological sort
xstr . . . . . extract and share strings in C programs

Source Code Control System

admin . . . . . create and administer SCCS files
cdc . . . . . change the delta commentary of an SCCS delta
comb . . . . . combine SCCS deltas
delta . . . . . make a delta (change) to an SCCS file
get . . . . . get a version of an SCCS file
help . . . . . ask for help
prs . . . . . print an SCCS file
rmdel . . . . . remove a delta from an SCCS file
sact . . . . . print current SCCS file editing activity
sccsdiff . . . . . compare two versions of an SCCS file
unget . . . . . undo a previous get of an SCCS file
val . . . . . validate SCCS file
vc . . . . . version control
what . . . . . identify SCCS files

Terminal Support

300 . . . . . handle special functions of DASI 300 and 300s terminals
4014 . . . . . paginator for the TEKTRONIX 4014 terminal
450 . . . . . handle special functions of the DASI 450 terminal
asa . . . . . interpret ASA carriage control characters
clear . . . . . clear terminal screen
hp . . . . . handle special functions of HP 2640 and 2621-series terminals
stty . . . . . set the options for a terminal
tabs . . . . . set tabs on a terminal
tdl . . . . . rs232 terminal download
tic . . . . . terminfo compiler
tset . . . . . set terminal, terminal inteface, and terminal environment
tty . . . . . get the terminal's name
wm . . . . . window management

Text Tools

Browsers, Editors, and Splitters

bfs . . . . . big file scanner
csplit . . . . . context split
ed . . . . . text editor
ex . . . . . text editor
more . . . . . text perusal
newform . . . . . change the format of a text file
pg . . . . . file perusal filter for soft-copy terminals
split . . . . . split a file into pieces
vi . . . . . screen-oriented (visual) display editor based on ex

Comparing Files

bdiff . . . . . big diff
cmp . . . . . compare two files
comm . . . . . select or reject lines common to two sorted files
diff . . . . . differential file comparator
diff3 . . . . . 3-way differential file comparison

diffmk . . . . . mark differences between files
sdiff . . . . . side-by-side difference program

Customizable Filters and Text Programming Languages

awk . . . . . pattern scanning and processing language
cut . . . . . cut out selected fields of each line of a file
fold . . . . . fold long lines for finite width output device
grep . . . . . search a file for a pattern
join . . . . . relational database operator
lex . . . . . generate programs for simple lexical tasks
paste . . . . merge same lines of several files or subsequent lines of one file
pr . . . . . print files
sed . . . . . stream editor
sort . . . . . sort and/or merge files
tail . . . . . deliver the last part of a file
tr . . . . . translate characters
uniq . . . . . report repeated lines in a file
yacc . . . . . yet another compiler-compiler

Graphics and Displays

banner . . . . . make posters
cal . . . . . print calendar
graph . . . . . draw a graph

Using and Programming the Shell

basename . . . . . deliver portions of path names
chroot . . . . . change root directory for a command
cd . . . . . change working directory
echo . . . . . echo arguments
env . . . . . set environment for command execution
expr . . . . . evaluate arguments as an expression
getopt . . . . . parse command options
line . . . . . read one line
machid . . . . . processor type
nice . . . . . run a command at low priority
nohup . . . . . run a command immune to hangups and quits
sh . . . . . shell, the standard/restricted command programming language
sleep . . . . . suspend execution for an interval
tee . . . . . pipe fitting
test . . . . . condition evaluation command
true . . . . . provide truth values
wait . . . . . await completion of process
xargs . . . . . construct argument list(s) and execute command

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 for system maintenance and administration.

(1N), (1NM)
Commands for the CTIX TCP/IP networking packages. To use these commands you must have a special version of the CTIX kernel that supports TCP/IP.

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), exit(2), wait(2), 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.

WARNINGS

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

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

The *300* command 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%. The *300* command can be used to print equations neatly, in the sequence:

neqn 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* 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.

−d*t*,*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*(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

The *300* command 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* names of, and resulting output for, the Greek and special characters supported by *300* are shown in *greek*(5).

SEE ALSO

450(1), eqn(1), graph(1G), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5).

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**] [**-c**N] [**-p**L] [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 interpreted 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 **!***cmd* will send the *cmd* to the shell.

The command line options are:

-t Do not wait between pages (useful for directing output into a file).

-n Start printing at the current cursor position and never erase the screen.

-c*N* Divide the screen into *N* columns and wait after the last column.

-p*L* 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

The *450* command 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). Use *450* 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.

Use *450* 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 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* names of, and resulting output for, the Greek and special characters supported by *450* are shown in *greek*(5).

SEE ALSO

300(1), eqn(1), graph(1G), mesg(1), nroff(1), stty(1), tabs(1), tbl(1), tplot(1G), greek(5)

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

accept, reject – allow/prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [**–r**[reason]] destinations

DESCRIPTION

Accept allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

Reject prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject*.

–r[*reason*] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next **–r** option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the **–r** option is not present or the **–r** option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

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

SYNOPSIS

/usr/lib/acct/acctdisk

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

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

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 **/etc/utmp/**, as described in *utmp*(4). The programs described in *acctcon*(1M) convert this file into session and charging records, which are then summarized by *acctmerg*(1M).

Process accounting is performed by the CTIX System 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*(4)) can be merged and summarized into total accounting records by *acctmerg* (see **tacct** format in *acct*(4)). *Prtacct* (see *acctsh*(1M)) is used to format any or all accounting records.

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

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

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

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

```
acctwtmp uname >> /etc/wtmp
acctwtmp "file save" >> /etc/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
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
CTIX Programmer's Guide.
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

acctcms – command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [options] files

DESCRIPTION

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

–a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, and "hog factor", characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore-minutes.

–c Sort by total CPU time, rather than total kcore-minutes.

–j Combine all commands invoked only once under "***other".

–n Sort by number of command invocations.

–s Any file names encountered hereafter are already in internal summary format.

–t Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old style **acctcms** internal summary format records.

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

–p Output a prime-time-only command summary.

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

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

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

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

SEE ALSO

acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

BUGS

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

NAME

acctcom – search and print process accounting file(s)

SYNOPSIS

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

DESCRIPTION

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

The command name is prepended with a **#** if it was executed with *super-user* privileges. If a process is not associated with a known terminal, a **?** is printed in the TTYNAME field.

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

If any *file* arguments are given, they are read 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 such files of which all but the current file are found in **/usr/adm/pacct***?*. The *options* are:

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

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

–f Print the *fork/exec* flag and system exit status columns in the output.

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

(total CPU time)/(elapsed time).

-i	Print columns containing the I/O counts in the output.
-k	Instead of memory size, show total kcore-minutes.
-m	Show mean core size (the default).
-r	Show CPU factor (user time/(system-time + user-time).
-t	Show separate system and user CPU times.
-v	Exclude column headings from the output.
-l *line*	Show only processes belonging to terminal **/dev/***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 processes associated with unknown user IDs.
-g *group*	Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.
-d *mm/dd*	Any *time* arguments following this flag are assumed to occur on the given month *mm* and the day *dd* rather than during last 24 hours. This is needed for looking at old files.
-s *time*	Select processes existing at or after *time*, given in the format *hr* [:*min* [:*sec*]].
-e *time*	Select processes existing at or before *time*.
-S *time*	Select processes starting at or after *time*.
-E *time*	Select processes ending at or before *time*. Using the same *time* for both **-S** and **-E** shows the processes that existed at *time*.
-n *pattern*	Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that + means one or more occurrences.
-q	Do not print any output records, just print the average statistics as with the **-a** option.
-o *ofile*	Copy selected process records in the input data format to *ofile*; supress standard output printing.
-H *factor*	Show only processes that exceed *factor*, where factor is the "hog factor" as explained in option **-h** above.

–O *sec*	Show only processes with CPU system time exceeding *sec* seconds.
–C *sec*	Show only processes with total CPU time, system plus user, exceeding *sec* seconds.
–I *chars*	Show only processes transferring more characters than *chars*.

Listing options together has the effect of a logical *and*.

FILES

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

SEE ALSO

ps(1),
acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), su(1), acct(2), acct(4), utmp(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time and option **–d** is not used, then *time* is interpreted as occurring on the previous day.

NAME

acctcon1, acctcon2 – connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/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 normally be redirected from **/etc/wtmp**. Its output is ASCII, giving device, 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. Hang-up, termination of *login*(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).

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

Acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see **tacct** format in *acct*(4)).

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

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), login(1), runacct(1M), acct(2), acct(4), utmp(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

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

/usr/lib/acct/acctmerg [options] [file] . . .

DESCRIPTION

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

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

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

EXAMPLES

```
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(4), utmp(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

acctprc1, acctprc2 – process accounting

SYNOPSIS

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

/usr/lib/acct/acctprc2

DESCRIPTION

Acctprc1 reads input in the form described by *acct*(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If **ctmp** is given, it is expected to contain a list of login sessions, 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), cron(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

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).

NOTE

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor. For example, on Convergent Technologies systems this measure would be in 4-kilobyte units.

NAME

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

SYNOPSIS

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

/usr/lib/acct/ckpacct [blocks]

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

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

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

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

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

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

/usr/lib/acct/startup

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

DESCRIPTION

Chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to **/usr/adm/fee**, to be merged with other accounting records during the night.

Ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* will be invoked with argument *switch*. If the number of free 512-byte disk blocks in the **/usr** file system falls below 500, *ckpacct* will automatically turn off the collection of process accounting records via the **off** argument to *turnacct*. When at least this number of blocks is restored, the accounting will be activated again. This feature is sensitive to the frequency at which *ckpacct* is executed, usually by *cron*.

Dodisk should be invoked by *cron* to perform the disk accounting functions. By default, it will do disk accounting on the special files in **/etc/checklist.** If the **–o** flag is used, it will do a slower version of disk accounting by login directory. *Files* specify the one or more filesystem names where disk accounting will be done. If *files* are used, disk accounting will be done on these filesystems only. If the **–o** flag is used, *files* should

be mount points of mounted filesystem. If omitted, they should be the special file names of mountable filesystems.

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

Monacct should be invoked once each month or each accounting period. *Number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01–12). This default is useful if *monacct* is to executed via *cron*(1M) on the first day of each month. *Monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

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

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

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

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

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

Shutacct should be invoked during a system shutdown (usually in **/etc/shutdown**) to turn process accounting off and append a "reason" record to **/etc/wtmp**.

Startup should be called by **/etc/rc** 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 turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr* (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then turns accounting back on again. This procedure is called by *ckpacct* and thus can be taken care of by the *cron* and used to keep **pacct** to a reasonable size.

FILES

/usr/adm/fee accumulator for fees

/usr/adm/pacct current file for per-process accounting

/usr/adm/pacct*
used if pacct gets large and during execution of daily accounting procedure

/etc/wtmp login/logoff summary

/usr/lib/acct/ptelus.awk
contains the limits for exceptional usage by login id

/usr/lib/acct/ptecms.awk
contains the limits for exceptional usage by command name

/usr/adm/acct/nite
working directory

/usr/lib/acct holds all accounting commands listed in sub-class 1M of this manual

/usr/adm/acct/sum
summary directory, should be saved

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

adb – absolute 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 CTIX programs.

Objfil is normally an executable program file, preferably containing a symbol 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 executed. 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 Hexadecimal by default or if preceded by 0x; octal if preceded by 0o or 0O; decimal if preceded by 0t or 0T.

integer.fraction
A 32-bit floating point number.

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

< name
The value of *name*, which is either a variable name or a 68010/68020 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 registers are **d0** through **d7**, **a0** through **a7**, **sp**, **pc**, **cc**, **sr**, and **usp**.

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*.

From C, only external variables are available as symbols. The symbol name is the same as the C variable name, except that an underscore (_) is prepended to any name that is the same as the name for a register.

(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 follows:

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. The value *n* is the length of the string including its zero

terminator.

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

i n Print as machine instructions. The value *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 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.

[?/]**l** *value mask*

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used then 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** *b1 e1 f1*[?/]
New values for (*b1, e1, f1*) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the **?** or **/** is followed by ***** then the second segment (*b2*, *e2*, *f2*) 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**.
- **b** Print all breakpoints and their associated counts and commands.
- **c** C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **fp**). 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.
- **f** Print the 68881 floating-point registers.

m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

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

c*s* The subprocess is continued with signal *s* (see *signal*(2)). If *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 **r**.

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

k The current subprocess, if any, is terminated.

VARIABLES

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

0 The last value printed.

1 The last offset part of an instruction source.

2 The previous value of variable 1.

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

b The base address of the data segment.

d The data segment size.

e The entry point.

m The "magic" number (0407, 0410, or 0413).

s The stack segment size.

t The text segment size.

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 (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows:

$b1 \leq address < e1 \Rightarrow$ *file address* $= address + f1 - b1$

otherwise

$b2 \leq address < e2 \Rightarrow$ *file address* $= address + f2 - b2,$

otherwise, the requested *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 **?** or **/** is followed by an ***** then only the second triple is used.

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

In order for *adb* to be used on large files all appropriate values are kept as signed 32-bit integers.

FILES

/dev/kmem
/dev/swap
a.out
core

SEE ALSO

ptrace(2), a.out(4), core(4).

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]] [**–r**rel] [**–t**[name]] [**–f**flag[flag-val]] [**–d**flag[flag-val]] [**–a**login] [**–e**login] [**–m**[mrlist]] [**–y**[comment]] [**–h**] [**–z**] files

DESCRIPTION

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

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

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

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

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

command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **–i** keyletter). Note that the **–i** keyletter implies the **–n** keyletter.

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

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

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

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

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

editing. The default value for an unspecified **c** flag is 9999.

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

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

i*str* Causes the "No id keywords (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 SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string, however the string must contain a keyword, and no embedded newlines.

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

l*list* A *list* of releases to which deltas can no longer be made (**get** **-e** against one of these "locked" releases fails). The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
<range> ::= *RELEASE NUMBER* | **a**

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

n Causes *delta*(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7,

releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

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

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

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

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

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

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

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

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

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

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

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

-m[*mrlist*] The list of Modification Requests (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The **v** flag must be set and the *MR* numbers are validated if the **v** flag has a value (the name of an *MR* number validation program).

Diagnostics will occur if the **v** flag is not set or *MR* validation fails.

−h Causes *admin* to check the structure of the SCCS file (see *sccsfile*(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 **−h**, above).

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

FILES

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

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

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner

allowing use of *ed*(1). *Care must be taken!* The edited file should *always* be processed by an **admin** **–h** to check for corruption followed by an **admin** **–z** to generate a proper check-sum. Another **admin** **–h** is recommended to ensure the SCCS file is valid.

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

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4). *CTIX Programmer's Guide,* Section 9.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar – archive and library maintainer for portable archives

SYNOPSIS

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

DESCRIPTION

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

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar*(4). The archive symbol table (described in *ar*(4)) is used by the link editor (*ld*(1)) to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *ar* when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *ar*(1) command is used to create or update the contents of such an archive, the symbol table is rebuilt. The **s** option described below will force the symbol table to be rebuilt.

Key is an optional –, followed by one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcls**. *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 dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the

archive. 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 Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, precede each file with a name.

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

l Place temporary files in the local current working directory, rather than in the directory specified by the environment variable TMPDIR or in the default directory **/tmp**.

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

FILES

/tmp/ar* temporaries

SEE ALSO

convert(1), file(1), ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4).

NOTES

This archive format is new to this release. *ar* will not accept archive files in the old format; the *convert*(1) command can be used to change an older archive file into an archive file that is recognized by this *ar* command.

BUGS

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

NAME

as – assembler

SYNOPSIS

as [**–o** objfile] [**–n**] [**–j**] [**–m**] [**–R**] [**–r**] [**–[bwl]**] [**–V**] [**–T**] sourcefile

DESCRIPTION

The *as* command translates mc68010 or mc68020 assembly language in *sourcefile* into object code. The result is a common object file, suitable for input to the link editor. The following flags may be specified in any order:

–o *objfile* Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.

–n Turn off long/short address optimization. By default, address optimization takes place.

–j Invoke the long-jump assembler. The address optimization algorithm chooses between long and short address lengths, with short lengths chosen when possible. Often, three distinct lengths are allowed by the machine architecture; a choice must be made between two of those lengths. When the two choices given to the assembler exclude the largest length allowed, then some addresses might be unrepresentable. The long-jump assembler will always have the largest length as one of its allowable choices. If the assembler is invoked without this option, and the case arises where an address is unrepresentable by either of the two allowed choices, then the user will be informed of the error, and advised to try again using the **–j** option.

–m Run the *m4* macro pre-processor on the input to the assembler.

–R Remove (unlink) the input file after assembly is completed.

–r Place all assembled data (normally placed in the **data** section) into the **text** section. This option effectively disables the **.data** pseudo operation. This option is off by default.

–[bwl] Create byte (**b**), halfword (**w**) or long (**l**) displacements for undefined symbols. (An undefined symbol is a reference to a symbol

whose definition is external to the input file or a forward reference.) The default value for this option is long (l) displacements.

-V Write the version number of the assembler being run on the standard error output.

-T Truncate symbols to eight characters.

FILES

/usr/tmp/as[1-6]*XXXXXX* temporary files

SEE ALSO

ld(1), m4(1), nm(1), strip(1), a.out(4).

WARNING

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

Use the **-b** or **-w** option only when undefined symbols are known to refer to locations representable by the specified default displacement. Use of either option when assembling a file containing a reference to a symbol that is to be resolved by the loader can lead to unpredictable results, since the loader may be unable to place the address of the symbol into the space provided.

BUGS

The **.align** assembler directive is not guaranteed to work in the **.text** section when optimization is performed.

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

NAME

asa – interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

Asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names are supplied. The first character of each line is assumed to be a control character; their meanings are:

' '	(blank) single new line before printing
0	double new line before printing
1	new page before printing
+	overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ' '. The first character of a line is *not* printed. If any such lines appear, an appropriate diagnostic will appear on standard error. This program forces the first line of each input file to start on a new page.

To view correctly the output of FORTRAN programs which use ASA carriage control characters, *asa* could be used as a filter thusly:

```
a.out | asa | lp
```

and the output, properly formatted and paginated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by:

```
asa file
```

NAME

at, batch – execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]
at *-r*job...
at *-l*[*job*...]

batch

DESCRIPTION

At and *batch* read commands from standard input to be executed at a later time. *At* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. *At* **-r** removes jobs previously scheduled with *at*. The **-l** option reports all jobs scheduled for the invoking user.

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

Users are permitted to use *at* if their name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** is checked to determine if the user should be denied access to *at*. If neither file exists, only root is allowed to submit a job. If **at.deny** exists and is empty, global usage is permitted. If **at.allow** exists and is empty, no usage is permitted. If **at.allow** exists, **at.deny** is ignored. The allow/deny files consist of one user name per line.

The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour*:*minute*. A suffix **am** or **pm** may be appended; otherwise a 24-hour clock time is understood. The suffix **zulu** may be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

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

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.)

Thus legitimate commands include:

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

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

Batch submits a batch job. It is almost equivalent to "at now", but not quite. For one, it goes into a different queue. For another, "at now" will respond with the error message "too late".

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

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *Sh*(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

batch
nroff *filename* > *outfile*
<code-D>
(hold down 'code' and depress 'D')

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

batch <<!
nroff *filename* 2>&1 > *outfile* | mail *loginid*
!

To have a job reschedule itself, invoke *at* from within the shell procedure, by including code similar to the following within the shell file:

echo "sh *shellfile*" | at 1900 thursday next week

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users

/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

cron(1), kill(1), mail(1), nice(1), ps(1), sh(1).

NOTE

At always runs **/bin/sh**, not *csh*.

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk – pattern scanning and processing language

SYNOPSIS

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

DESCRIPTION

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

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

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

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

A pattern-action statement has the form:

pattern { action }

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

```
if ( conditional ) statement
        [ else statement ]
while ( conditional ) statement
for ( expression ; 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 **-F***c* option.

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
       { print }
```

command line: awk -f program n=5 input

SEE ALSO

grep(1), lex(1), malloc(3X), sed(1).
CTIX Programmer's Guide, Section 16.

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.

SEE ALSO

echo(1).

NAME

basename, dirname – deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

Basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (` `) within shell procedures.

Dirname 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).

BUGS

The *basename* of / is null and is considered an error.

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [**–c**] [**–l**] [file ...]

DESCRIPTION

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

Comments

are enclosed in /* and */.

Names

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

Other operands

arbitrarily long numbers with optional sign and decimal point.
(E)
sqrt (E)

length (E) — number of significant decimal digits

scale (E) — number of digits right of decimal point
L (E , ... , E)

Operators

+ – * / % ^
(% is remainder; ^ is power)

++ –– (prefix and postfix; apply to names)"

== <= >= != < >
= =+ =– =* =/ =% =^

Statements

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

Function definitions

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

Functions in –l math library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc*(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

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
```

```
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

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

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

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

FILES

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

SEE ALSO

dc(1).
CTIX Programmer's Guide, Section 12.

BUGS

No | | yet.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

NAME

bcheck – print out the list of blocks associated with i-node(s)

SYNOPSIS

/usr/local/bin/bcheck [*-i* <*number*>] <special device>

DESCRIPTION

Bcheck will print out a list of all 1024-byte blocks associated with each i-node for a filesystem on a <*special device*>. If the *-i* <*number*> option is given, the printout is restricted to the i-node <*number*>.

EXAMPLES

bcheck /dev/rdsk/c0d0s1

bcheck -i 2 /dev/rdsk/c0d0s3

SEE ALSO

ncheck(1M).

NAME

bcopy – interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

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

The *Bfs* command is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 512 characters, including new-line, per line. *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

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

All address expressions described under *ed* are supported, with the exception of finite range constructions (\{...\}). 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 *regcmp*(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*, except that the default command list for **g** and **v** is the null command, not **p**. 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**. The **Xf** commands may be nested to a depth of 10.

xn List the marks currently in use (marks are set by the **k** commnad).

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

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

(. , .)**xb**/*regular expression*/*label*
A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

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

```
xb/^/ label
```

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

xt *number*

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

xv[*digit*] [*spaces*] [*value*]

The variable name is the specified *digit* following the **xv**. The commands **xv5100** or **xv5 100** both assign the value **100** to the variable **5**. The command **Xv61,100p** assigns the value **1,100p** to the variable **6**. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables **5** and **6**:

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

will all print the first 100 lines.

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

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

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

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.
Another feature of the **xv** command is that the first line of output from a CTIX command can be stored into a variable. The only requirement is that the first character of *value* be an !. For example:

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

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

```
xv7\!date
```

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

xbz *label*

xbn *label*

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

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

xc [*switch*]

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

SEE ALSO

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

DIAGNOSTICS

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

NAME

brc, bcheckrc, rc, powerfail, drvload – system initialization shell scripts

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/rc

/etc/powerfail

/etc/drvload

DESCRIPTION

The *brc*, *bcheckrc*, *rc*, *drvload*, and *powerfail* shell procedures are executed via entries in **/etc/inittab** by *init*(1M). Execpt for *powerfail*, they are run when the system is changed out of *SINGLE USER* mode. *Powerfail* is executed whenever a system power failure is detected.

The *brc* procedure clears the mounted file system table, **/etc/mnttab** (see *mnttab*(4)).

The *bcheckrc* procedure performs all the necessary consistency checks to prepare the system to change into multi-user mode. It actually contains two procedures: an interactive procedure that runs *fsck*(1M) and sets the time; and a noninteractive procedure that only checks the file system. The administrator chooses the interactive or noninteractive procedure by modifying the line in *bcheckrc* that sets the variable **CONSOLE**, **PRESENT** for interactive, **ABSENT** to noninteractive. If the **ABSENT** procedure fails because of file system problems or because it was interrupted from the controlling terminal, it switches the system to state 6, which is normally CTIX Administrator Mode. On a MightyFrame system *bcheckrc* also sets the date to the date currently in the real-time clock.

The *rc* procedure starts all system daemons before the terminal lines are enabled for multi-user mode. In addition, file systems are mounted and accounting, error logging, system activity logging, and printer spooling (if the *lp*(1) system is in use) are activated in this procedure.

The *powerfail* procedure is invoked when the system detects a power failure condition. It calls *halt*(1M) to bring down the system gracefully.

The *drvload* procedure causes any desired device drivers and swap areas to be loaded into the system. The system namelist is rebuilt from **/unix** prior to loading

any drivers. This procedure uses *hinv*(1M) to determine what hardware exists and then loads the appropriate drivers.

These shell procedures, in particular *rc* may be used for several run-level states. The who(1) command may be used to get the run-level information.

FILES

/unix
/etc/log/confile

SEE ALSO

conlocate(1M), date(1), fsck(1M), halt(1), hinv(1M), init(1M), shutdown(1M), who(1), inittab(4), mnttab(4).

NAME

cal – print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *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 83" 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 "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". On weekends "tomorrow" extends through Monday.

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

FILES

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

SEE ALSO

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**] [**–v** [**–t**] [**–e**]] file . . .

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output. Thus:

cat file

prints the file, and:

cat file1 file2 >file3

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

If no input file is given, or if the argument – is encountered, *cat* reads from the standard input file. Output is buffered unless the **–u** option is specified. The **–s** option makes *cat* silent about non-existent files.

The **–v** option causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. Control characters are printed ^*X* (control-*x*); the DEL character (octal 0177) is printed ^?. Non-ASCII characters (with the high bit set) are printed as **M-***x*, where *x* is the character specified by the seven low order bits.

When used with the **–v** option, **–t** causes tabs to be printed as ^I's, and **–e** causes a $ character to be printed at the end of each line (prior to the new-line). The **–t** and **–e** options are ignored if the **–v** option is not specified.

WARNING

Command formats such as

cat file1 file2 >file1

will cause the original data in *file1* to be lost; therefore, take care when using shell special characters.

SEE ALSO

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

NAME

catman – create the cat files for the manual

SYNOPSIS

/etc/catman [**–p**] [**–n**] [**–w**] [sections]

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the nroff input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* will recreate the **/usr/lib/whatis** database.

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

catman 123

will cause the updating to happen only to manual sections 1, 2, and 3.

Options:

–n prevents creation of **/usr/lib/whatis**.

–p prints what would be done instead of doing it.

–w causes only the **/usr/lib/whatis** database to be created. No manual reformatting is done.

FILES

/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/makewhatis	commands to make whatis database

SEE ALSO

man(1).

NAME

cb – C program beautifier

SYNOPSIS

cb [**–s**] [**–j**] [**–l** leng] [file ...]

DESCRIPTION

Cb reads C programs either from its arguments or from the standard input and writes them on the standard output with spacing and indentation that displays the structure of the code. Under default options, *cb* preserves all user new-lines. Under the **–s** flag *cb* canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*. The **–j** flag causes split lines to be put back together. The **–l** flag causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).
The C Programming Language by B. W. Kernighan and D. M. Ritchie.

BUGS

Punctuation that is hidden in preprocessor statements will cause indentation errors.

NAME

cc – C compiler

SYNOPSIS

cc [option] ... file ...

DESCRIPTION

Cc is the CTIX Portable C compiler. It accepts 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 one go.

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*. See *ld*(1) for link editor options and *cpp*(1) for more preprocessor options.

–# Display without execution each command that *cc* generates.

–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 that counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one that automatically calls *monitor*(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).

–g Cause the compiler to generate additional information needed for the use of *sdb*(1).

–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 *cpp*(1) on the named C programs and send the result to the standard output.

-P Run only *cpp*(1) on the named C programs and leave the result on corresponding files suffixed **.i**.

-68020 Generate code for the mc68020 processor.

-68881 Generate code for the mc68881 floating-point coprocessor.

-68010 Generate code for the mc68010 processor.

-68000 Generate code for the mc68000 processor.

-v Verbose. Print pass names as they are performed.

-T Truncate variable names to eight characters. Tell the loader to match eight character names (same as **-G** in the loader).

-w Tell the linker (*ld*) not to print warnings about symbols that partially matched.

The C compiler uses one of three code generators for the 68010, 68020, and 68020/68881. You can select one of these by two mechanisms. The first is to specify the number on the command line. The second is to use the CENVIRON shell variable.

The CENVIRON variable has the following syntax:

CPU=xxxxx,FPU=yyyyy

where CPU indicates the central processor to generate for and FPU indicates the style of floating-point math to use. *xxxxx* may be 68010 or 68020, and *yyyyy* may be 68881 or SOFTWARE. The FPU parameter may be deleted; the default is SOFTWARE. The CENVIRON variable should always be set to the appropriate values in the **.profile** or **.cshrc** files.

The C compiler interprets two shell variables which, along with the CENVIRON variable, allow cross-compilation for any CTIX machine:

LIBROOT This variable is a path which is prepended to normal library names when searching for a library. See also *ld*(1).

INCROOT This variable is a path which is prepended to the **/usr/include** and **/usr/include/sys** directories during include file searches. See also *cpp*(1).

The following options are useful only on systems where work is being done on the C compiler. CTIX normally comes with only one version of the compiler, and that

version works in a single pass. The options below provide for alternative versions of the compiler, including two-pass versions.

-B*string*
Construct pathnames for substitute preprocessor, compiler, assembler and link editor passes by concatenating *string* with the suffixes **cpp** (preprocessor), **c0**, (or **ccom** (compiler first pass), **ccom20**, **ccom20.81**, or **comp** (see under FILES below), **c1** (compiler second pass), **c2** (optimizer) (or **optim**), **as** (assembler), and **ld** (link editor). If *string* is empty it is taken to be **/lib/o**.

-t[**p012al**]
Find only the designated preprocessor, compiler, assembler and link editor passes in the files whose names are constructed by a **-B** option. In the absence of a **-B** option, the *string* is taken to be **/lib/n**. The value **-t** *""* is equivalent to **-tp012**.

-W*c*,*arg1*[,*arg2*...]
Hand off the argument[s] *argi* to pass *c* where *c* is one of [**p012al**] indicating preprocessor, compiler first pass, compiler second pass, optimizer, assembler, or link editor, respectively.

-d This option is no longer allowed because of a conflict of meaning. The **-W** option must be used to specify precisely its destination. To indicate the **-d** option for the link editor, use **-Wl,-d**.

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

Note that modules appear to *ld* in the same order they (or their source code versions) appear to *cc*. Thus a library or object file should appear in the *cc* argument list after any module that refers to it.

The C language standard was extended to include arbitrary length variable names. The option pair

"**-Wp,-T -W0,-XT**" will cause the current compiler to behave the same as previous compilers with respect to the length of variable names.

FILES

file.c	input file
file.o	object file
a.out	linked output
/tmp/ctm*	temporary
/lib/cpp	C preprocessor *cpp*(1)
/lib/ccom	compiler
/lib/ccom20	68020 compiler
/lib/ccom20.81	68020/68881 compiler
/lib/optim	optional optimizer
/bin/as	assembler, *as*(1)
/bin/ld	link editor, *ld*(1)
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	profiling startoff
/lib/libc.a	standard C library, see section (3)
/lib/libp/lib/*.a	profiled versions of libraries

SEE ALSO

adb(1), cpp(1), as(1), ld(1), prof(1), monitor(3C).
CTIX Programmer's Guide, Section 12.
The C Programming Language by B. W. Kernighan and D. M. Ritchie.

NOTES

By default, the return value from a C program is completely random. The only two guaranteed ways to return a specific value are to explicitly call *exit*(2) or to leave the function **main**( ) with a "**return** *expression*;" construct.

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor.

NAME

cd – change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, the value of shell parameter $HOME is used as the new working directory. If *directory* specifies a complete path starting with /, ., .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the $CDPATH shell variable. $CDPATH has the same syntax as, and similar semantics to, the $PATH shell variable. *Cd* must have execute (search) permission in *directory*.

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

SEE ALSO

pwd(1), sh(1), chdir(2).

NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

cdc **–r**SID [**–m**[mrlist]] [**–y**[comment]] files

DESCRIPTION

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

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

If a directory is named, *cdc* behaves as 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:

–r*SID* — Used to specify the *S*CCS *ID*entification (*SID*) string of a delta for which the delta commentary is to be changed.

–m[*mrlist*] — If the SCCS file has the **v** flag set (see *admin*(1)) 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(4).
CTIX Programmer's Guide, Section 9.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cflow – generate C flowgraph

SYNOPSIS

cflow [**–r**] [**–ix**] [**–i_**] [**–dnum**] files

DESCRIPTION

Cflow analyzes a collection of C, YACC, LEX, assembler, and object files and attempts to build a graph charting the external references. Files suffixed in **.y**, **.l**, **.c**, and **.i** are YACC'd, LEX'd, and C-preprocessed (bypassed for **.i** files) as appropriate and then run through the first pass of *lint*(1). (The **–I**, **–D**, and **–U** options of the C-preprocessor are also understood.) Files suffixed with **.s** are assembled and information is extracted (as in **.o** files) from the symbol table. The output of all this non-trivial processing is collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference (i.e., line) number, followed by a suitable number of tabs indicating the level. Then the name of the global (normally only a function not defined as an external or beginning with an underscore; see below for the **–i** inclusion option) a colon and its definition. For information extracted from C source, the definition consists of an abstract type declaration (e.g., **char** *), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only < > is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}
```

```
f()
{
        i = h();
}
```

the command

```
cflow -ix file.c
```

produces the output

```
1       main: int(), <file.c 4>
2               f: int(), <file.c 11>
3                       h: <>
4                       i: int, <file.c 1>
5               g: <>
```

When the nesting level becomes too deep, the **-e** option of *pr*(1) can be used to compress the tab expansion to something less than every eight spaces.

The following options are interpreted by *cflow*:

-r Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

-ix Include external and static data symbols. The default is to include only functions in the flowgraph.

-i_ Include names that begin with an underscore. The default is to exclude these functions (and data if *-ix* is used).

-dnum The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be met with contempt.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO

as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

chmod – change mode

SYNOPSIS

chmod mode files

DESCRIPTION

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

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see *chmod*(2)
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, 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, or sticky); **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with = to take away all permissions.

Multiple symbolic modes separated by commas 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. Only the super-user may set the sticky bit. In order to set the group ID, the group of the file must correspond to your current group ID.

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.

If either command 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.

FILES

/etc/passwd
/etc/group

SEE ALSO

chown(2), group(4), passwd(4).

NAME

chroot – change root directory for a command

SYNOPSIS

/etc/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

clear – clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear prints the string that clears your terminal's screen. The program obtains this string from the *termcap*(5) database, using the **TERM** environment variable to determine the type of terminal.

FILES

/etc/termcap terminal capability data base

SEE ALSO

sh(1), termcap(5).

NAME

clri – clear i-node

SYNOPSIS

/etc/clri file-system i-number ...

DESCRIPTION

Clri 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 *clri* 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(4).

BUGS

If the file is open, *clri* 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* and output resulting from use of the *tbl*(1) preprocessor.

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

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

Unless the **–x** option is given, *col* will convert white space to tabs on output wherever possible to shorten printing time.

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

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

Normally, *col* will ignore any 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* 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**] [**–p**sid] [**–c**list] files

DESCRIPTION

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

The generated shell procedure is written on the standard output.

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

–p*SID* The *S*CCS *ID*entification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

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

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

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

100 * (original – combined) / original

It is recommended that before any SCCS files

are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

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

FILES

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

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sh(1), sccsfile(4).
CTIX Programmer's Guide, Section 9.

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.

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 CTIX system

SYNOPSIS

/etc/config [**–t**] [**–l** file] [**–c** file] [**–m** file] dfile

DESCRIPTION

Config is a program that takes a description of a CTIX system, generates a configuration table file, and generates a hardware interface file. The configuration table file is a C program defining the configuration tables for the various devices on the system. The hardware interface file provides information regarding the interface between the hardware and device handlers.

The **–l** option specifies the name of the hardware interface file; **low.s** is the default.

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 CTIX 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.

First **Part of** *dfile*

Each line contains one field:

devname

where *devname* is the name of the device (as it appears in the **/etc/master** device table).

Second **Part of** *dfile*

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

1. *Root/pipe device specification*

Two lines of three fields each:

root devname minor
pipe devname minor

where *minor* is the minor device number (in octal) of the slice on the winchester.

2. *Swap device specification*

One line that contains five fields, and one line that contains three fields, as follows:

swap devname minor swplo nswap

where *swplo* is the lowest disk block (decimal) in the swap area and *nswap* is the maximum number of 1K-byte disk blocks (decimal) in the swap area. The kernel sizes the actual swap area size and configures itself for up to this maximum.

3. *Parameter specification*

There are any number of lines of two fields each, chosen from the following list. *Number* is decimal. This list is not complete; parameters not on the list either must not be changed or have no effect.

```
buffers number /* number of 1024-byte file system
                        caching buffers */
dmmxsz number
                /* max number of pages per
                        loadable driver */
inodes  number /* max open inodes in system */
files   number /* max open files in system */
nflocks number /* max locks active in system */
mounts number /* max file systems mounted */
regions number /* total number of regions in system */
procs   number /* max processes in system */
maxproc number
                /* max processes per user ID */
maxfsiz number /* ulimit default in 512-byte blocks */
maxumem number
                /* max number of pages per process */
cbufsize number
                /* console circular buffer size in bytes */
mesg    0 or 1  /* configure for messages */
msgmax number
                /* max chars in a message */
msgmni number
                /* max active message queues */
```

```
msgmnb number
                    /* max total chars in message
                            queues */
msgtql  number /* max messages in system */
msgssz  number
msgseq  number /* msgssz * msgseq = number bytes of
                            system buffering */
nlldrv  number /* max number of loadable drivers */
sem     0 or 1  /* configure for semaphores */
semmni number /* max active semaphores */
semmns number /* max semaphores in system */
semmsi number /* max semaphores per ID */
semopm number
                    /* max operations per semop call */
semume number /* max undo structures per process */
semmnu  number
                    /* max undo structures in
                            system */
shmem  0 or 1   /* configure for shared memory */
shmmax number
                    /* max bytes in a shared segment */
shmmin number /* min bytes in a shared segment */
shmmni number /* max active shared segments */
shmseg  number /* max attached segments per process */
shmbrk number /* gap in pages between data and
                            shared memory */
debugger 0 or 1 /* configure low-level
                    kernel debugger */
```

Certain parameters if set to 0 will allow the kernel to autoconfigure. Procs, regions, clists, i-nodes, files, and buffers are autoconfigurable. The number of procs is based on the number of users; regions, i_nodes, and files are based on the number of procs. The number of clists is based on the number of serial and cluster ports. The number of buffers is based on the amount of physical memory. Any or all of these may be overridden. Maxumem may also be set to 0, in which case it floats between 1M byte plus one quarter of the total swap space.

EXAMPLE

To configure a system with the following devices:

Onboard quarter-inch tape
Onboard ST506 disks (root)
Interphase SMD disk controller
RS-232-C (any number of ports)
one parallel line printer
root device is a winchester (drive 0, section 1)
pipe device is a winchester (drive 0, section 1)

```
swap device is a winchester (drive 0, section 2),
        with a swplo of 1 and an nswap of 8000
number of buffers is 100
number of processes is 100
maximum number of processes per user ID is 25
number of mounts is 6
number of inodes is 100
number of files is 120
number of character buffers is 64
messages are to be included
semaphores are to be included
```

The actual system configuration would be specified as follows:

```
diskonbd
Vsmd3200
serial
qic
console
plp
root    diskonbd        01
pipe    diskonbd        01
swap    diskonbd        02      0       8000
* Comments may be inserted in this manner
buffers 100
procs   100
maxproc 25
mounts  6
inodes  100
files   120
mesg    1
sema    1
clists  64
```

FILES

/etc/master	default input master device table
low.s	default output hardware interface file
conf.c	default output configuration table file

SEE ALSO

ldeeprom(1M), master(4).

DIAGNOSTICS

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

BUGS

The –t option does not know about devices that have aliases.

FILES

/dev/console
/etc/log/confile

SEE ALSO

conlocate(1M), syslocal(2).

WARNING

Normal system processing is suspended while the kernel debugger is active.

NAME

disk – general disk driver

SYNOPSIS

#include <sys/types.h>
#include <sys/gdisk.h>
#include <sys/gdioctl.h>

DESCRIPTION

The files
/dev/rdsk/c0d0s0
through
/dev/rdsk/c*x*d*x*s*x*
and
/dev/dsk/c0d0s0
through
/dev/dsk/c*x*d*x*s*x*
refer to CTIX device names and slices, where **c***x* is the controller number, **d***x* is the drive number, **s***x* is the slice number, and *x* is a hexadecimal digit. An **r** in the name indicates the character (raw) interface,

MightyFrame and MiniFrame format a disk with 512-byte physical sectors. Winchester disks have 17 physical sectors per track. SMD drives have 33 to 65 physical sectors per track.

Block input/output uses 1024-byte logical blocks. Winchester disks have 8 logical blocks on each track, with the leftover physical block available as an alternate for a bad block. SMD disks have 16 to 32 logical blocks on each track, with the leftover physical block available as an alternate for a bad block.

Logical block zero contains the *Volume Home Block*, which describes the disk. The following structure defines the volume home block.

```
struct vhbd {
    uint    magic;          /* Mitiframe disk format code */
    int     chksum;         /* adjustment so 32 bit sum starting
                               from magic for 1K bytes sums to -1 */
    struct gdswprt dsk;     /* specific description of this disk */
    struct partit partab[MAXSLICE];/* partition table */
    struct resdes{          /* reserved area special files */
            daddr_t blkstart; /* start logical block # */
            ushort nblocks; /* length in logical blocks
                               (zero implies not present) */
    } resmap[8];
/*  resmap consists of the following entries:
 *          loader area
 *          bad block table
```

NAME

conlocate – locate a terminal to use as the virtual system console

SYNOPSIS

/etc/conlocate [**–r**] [**–i***n*] [**–t**]

DESCRIPTION

Conlocate searches for a terminal to use as the system console, **/dev/syscon**. It scans **/etc/inittab** for terminals that get a *getty*(1M) in state 6, and spawns children to monitor the terminals for attempted logins. Each child does all the I/O control and login verification of the *getty-login* sequence, but only root is actually permitted to log in. The first terminal to have *root* log in has its tty linked to **/dev/syscon**. *Conlocate* then writes the new virtual system console's communication options, which have just been set from the values in **/etc/gettydefs**, to its own standard output, using *stty*(1) **–g** format.

Conlocate understands the following options:

–r If **/dev/syscon** exists and is openable, exit without scanning for a new one.

–i*n* Scan run level *n* instead of run level 6.

–t Begin by monitoring for logins on the existing **/dev/syscon**. If root logs in at that terminal within 20 seconds, abandon the search for another console.

FILES

/dev/syscon	virtual system console
/etc/inittab	definitions of operating states
/etc/gettydefs	communication options

SEE ALSO

init (1M), stty(1), gettydefs(4), inittab(4), termio(7).

WARNING

Beware of collision with other processes that might be trying to open the same terminals, especially *gettys* spawned by *init*.

NAME

convert – convert object and archive files to common formats

SYNOPSIS

convert [**–5**] infile outfile

DESCRIPTION

Convert transforms input *infile* to output *outfile*. *Infile* must be different from *outfile*. The **–5** option causes *convert* to work exactly as it did for UNIX system release 5.0. *Infile* may be any one of the following:

1) a pre-UNIX system 5.0 object file or link-edited (a.out) module (only with the **–5** option),

2) a pre-UNIX system 5.0 archive of object files or link edited (a.out) modules (only with the **–5** option), or

3) a UNIX system 5.0 archive file (without the **–5** option).

Convert will transform *infile* to one of the following (respectively):

1) an equivalent UNIX system 5.0 object file or link edited (a.out) module (with the **–5** option),

2) an equivalent UNIX system 5.0 archive of equivalent object files or link edited (a.out) modules (with the **–5** option), and

3) an equivalent UNIX system 5.0 release 2.0 portable archive containing unaltered members (without the **–5** option).

All other types of input to the *convert* command will be passed unmodified from the input file to the output file (along with appropriate warning messages). When transforming archive files with the **–5** option, the *convert*(1) command will inform the user that the archive symbol table has been deleted. To generate an archive symbol table, this archive file must be transformed again by *convert* without the **–5** option to create a UNIX system 5.0 archive file. Then the archive symbol table may be created by executing the *ar*(1) command with the **ts** option. If a UNIX system 5.0 archive with an archive symbol table is being transformed, the archive symbol table will automatically be converted.

FILES

/tmp/conv*

SEE ALSO

ar(1), arcv(1), a.out(4), ar(4).

NAME

cp, ln, mv – copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target
ln [**–f**] file1 [file2 ...] target
mv [**–f**] file1 [file2 ...] target

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same (take care when using *sh*(1) and *csh*(1) metacharacters). If *target* is a directory, then one or more files are copied (linked, moved) to that directory. If *target* is a file, its contents are destroyed.

If *mv* or *ln* determines that the mode of *target* forbids writing, it will print the mode (see *chmod*(2)), ask for a response, and read the standard input for one line (if the standard input is a terminal); if the line begins with **y**, the *mv* or *ln* occurs, if permissible; if not, the command exits. No questions are asked and the *mv* or *ln* is done when the **–f** option is used or if the standard input is not a terminal.

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; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

When using *cp*, if *target* is not a file, a new file is created which has the same mode as *file1* except that the sticky bit is not set unless you are super-user; the owner and group of *target* are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, nor group. The last modification time of *target* (and last access time, if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

SEE ALSO

cpio(1), rm(1), chmod(2).

WARNING

When the destination of a copy is a file that already exists, *cp* will try to overwrite it, not remove it; this preserves the destination files ownership, and so forth. If the destination file has an ownership you do not want, remove it before doing the copy.

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 copying 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 [**acBQv**]

cpio –i [**BQcdmrtuvfsSb6**] [patterns]

cpio –p [**adlmuv**] 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. Output is padded to a 512-byte boundary.

Cpio –i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio –o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh*(1). In *patterns*, meta-characters **?**, *****, and [. . .] match the slash / character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous **cpio –o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous **cpio –o**.

Cpio –p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

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??** or raw floppy disks).

Q Input/output is to be blocked 65,536 bytes to the record. Works like **–B** option, with which it is mutually exclusive. The **–Q** option optimizes quarter-inch tape access.

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.

f Copy in all files except those in *patterns*.

s Swap bytes. Use only with the −**i** option.

S Swap halfwords. Use only with the −**i** option.

b Swap both bytes and halfwords. Use only with the −**i** option.

6 Process an old (i.e., UNIX System *Sixth* Edition format) file. Only useful with −**i** (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 . -depth -print | cpio -pdl newdir
```

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

```
find . -cpio /dev/rmt0
```

SEE ALSO

ar(1), find(1), cpio(4).

NOTES

The −**Q** option can be used with the −**p** option to improve performance, but at the penalty of using more memory.

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

cpp – the C language preprocessor

SYNOPSIS

/lib/cpp [option ...] [ifile [ofile]]

DESCRIPTION

Cpp is the C language preprocessor which is invoked as the first pass of any C compilation using the *cc*(1) command. Thus the output of *cpp* is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, *cpp* and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of *cpp* other than in this framework is not suggested. The preferred way to invoke *cpp* is through the *cc*(1) command, since the functionality of *cpp* may someday be moved elsewhere. See *m4*(1) for a general macro processor.

Cpp optionally accepts two file names as arguments. *Ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to *cpp* are recognized:

–P Preprocess the input without producing the line control information used by the next pass of the C compiler.

–C By default, *cpp* strips C-style comments. If the **–C** option is specified, all comments (except those found on *cpp* directive lines) are passed along.

–U*name*

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, u3b5, vax, mc68k, mc68000, mc68010, mc68020
system variants:	RES, RT
line(1):	lint

–D*name*

–D*name*=*def*

Define *name* as if by a **#define** directive. If no =*def* is given, *name* is defined as 1. The **–D**

option has lower precedence that the **-U** option. That is, if the same name is used in both a **-U** option and a **-D** option, the name will be undefined regardless of the order of the options.

-T Preprocessor symbols are no longer restricted to eight characters. The **-T** option forces *cpp* to use only the first eight characters for distinguishing different preprocessor names. This behavior is the same as previous preprocessors with respect to the length of names and is included for backward compatability.

-I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in " " will be searched for first in the directory of the file with the **#include** line, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the file with the **#include** line is not searched. By default, *cpp* searches for the name enclosed in < > in **/usr/include**; however, if the shell variable INCROOT is set, *cpp* prepends the value of INCROOT to the standard list. This is particularly useful for cross-machine compilation.

Two special names are understood by *cpp*. The name __LINE__ is defined as the current line number (as a decimal integer) as known by *cpp*, and __FILE__ is defined as the current file name (as a C string) as known by *cpp*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines begun by #. Any number of blanks and tabs are allowed between the # and the directive. The directives are:

#define *name token-string*
Replace subsequent instances of *name* with *token-string*.

#define *name*(*arg*, ..., *arg*) *token-string*
Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma-separated set of tokens, and a) by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the

comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#undef *name*
Cause the definition of *name* (if any) to be forgotten from now on.

#include *"filename"*
#include *<filename>*
Include at this point the contents of *filename* (which will then be run through *cpp*). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See the –I option above for more detail.

#line *integer-constant "filename"*
Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If *"filename"* is not given, the current file name is unchanged.

#endif
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name*
The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name*
The lines following will not appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*
Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary –, **!**, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a

unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output. And vice versa.

The test directives and the possible **#else** directives can be nested.

FILES

/usr/include	standard directory for **#include** files

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

The error messages produced by *cpp* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When new-line characters were found in argument lists for macros to be expanded, previous versions of *cpp* put out the new-lines as they were found and expanded. The current version of *cpp* replaces these new-lines with blanks to alleviate problems that the previous versions had when this occurred.

NAME

cpset – install object files in binary directories

SYNOPSIS

cpset [**-o**] object directory [mode owner group]

DESCRIPTION

Cpset is used to install the specified *object* file in the given *directory*. The *mode*, *owner*, and *group*, of the destination file may be specified on the command line. If this data is omitted, two results are possible:

If the user of *cpset* has administrative permissions (that is, the user's numerical ID is less than 100), the following defaults are provided:

mode – 0755

owner – bin

group – bin

If the user is not an administrator, the default, owner, and group of the destination file will be that of the invoker.

An optional argument of **–o** will force *cpset* to move *object* to OLD*object* in the destination directory before installing the new object.

For example:

cpset echo /bin 0755 bin bin

cpset echo /bin

cpset echo /bin/echo

All the examples above have the same effect (assuming the user is an administrator). The file **echo** will be copied into **/bin** and will be given **0755, bin, bin** as the mode, owner, and group, respectively.

Cpset utilizes the file **/usr/src/destinations** to determine the final destination of a file. The locations file contains pairs of pathnames separated by spaces or tabs. The first name is the "official" destination (for example: **/bin/echo**). The second name is the new destination. For example, if *echo* is moved from **/bin** to **/usr/bin**, the entry in **/usr/src/destinations** would be:

/bin/echo /usr/bin/echo

When the actual installation happens, *cpset* verifies that the "old" pathname does not exist. If a file exists at that location, *cpset* issues a warning and continues. This

file does not exist on a distribution tape; it is used by sites to track local command movement. The procedures used to build the source will be responsible for defining the "official" locations of the source.

Cross Generation

The environment variable **ROOT** will be used to locate the destination file (in the form **$ROOT/usr/src/destinations**). This is necessary in the cases where cross generation is being done on a production system.

SEE ALSO

install(1M), make(1).

NAME

crash – examine system images

SYNOPSIS

/etc/crash [system] [namelist]

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 and a *namelist* file to be used for symbol values.

The default values are **/dev/kmem** and **/unix**; 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 default process is set to be that of the process running at the time of the crash. This is determined by a value stored in a fixed location by the dump mechanism.

The *system* image may be **/dev/kmem**, regular files, or partition zero of a disk.

COMMANDS

Input to *crash* is typically of the form:

command [options] [structures to be printed].

When allowed, *options* will modify the format of the printout. 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 standard format.

In general, those commands that perform I/O with addresses assume hexadecimal on 32-bit machines and octal on 16-bit machines.

The current repertory consists of:

user [list of process table entries]

Aliases: **uarea**, **u_area**, **u**.

Print the user structure of the named process as determined by the information contained in the process table entry. If no entry number is given, the information from the last executing process will be printed. Swapped processes produce an error message.

trace [**–r**] [list of process table entries]
Aliases: **t**.
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 **kfp**. Otherwise the trace starts at the value of the **fp** stored in **u_rsav**. If no entry number is given, the information from the last executing process will be printed.

kfp [stack frame pointer]
Aliases: **r6**, **fp**.
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 [list of process table entries]
Aliases: **stk**, **s**, **kernel**, **k**.
Format a dump of the kernel stack of a process. The addresses shown are virtual system data addresses rather than true physical locations. If no entry number is given, the information from the last executing process will be printed.

proc [**–r**] [list of process table entries]
Aliases: **ps**, **p**.
Format the process table. The **–r** option causes only runnable processes to be printed. The **–** alone generates a longer listing.

i-node [**–**] [list of inode table entries]
Aliases: **ino**, **i**.
Format the i-node table. The **–** option will also print the i-node data block addresses.

file [list of file table entries]
Aliases: **files**, **f**.
Format the file table.

lck Aliases: **l**
Print the active and sleep record lock tables; also verify the correctness of the record locking linked lists.

mount [list of mount table entries]
Aliases: **mnt**, **m**.
Format the mount table.

tty [type] [**–**] [list of tty entries]
Aliases: **term**, **pt**, **gt**, **ser**.
Print the tty structures. The *type* argument determines which structure will be used (such as **pt** or **ser**; the last *type* is remembered.) The **–**

option prints the *stty*(1) parameters for the given line.

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

var Aliases: **tunables**, **tunable**, **tune**, **v**.
Print the tunable system parameters.

buf [options] [list of buffer headers]
Aliases: **hdr**, **bufhdr**.
Format the system buffer headers. With no parameters, all buffer headers are shown. With no options but a list of indexes or addresses, only the specified buffer headers are shown. With an option and a single index or address, the chain of buffer headers beginning at the specified buffer header is shown. Various option show various chains:

- **−a** Trace the available chain both ways by following both the *av_forw* and *av_back* fields in the headers.
- **−af** Trace the available chain by following the *av_forw* fields.
- **−ab** Trace the available chain by following the *av_back*. fields.
- **−f** Trace the chain for the device by following the *b_forw* fields. −**b** Trace the chain for the device by following the *b_back* fields.
- *−n* Follow the *b_forw*, *b_back*, *av_forw*, and *av_back* fields for *n* headers each.
- **−** Follow the *b_forw*, *b_back*, *av_forw*, and *av_back* fields all the way through their chains.

buffer [format] [list of buffers]
Alias: **b**.
Print the data in a system buffer according to *format*. If *format* is omitted, the previous *format* is used. Valid formats include **decimal**, **octal**, **hex**, **character**, **byte**, **directory**, **i-node**, and **write**. The last creates a file in the current directory (see *FILES*) containing the buffer data.

callout
Aliases: **calls**, **call**, **c**, **timeout**, **time**, **tout**.
Print all entries in the callout table.

region [region table number | region table address]
Prints region table. Region table address must be of the form 0x ... .

fcallout
Aliases: **fcalls**, **fcall**, **fc**, **ftimeout**, **ftime**, **ftout**.
Print all entries in the fcallout 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.

cblk [-]
Format the cblock table. The - option checks cblock usage.

pm [symbol name or address] [count] [format]

od [symbol name or address] [count] [format]
Aliases: **dump**, **rd**.
Dump *count* data values starting at the symbol value or address given according to *format*. **Od** dumps virtual addresses; **pm** dumps physical addresses. Allowable formats are **octal**, **longoct**, **decimal**, **longdec**, **character**, **hex**, or **byte**.

shm [-] [list of shared memory header table entries]
Format the shared memory header table. If the - option is used, also display information about the last change, the last *shmop*, and attached processes.

shminfo
Display the system's shared memory information structure.

msg [-] [list of ipc message queue header table entries]
Format the ipc message queu header table. If the - option is used, also display information

about the last change, the last *msgop*, and any messages on the queue contained in the message headers.

msginfo
Print the system message information structure.

msgtext [format]
[list of ipc message queue header table entries]
Print the text of the messages on a queue according to format. If *format* is omitted, the previous format is used.

! Escape to shell.

q Exit from *crash*.

notify [list of notification table entries]
Print a requested notification.

unotify [list of process table entries]
Print queued notifications for given process.

? [start letter]
Print synopsis of commands. Optional start letter prints only those commands beginning with that letter.

pfdat [list of page frame numbers]
Alias: pf
Print information about a physical page of memory.

pfree [-] [list of page frame numbers]
If no options are given, print out number of pages on free list. With the option '-', print all pages on free list. Giving a specific page number simply reports whether that page is on the free list.

phash [list of hash slots]
Alias: ph
Print hash lists of physical pages. With no arguments, print all hash lists with their respective pages.

pregion [list of process table entries]
Alias: prg
Print currently attached regions of a process.

w Print toggle warning. Primarily useful in tracking virtual to physical address translations.

ALIASES
There are built-in aliases for many of the *formats* as well as those listed for the commands. Some of them are:

byte	b.
character	char, c.
decimal	dec, ec.

FORMATS AND FORMAT ALIASES

Here are the standard formats and format aliases:

Format	*Meaning*	*Aliases*
byte	byte	**b**
bytedec	byte of decimal	**bd**
byteoct	byte of octal	**bo**
bytehex	byte of hexadecimal	**bh, bx**
character	ASCII character	**char, c**
worddec	2 bytes	**wd, decimal, dec, e**
wordoct	2 bytes	**wo, octal, oct, o**
wordhex	2 bytes	**wx, wh**
longdec	4 bytes	**ld, D**
longoct	4 bytes	**lo, O**
longhex	4 bytes	**lx, X, hex, h, x**
directory	directory	**direct, dir, d**
inode	inode	**ino, i**
write	write	**w**

FILES

/usr/include/sys/*.h	header files for table and structure information
/dev/kmem	default system image file
/unix	default namelist file
buf.#	files created containing buffer data

SEE ALSO

mount(1M), nm(1), ps(1), sh(1), stty(1).

NAME

cron – clock demon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to the instructions found in crontab files; users can submit their own crontab file via the *crontab* command. Commands which are to be executed only once may be submitted via the *at command.* Since *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*(1M)).

Cron only examines crontab files and at command files during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

FILES

/usr/lib/cron
main cron directory
/usr/lib/cron/log
accounting information
/usr/spool/cron
spool area

SEE ALSO

at(1), crontab(1), init(1M), sh(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

DIAGNOSTICS

A history of all actions taken by *cron* are recorded in **/usr/lib/cron/log**.

NAME

crontab - user crontab file

SYNOPSIS

crontab [file]
crontab -r
crontab -l

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The -r option removes a user's crontab from the crontab directory. *Crontab* -l will list the crontab file for the invoking user.

A user is permitted to use *crontab* if their name appears in the file **/usr/lib/cron/cron.allow.** If that file does not exist, the file **/usr/lib/cron/cron.deny** is checked to determine if the user should be denied access to *crontab*. If neither file exists, only root is allowed to submit a job. If **cron.deny** exists and is emtpy, global usage is permitted. If **cron.allow** exists and is empty, no usage is permitted. If **cron.allow** exists, **cron.deny** is ignored. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0-59),
hour (0-23),
day of the month (1-31),
month of the year (1-12),
day of the week (0-6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values), or a list of elements separated by commas. An element is either a number, or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated

to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your **$HOME** directory with an **arg0** of **sh.** Users who desire to have their *.profile* executed must explicitly do so in the crontab file. *Cron* supplies a default environment for every shell, defining HOME, LOGNAME, SHELL(=**/bin/sh**), TZ, and **PATH(=:/bin:/usr/bin:/usr/local/bin)**.

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

cron(1M), sh(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

BUGS

Crontab runs *sh* even if your login shell is *csh*.

NAME

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

SYNOPSIS

csh [**–cefinstvVxX**] [arg ...]

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see **History Substitutions**) job control facilities (see **Jobs**) and a C-like syntax.

An instance of *csh* begins by executing commands from the file **.cshrc** in the *home* directory of the invoker. If this is a login shell, then it also executes **/etc/cprofile** and commands from the file **.login** there. It is typical for users on crt's to put *tset*(1) in their **.login** files.

In the normal case, the shell will then begin reading commands from the terminal, prompting with '% '. Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates, it executes commands from the file **.logout** in the user's home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters **&** | **;** < > (form separate words. If doubled in **&&**, | |, <<or>> these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with \. A newline preceded by a \ is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, ', `, or " form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of ' or " characters a newline preceded by a \ gives a true newline character.

When the shell's input is not a terminal, the character **#** introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by \ and in quotations using `, ', and ".

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by the | character forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ';', and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an &.

Any of the above may be placed in ( ) to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with | | or && indicating, as in the C language, that the second is to be executed only if the first succeeds or fails, respectively. (See *Expressions.)*

Jobs

The shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the *jobs* command, and assigns them small integer numbers. When a job is started asynchronously with &, the shell prints a line which looks like:

[1] 1234

indicating that the jobs which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + and the previous job with a −.

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify,* the shell will notify you immediately of changes of status in background jobs. There is also a shell command *notify* which marks a single process so that its status changes will be immediately reported. By default *notify* marks the current process; simply say 'notify' after starting a background job to mark it.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character **!** and may begin **anywhere** in the input stream (with the proviso that they **do not** nest.) This **!** may be preceded by a \ to prevent its special meaning; for convenience, a **!** is passed unchanged when it is followed by a blank, tab, newline, = or (. (History substitutions also occur when an input line begins with ↑. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of which is controlled by the ***history*** variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the ***history*** command:

```
 9  write michael
10  ex write.c
11  cat oldwrite.c
12  diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing **!** in the prompt string.

With the current event 13 we can refer to previous events by event number **!11**, relatively as in **!-2** (referring to the same event), by a prefix of a command word as in **!d** for event 12 or **!wri** for event 9, or by a string contained in a word in the command as in **!?mic?** also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a

special case !! refers to the previous command; thus !! alone is essentially a *redo*.

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

0	first (command) word
n	*n* 'th argument
↑	first argument, i.e., 1
$	last argument
%	word matched by (immediately preceding) ?*s*? search
x–*y*	range of words
–*y*	abbreviates **0**–*y*
*	abbreviates ↑–$, or nothing if only one word in event
*x**	abbreviates *x*–$
x–	like *x** but omitting word '$'

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ↑, $, *, –, or %. After the optional word designator can be placed a sequence of modifiers, each preceded by a :. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing **.xxx** component, leaving the root name.
e	Remove all but the extension **.xxx** part.
s/*l*/*r*/	Substitute *l* for *r*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g., **g&**.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like **q**, but break into words at blanks, tabs and newlines.

Unless preceded by a 'g' the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of /; a \ quotes the delimiter into the *l* and *r* strings. The

& character in the right hand side is replaced by the text from the left. A \ quotes & also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in **!?***s*?. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing '?' in a contextual scan.
A history reference may be given without an event specification, e.g. '!$'. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus **!?foo?↑ !$** gives the first and last arguments from the command matching **?foo?**.
A special abbreviation of a history reference occurs when the first non-blank character of an input line is a '↑'. This is equivalent to **!:s↑** providing a convenient shorthand for substitutions on the text of the previous line. Thus **↑lb↑lib** fixes the spelling of **lib** in the previous command. Finally, a history substitution may be surrounded with { and } if necessary to insulate it from the characters which follow. Thus, after **ls −ld ~paul** we might do **!{l}a** to do **ls**−ld ~paula, while **!la** would look for a command starting **la**.

Quotations with ' and "

The quotation of strings by ' and " can be used to prevent all or some of the remaining substitutions. Strings enclosed in ' are prevented any further interpretation. Strings enclosed in " may be expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitition* below) does a quoted string yield parts of more than one word; ' quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for **ls** is **ls −l** the command **ls /usr** would map to **ls −l /usr**, the argument list here being

undisturbed. Similarly if the alias for **lookup** was **grep !↑ /etc/passwd** then **lookup bill** would map to **grep bill /etc/passwd**.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can **alias print 'pr \!* | lpr'** to make a command which *pr's* its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the **-v** command line option.

Other operations treat variables numerically. The **@** command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by $ characters. This expansion can be prevented by preceding the $ with a \ except within **s** where it **always** occurs, and within 's where it **never** occurs. Strings quoted by ` are interpreted later (see *Command substitution* below) so $ substitution does not occur there until later, if at all. A $ is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately.

Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in " or given the **:q** modifier, the results of variable substitution may eventually be command and filename substituted. Within a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the **:q** modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$name
${name}
: Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.
If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}
: May be used to select only some of the words from the value of *name*. The selector is subjected to '$' substitution and may consist of a single number or two numbers separated by a –. The first word of a variables value is numbered **1**. If the first number of a range is omitted it defaults to **1**. If the last member of a range is omitted it defaults to **$#name**. The selector * selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
: Gives the number of words in the variable. This is useful for later use in a '[selector]'.

$0
Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}
Equivalent to **$argv[number]**.

$*
Equivalent to **$argv[*]**.

The modifiers **:h**, **:t**, **:r**, **:q**, and **:x** may be applied to the substitutions above as may **:gh**, **:gt**, and **:gr**. If braces **{ }** appear in the command form then the modifiers must appear within the braces. The current implementation allows only one **:** modifier on each $ expansion.

The following substitutions may not be modified with **:** modifiers.

$?name
${?name}
Substitutes the string **1** if name is set, **0** if it is not.

$?0
Substitutes **1** if the current input filename is known, **0** if it is not.

$$
Substitute the (decimal) process number of the (parent) shell.

$<
Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in `. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then

replacing the original string. Within "s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters *, ?, [, or { or begins with the character ~, then that word is a candidate for filename substitution, also known as "globbing." This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters *, ?, and [imply pattern matching, the characters ~, and { being more akin to abbreviations.

In matching filenames, the character . at the beginning of a filename or immediately following a /, as well as the character / must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence [...] matches any one of the characters enclosed. Within [...], a pair of characters separated by – matches any character lexically between the two.

The character ~ at the beginning of a filename is used to refer to home directories. Standing alone, i.e., ~, it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and – characters the shell searches for a user with that name and substitutes their home directory; thus **~ken** might expand to **/usr/ken** and **~ken/chmach** to **/usr/ken/chmach**. If the character ~ is followed by a character other than a letter or / or appears not at the beginning of a word, it is left undisturbed.

The metanotation **a{b,c,d}e** is a shorthand for **abe ace ade** . Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus **~source/s1/{oldls,ls}.c** expands to **/usr/source/s1/oldls.c /usr/source/s1/ls.c** whether or not these files exist without any chance of error if the home directory for **source** is **/usr/source**. Similarly **../{memo,*box}** might expand to **../memo ../box**

../mbox . (Note that **memo** was not sorted with the results of matching ***box**.) As a special case {,} and {} are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

< name
: Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word
: Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, ', ", or ` appears in *word* variable and command substitution is performed on the intervening lines, allowing \ to quote $, \, and `. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name
>! name
>& name
>&! name
: The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, its is truncated, its previous contents being lost.

  If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g., a terminal or **/dev/null**) or an error results. This helps prevent accidental destruction of files. In this case the ! forms can be used and suppress this check.

  The forms involving & route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as < input filenames are.

>> name
>>& name
>>! name
>>&! name
: Uses file *name* as standard output like > but

places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the ! forms is given. Otherwise similar to >.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form |& rather than just |.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit*, *if*, and *while* commands. The following operators are available:

| | || && | ↑ & == != =~ !~ <= >=
< > << >> + - * / % ! ~ ( )

Here the precedence increases to the right, ==, !=, =~, and !~, <=, >=, <, and >, <<, and >>, +, and -, *, /, and % being, in groups, at the same level. The ==, !=, =~, and !~ operators compare their arguments as strings; all others operate on numbers. The operators =~ and !~ are like != and == except that the right hand side is a *pattern* (containing, e.g., *'s, ?'s and instances of [...]) against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings that begin with **0** are considered octal numbers. Null or missing arguments are considered **0**. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser (& |, <, >, (,)) they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in { and } and file enquiries of the form *-l name* where *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e., 0. Command executions succeed, returning true, i.e., 1, if the command exits with status 0, otherwise they fail, returning false, i.e., 0. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach*, *switch*, and *while* statements, as well as the *if-then-else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last, it is executed in a subshell.

alias
alias name
alias name wordlist
The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name; wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*.

break
Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a *switch*, resuming after the *endsw*.

case label:
A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name
Change the shells working directory to directory *name*. If no argument is given then change to the home directory of the user.
If *name* is not found as a subdirectory of the current directory (and does not begin with /, ./ or ../), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with /, then this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:
Labels the default case in a *switch* statement. The default should come after all *case* labels.

dirs
Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist
echo **-n** wordlist
The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the **-n** option is specified. Note that this differs from **/bin/echo**.

else
end
endif
endsw
See the description of the *foreach, if, switch,* and *while* statements below.

eval arg ...
(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of using *eval*.

exec command
The specified command is executed in place of the current shell.

exit
exit(expr)
The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

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

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with **?** before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist
Like *echo* but no \ escapes are recognized and

words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word
The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

history
history *n*
history **-r** *n*
Displays the history event list; if *n* is given only the *n* most recent events are printed. The **-r** option reverses the order of printout to be most recent first rather than oldest first.

if (expr) command
If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif
If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else.)*

jobs
jobs **-l**
Lists the active jobs; given the **-l** options lists process id's in addition to the normal information.

kill %job
kill –sig %job ...
kill pid
kill –sig pid ...
kill –l

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix "SIG"). The signal names are listed by **kill** –l. There is no default, saying just **kill** does not send a signal to the current job.

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

Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

Resources controllable currently include *filesize* (the largest single file which can be created).

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. The default scale is 'k' or 'kilobytes' (1024 bytes); a scale factor of 'm' or 'megabytes' may also be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

login

Terminate a login shell, replacing it with an instance of **/bin/login.** This is one way to log off, included for compatibility with *sh*(1).

logout

Terminate a login shell. Especially useful if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using **nice –number ...**.

Command is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

nohup
nohup command
The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with & are effectively *nohup'ed.*

notify
notify %job ...
Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

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

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd
popd +n
Pops the directory stack, returning to the new top directory. With a argument '+*n*' discards the *n* th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd
pushd name
pushd +n
With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (a la *cd)* and pushes the old current working directory (as in *cwd)* onto the directory stack. With a

numeric argument, rotates the *n* th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index'th* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variables USER, TERM, PATH, and CDPATH are automatically imported to and exported from the *csh* variables *user*, *term*, *path*, and *cdpath;* there is no need to use *setenv* for these.

shift
shift variable
The members of *argv* are shifted to the left, discarding *argv[1]*. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name
The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands.

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw
Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters *, ?, and [...] may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command
With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

ulimit **-f** *n*
imposes a size limit of *n*.
-f imposes a size limit of *n* blocks on files written

by child processes (files of any size may be read). With no argument, the current limit is printed.

umask
umask value
The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be *unaliased.*

unhash
Use of the internal hash table to speed location of executed programs is disabled.

unset pattern
All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset *'; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset.*

unsetenv pattern
Removes all variables whose name match the specified pattern from the environment. See also the *setenv* command above and *printenv*(1).

wait
All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)
...
end
While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

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

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains <, >, *&*, or |, then at least this part of the expression must be placed within ( ). The third form assigns the value of *expr* to the *index'th* argument of *name*. Both *name* and its *index'th* component must already exist. Beware of conflicts between the kill character and this use of **@**.

The operators *=, +=, etc., are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix ++ and -- operators increment and decrement *name* respectively, i.e., **@** i++.

Pre-defined and environment variables

The following variables have special meaning to the shell. Of these, *argv, cwd, home, path, cdpath, prompt, shell* and *status* are always set by the shell. Except for *cwd* and *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file **.cshrc** as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it.

argv — Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e., **$1** is replaced by **$argv[1]**, etc.

cdpath — Gives a list of alternate directories searched to find subdirectories in *chdir* commands.

cwd — The full pathname of the current directory.

echo Set when the **-x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character !. The second character of its value replaces the character ↑ in quick substitutions.

history Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of ***history*** may run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the invoker, initialized from the environment. The filename expansion of ~ refers to this variable.

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by code-D's.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says 'New mail in *name*' when

there is mail in the file *name*.

noclobber As described in the section on *Input/output*, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that >> redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e., **echo** [still gives an error.

notify If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a prompt.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full path names will execute. The usual search path is ., **/bin**, and **/usr/bin**, but this may vary from system to system. For the super-user the default search path is **/etc**, **/bin**, and **/usr/bin**. A shell which is given neither the **-c** nor the **-t** option will normally hash the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be found.

prompt The string which is printed before each command is read from an interactive terminal input. If a ! appears in the string it will be replaced by the current event number unless a preceding \ is

given. Default is **%**, or **#** for the super-user.

shell The file in which the shell object code resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Non-builtin Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status **1**, all other builtin commands set status **0**.

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose Set by the **−v** command line option, causes the words of each command to be printed after history substitution.

Non-builtin command execution

When a command to be executed is found to not be a builtin command, the shell attempts to execute the command via *execv*(2). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a **−c** nor a **−t** option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a **−c** or **−t** argument, and in any case for each directory component of *path* which does not begin with a /, the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus **(cd ; pwd) ; pwd** prints the *home*

directory; leaving you where you were (printing this after the home directory), while **cd ; pwd** leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g., **$shell**). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell is – then this is a login shell. The flag arguments are interpreted as follows:

–c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

–e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

–f The shell will start faster, because it will neither search for nor execute commands from the file **.cshrc** in the invoker's home directory.

–i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

–n Commands are parsed, but not executed. This aids in syntactic checking of shell scripts.

–s Command input is taken from the standard input.

–t A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.

–v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

–x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

-V Causes the *verbose* variable to be set even before **.cshrc** is executed.

-X Is to **-x** as **-V** is to **-v**.

After processing of flag arguments if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options was given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by **$0**. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a "standard" shell if the first character of a script is not a #, i.e., if the script does not start with a comment. Remaining arguments initialize the variable *argv*.

Signal handling

The shell normally ignores *quit* signals. Jobs running detached (by the & command) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file **.logout**.

AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp of I.I.A.S.A, Laxenburg, Austria, with different syntax than that used now.

FILES

/etc/cprofile	Read by the login shell before **.cshrc**.
~/.cshrc	Read at beginning of execution by each shell.
~/.login	Read by login shell, after **.cshrc** at login.
~/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a #.
/tmp/sh*	Temporary file for <<.
/etc/passwd	Source of home directories for ~**name**.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves

filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), shl(1), access(2), fork(2), pipe(2), umask(2), wait(2), a.out(5).

NOTES

Csh may not be compatible with some shell commands, such as *at*(1), *newgrp*(1), and *wm*(1).
If the first character in an executable file is #, the file is interpreted as a *csh* script. Because # is interpreted as a comment delimiter by *sh*, it is recommended that *sh* scripts begin with a blank line.

BUGS

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

Commands within loops, prompted for by ?, are not placed in the *history* list. *Csh* should parse the control structure rather recognizing built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; metasyntax.

It should be possible to use the : modifiers on the output of command substitutions. All and more than one : modifier should be allowed on $ substitutions.

NAME

csplit – context split

SYNOPSIS

csplit [**–s**] [**–k**] [**–f** prefix] file arg1 [. . . argn]

DESCRIPTION

Csplit reads *file* and separates it into n+1 sections, defined by the arguments *arg1. . . argn*. By default the sections are placed in xx00 . . . xx*n* (*n* may not be greater than 99). These sections get the following pieces of *file*:

00: From the start of *file* up to (but not including) the line referenced by *arg1*.

01: From the line referenced by *arg1* up to the line referenced by *arg2*.

⋮

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

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

The options to *csplit* are:

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

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

–f *prefix* If the **–f** option is used, the created files are named *prefix***00 . . .** *prefixn*. The default is **xx00 . . . xx***n*.

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

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

%rexp%

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

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

{*num*} Repeat argument. This argument may follow any of the above arguments. If it follows a *rexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

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

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(5).

DIAGNOSTICS

Self explanatory except for:

```
arg - out of range
```

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

NAME

ct – spawn getty to a remote terminal

SYNOPSIS

ct [**–h**] [**–v**] [**–wn**] [**–s**speed] telno ...

DESCRIPTION

Ct dials the phone number of a modem that is attached to a terminal, and spawns a *getty* process to that terminal. *Telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. If more than one telephone number is specified, *ct* will try each in succession until one answers; this is useful for specifying alternate dialing paths.

Ct 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. If there are no free lines, *ct* will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. *Ct* will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the **–w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

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 the standard error output stream.

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

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

Of course, the destination terminal must be attached to a modem that can answer the telephone.

FILES

/usr/lib/uucp/L-devices
/usr/adm/ctlog

SEE ALSO

cu(1C), login(1), uucp(1C).

NAME

ctags – create a tags file

SYNOPSIS

ctags [**–u**] [**–v**] [**–w**] [**–x**] name ...

DESCRIPTION

Ctags creates a tags file, *tags*, from the specified C, Pascal, and FORTRAN sources. The *ex*(1) **tags** command uses a tags file to find specified objects, functions in this case, in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition, with the fields separated by blanks or tabs.

If a file's name ends with **.c** or **.h**, it is searched for C function and macro definitions. The *main* function is treated as a special case, so as to permit multiple programs in one directory: the tag is the name of the file, striped of leading directory names and trailing **.c**, with **M** prepended.

If a file's name does not end with **.c** or **.h**, it is searched for Pascal definitions, then for FORTRAN definitions, then for C definitions.

These are the options:

–w No warning diagnostics.

–u Update the tags file. (It is usually faster just to rebuild the tags file.)

–a Append new definitions to the end of the tags file.

–x Process a list of function definitions, with line numbers and file names.

FILES

tags output tags file

SEE ALSO

ex(1), vi(1).

AUTHOR

Ken Arnold; FORTRAN added by Jim Kleckner; Bill Joy added Pascal and **–x** replacing *cxref;* C typedefs added by Ed Pelegri-Llopart.

WARNING

Recognition of FORTRAN and Pascal objects is done is a very simpleminded way. No attempt is made to deal with block structure.

NAME

ctinstall – install software

SYNOPSIS

/install/ctinstall [update | install] [groups ...]

DESCRIPTION

Ctinstall is used to install operating system software and application software from quarter-inch tape and diskette media. It must be invoked in single-user mode.

If no arguments are provided to *ctinstall*, the user will be prompted for the required information. The option *install* is for raw, or first installs; *update* is for software updates; *groups* is any number of group names specified in the software product's associated proto file.

EXAMPLE

A sample installation session is illustrated here. User responses are shown in **bold** type. A carriage return is implied after all user input.

cd /
halt
Ok To Stop Or Reset Processor
/install/ctinstall

©(#)ctinstall 1.1

Positioning the Tape for Product Installation.

Update or new installation of ISAM 5.00 ('update' or 'install')?: **install**

Running fsck on root file system.
If there are any problems the system will re-boot.
After the system is back up re-execute /install/ctinstall in single user mode.

/dev/dsk/c0d0s1
File System: Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
NNN files NNNN blocks NNNN free

Unmounting /usr.

Running fsck on /usr file system.

/dev/dsk/c0d0s3
File System: Volume:

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
NNN files NNNN blocks NNNN free

Re-mounting /usr.

Please enter your group choices for ISAM separated by blanks.
Your choices are:

ISAM

If you'd like all of the groups, type 'all': **ISAM**

This procedure will install the following ISAM 5.00 group(s) on your system:

ISAM

BE SURE YOU BACK UP ANYTHING YOU HAVE CHANGED
BEFORE PROCEEDING.

Type 'yes' to continue: **yes**

Starting to Install Group(s) ISAM.
Installing Group ISAM.

Calculating size required for group ISAM.
Installation will require an additional NNN root Blocks (512 Byte Blocks).
(Currently NNNN 512 Byte blocks are available on root.)
Installation will require an additional NNN /usr Blocks (512 Byte Blocks).
(Currently NNNN 512 Byte blocks are available on /usr.)

Installing required ISAM files.
install/IsamRel
usr/include/isam.h
usr/include/iserc.h
usr/lib/isam/IsamConfig
usr/lib/isam/IsamCreate
usr/lib/isam/IsamProtect
usr/lib/isam/IsamReorg
usr/lib/isam/IsamStat
usr/lib/isam/IsamStop
usr/lib/isam/IsamTransfer
usr/lib/isam/IxFilter

usr/lib/isam/IxSpec
usr/lib/isam/isam

Checking permissions, modes and omissions on new ISAM commands.
Completed Installation of Group ISAM.
Rewinding tape.

Installation Complete.

SEE ALSO

qlist(1), qinstall(1).
Release Notice for software product being installed.

BUGS

Ctinstall does not understand mountable file systems other than /**usr**.

NAME

ctrace – C program debugger

SYNOPSIS

ctrace [options] [file]

DESCRIPTION

Ctrace allows you to follow the execution of a C program, statement by statement. The effect is similar to executing a shell procedure with the **–x** option. *Ctrace* reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc*(1) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(1) or *tail*(1) commands.

The only *options* you will commonly use are:

–f *functions* Trace only these *functions*.
–v *functions* Trace all but these *functions*.

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these *options*:

–o Octal
–x Hexadecimal
–u Unsigned
–e Floating point

These *options* are used only in special circumstances:

−l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

−s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

−t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The Diagnostics section explains when to use this option.

−P Run the C preprocessor on the input before tracing it. You can also use the **−D**, **−I**, and **−U** *cc*(1) preprocessor options.

These *options* are used to tailor the run-time trace package when the traced program will run in an environment other than CTIX or other UNIX-compatible systems:

−b Use only basic functions in the trace code, that is, those in *ctype*(3C), *printf*(3S), and *string*(3C). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(2), *fflush*(3S), *longjmp*(3C), or *setjmp*(3C).

−p *'s'* Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.

-r f Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **-p** option).

EXAMPLE

If the file *lc.c* contains this C program:

```
 1 #include <stdio.h>
 2 main()        /* count lines in input */
 3 {
 4       int c, nl;
 5
 6       nl = 0;
 7       while ((c = getchar()) != EOF)
 8               if (c = '\n')
 9                       ++nl;
10       printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
(cntl-d),
```

the program will be compiled and executed. The output of the program will be the number **2**, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
 2 main()
 6      nl = 0;
        /* nl == 0 */
 7      while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
        /* c == 49 or '1' */
 8              if (c = '\n')
                /* c == 10 or '\n' */
 9                      ++nl;
                        /* nl == 1 */
 7      while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
 8              if (c = '\n')
                /* c == 10 or '\n' */
 9                      ++nl;
                        /* nl == 2 */
 7      while ((c = getchar()) != EOF)
```

If you now enter an end of file character (cntl-d) the final output will be:

```
        /* c == -1 */
10      printf("%d\n", nl);
        /* nl == 2 */2
         return
```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equal operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the **-f** or **-v** options to trace specific functions. This does not give you statement by statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding *ctroff*() and *ctron*() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from *sdb*(1) if you compile with the **-g** option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
r
```

You can also turn the trace off and on by setting static variable tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly, such as *adb*(1).

DIAGNOSTICS

This section contains diagnostic messages from both *ctrace* and *cc*(1), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

Ctrace Diagnostics

warning: some variables are not traced in this statement

Only 10 variables are traced in a statement to

prevent the C compiler "out of tree space; simplify expression" error. Use the **-t** option to increase this number.

warning: statement too long to trace
This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use -P option
This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

'if ... else if' sequence too long
Split the sequence by removing an **else** from the middle.

possible syntax error, try -P option
Use the **-P** option to preprocess the *ctrace* input, along with any appropriate **-D**, **-I**, and **-U** preprocessor options. If you still get the error message, check the Warnings section below.

Cc Diagnostics

warning: floating point not implemented
warning: illegal combination of pointer and integer
warning: statement not reached
warning: sizeof returns 0
Ignore these messages.

compiler takes size of function
See the *ctrace* "possible syntax error" message above.

yacc stack overflow
See the *ctrace* "'if ... else if' sequence too long" message above.

out of tree space; simplify expression
Use the **-t** option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

redeclaration of signal
Either correct this declaration of *signal*(2), or remove it and #include <signal.h>.

WARNINGS

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

Ctrace assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g. "int EOF;", will cause a syntax error.

BUGS

Ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. Ctrace may choose to print the address of an aggregate or use the wrong format (e.g., %e for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES

runtime.c	run-time trace package

SEE ALSO

signal(2), ctype(3C), fflush(3S), longjmp(3C), printf(3S), setjmp(3C), string(3C).

NAME

cu – call another computer system

SYNOPSIS

cu [**–s**speed] [**–l**line] [**–h**] [**–t**] [**–d**] [**–m**] [**–o**] [**–e**] [**–n**] **telno | systemname | dir**

DESCRIPTION

Cu calls up another computer system or a terminal. It manages an interactive conversation with possible transfers of ASCII files.

cu accepts the following options and arguments.

–sspeed
Specifies the transmission speed(110, 150, 300, 600, 1200, 4800, 9600); 300 is the default value. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

–lline Specifies a device name to use as the communication line. This can be used to override searching for the first available line having the right speed. When the -l option is used without the -s option, the speed of a line is taken from the file **/usr/lib/uucp/L-devices**. When the -l and -s options are used simultaneously, cu will search the L-devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/ttyab**), in this case a phone number is not required but the string dir may be use to specify a null acu. If the specified device is associated with an auto dialer, a phone number must be provided.

–h Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

–t Used when dialing an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

–d Causes diagnostic traces to be printed.

–e Designates that even parity is to be generated for data sent to the remote.

-o Designates that odd parity is to be generated for data sent to the remote.

-m Designates a direct line which has modem control.

-n Will request the phone number to be dialed from the user rather than taking it from the command line.

telno When using an automatic dialer the argument is the telephone number with equal signs for secondary dial tone or minus signs for delays, at appropriate places.

systemname
A **uucp** system name may be used rather than a phone number; in this case, cu will obtain an appropriate direct line or phone number from **/usr/lib/uucp/L.sys** (the appropriate baud rate is also read along with phone numbers). Cu will try each phone number or direct line for **systemname** in the L.sys file until a connection is made or all the entries are tried.

dir Using **dir** insures that cu will use the line specified by the -l option.

After making the connection, *cu* runs as two processes: the *transmit* process reads data from the standard input and, except for lines beginning with ~, passes it to the remote system; the *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote 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* on the local system (via **sh -c**).
~$*cmd* . . .	run *cmd* locally and send its output to the remote system.
~%cd	change the directory on the local system. NOTE: **~!cd will cause the command to be run by a sub-shell; probably not what was intended.**

~%**take** *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~%**put** *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places.
~~...	send the line ~... to the remote system.
~%**break**	transmit a **BREAK** to the remote system.
~%**nostop**	toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

The *receive* process normally copies data from the remote system to its standard output. A line from the remote that begins with ~> initiates an output diversion to a file. The complete sequence is:

~>[>]: *file*
zero or more lines to be written to *file*
~>

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

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

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

When **cu** is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. For example, uname can be executed on Z, X, and Y as follows:

uname
Z
~!uname

X
~~!uname
Y
In general, ~ causes the command to be executed on the original machine, ~~ causes the command to be executed on the next machine in the chain.

EXAMPLES

To dial a system whose number is 9 201 555 1212 using 1200 baud:

```
cu -s1200 9=2015551212
```

If the speed is not specified, 300 is the default value.

To login to a system connected by a direct line:

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

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

```
cu -s1200 -l /dev/ttyXX dir
```

To dial a system using a specific line:

```
cu -l /dev/culXX 2015551212
```

To use a system name:

```
cu YYYZZZ
```

FILES

/usr/lib/uucp/L.sys
/usr/lib/uucp/L-devices
/usr/spool/uucp/LCK..(tty-device)
/dev/null
/usr/lib/uucp/modemcap
/usr/lib/uucp/L-dialcodes

SEE ALSO

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

DIAGNOSTICS

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

BUGS

Cu buffers input internally.
There is an artificial slowing of transmission by *cu* during the **~%put** operation so that loss of data is unlikely.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

The meanings of the options are:

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

–c*list* The *list* following **–c** (no space) specifies character positions (e.g., **–c1–72** would pass the first 72 characters of each line).

–f*list* The *list* following **–f** is a list of fields assumed to be separated in the file by a delimiter character (see **–d**); e.g., **–f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless **–s** is specified.

–d*char* The character following **–d** is the field delimiter (**–f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.

–s Suppresses lines with no delimiter characters in case of **–f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the **–c** or **–f** option must be specified.

HINTS

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

EXAMPLES

cut -d: -f1,5 /etc/passwd	mapping of user IDs to names
name=`who am i | cut -f1 -d" "`	to set **name** to current login name.

DIAGNOSTICS

line too long — A line can have no more than 1023 characters or fields.

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

no fields — The *list* is empty.

SEE ALSO

grep(1), paste(1).

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
!$&()'`*+@.,/:;=?[]|-_^~"<>{}#

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

Character	*Symbol*	*Troff Name*
"Cents" sign	¢	\(ct
EBCDIC "not" sign	¬	\(no
Left arrow	←	\(<-
Right arrow	→	\(->
Down arrow	↓	\(da
Vertical single quote	'	\(fm
Control-shift indicator	†	\(dg
Visible space indicator	□	\(sq
Hyphen	-	\(hy

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

.PC ***arg1 arg2 arg3 ... argn***
Same as .CP, except that the even-numbered arguments are set in the CW font and the odd-numbered ones in the prevailing 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 (\) 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 particular, lines that begin with . and ' are passed through untouched (except if they contain delimiters–see below). In either case, *cw* hides the effect of the font changes generated by the .CW and .CN requests; *cw* also defeats all ligatures (fi, ff, etc.) in the CW font.

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 example under *BUGS* below). *Cw* treats text between delimiters in the same manner as text enclosed by .CW /.CN pairs, except that, for aesthetic reasons, spaces and backspaces inside .CW /.CN pairs have the same width as other CW characters, while spaces and backspaces between delimiters are half as wide, so they have the same width as spaces in the prevailing text (but are *not* adjustable). Font changes due to delimiters are *not* hidden.

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

The options are:

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

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

-f*n* 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 (or when - is specified as the last argument), 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.

HINTS

Typical definitions of the .CW and .CN macros meant to be used with the *mm*(5) macro package:

```
.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 .5i 1i 1.5i 2i 2.5i 3i ...
.vs
.ps
.DE
..
```

At the very least, the .CW macro should invoke the *troff*(1) 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 ***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: ***cw***, ***tbl***, and ***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 *tbl*(1) format information not be modified by *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 <- represent a backspace, d<-<-\(dg yields ‡ (Because backspaces are half as wide between delimiters as inside .CW /.CN pairs–see above–two backspaces are required for each overstrike between delimiters.)

FILES

/usr/lib/font/ftCW CW font-width table

SEE ALSO

eqn(1), mmt(1), tbl(1), troff(1), mm(5), mv(5).

WARNINGS

If text preprocessed by *cw* is to make any sense, it must

be set on a typesetter equipped with the CW font or on a STARE facility; on the latter, the CW font appears as bold, but with the proper CW spacing.

BUGS

Only a masochist would use periods (.), backslashes (), or double quotes (") as delimiters, or as arguments to .CP and .PC.
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 *troff*(1) half- and quarter-spaces (\| and \^) is most salutary, e.g., one should use _&_\^, (rather than just plain _&_,) to obtain &, (assuming that _ is used for both delimiters).
Using *cw* with *nroff* is silly.
The output of *cw* is hard to read.
See also *BUGS* under *troff*(1).

NAME

cxref – generate C program cross-reference

SYNOPSIS

cxref [options] files

DESCRIPTION

Cxref analyzes a collection of C files and attempts to build a cross-reference table. *Cxref* utilizes a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or with the **–c** option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **–D**, **–I** and **–U** options (which are identical to their interpretation by *cc*(1)), the following *options* are interpreted by *cxref*:

–c Print a combined cross-reference of all input files.

–w<*num*>
Width option which formats output no wider than <num> (decimal) columns. This option will default to 80 if <num> is not specified or is less than 51.

–o file Direct output to named *file*.

–s Operate silently; does not print input file names.

–t Format listing for 80-column width.

FILES

/usr/lib/xcpp special version of C-preprocessor.

SEE ALSO

cc(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files, anyway.

BUGS

Cxref considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that **#includes ctype.h** will contain many declarations of the variable **c**.

NAME

date – print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

MightyFrame Only:
date [–]

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 MightyFrame system has a real-time clock that sets the current system date. The **date** – command sets the system time to that of the real-time clock. If arguments are given, *date* changes the time on the real-time clock.

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 conversion 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:

n	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	day of year – 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 have generated as output:

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

DIAGNOSTICS

No permission	if you are not 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.

SEE ALSO

printf(3S).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.) The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number

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

+ – / * % ^

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

s*x*

The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x*

The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d

The top value on the stack is duplicated.

p

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

f

All values on the stack are printed.

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

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

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

[... **]** puts the bracketed ASCII string onto the top of the stack.

<*x* **>***x* **=***x*
The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! interprets the rest of the line as a CTIX system command.

c All values on the stack are popped.

i The top value on the stack is popped and used as the number radix for further input. I pushes the input base on the top of the stack.

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

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

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

z The stack level is pushed onto the stack.

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

? A line of input is taken from the input source (usually the terminal) and executed.

; : are used by *bc* for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented
: 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

dcopy – copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [**–s**X] [**–an**] [**–d**] [**–v**] [**–f**fsize[:isize]] inputfs outputfs

DESCRIPTION

Dcopy copies file system *inputfs* to *outputfs*. *Inputfs* is the existing file system; *outputfs* is an appropriately sized file system, to hold the reorganized result. For best results *inputfs* should be the raw device and *outputfs* should be the block device. *Dcopy* should be run on unmounted file systems (in the case of the root file system, copy to a new slice). With no arguments, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are

–s*X* supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the **–s** option of *fsck*(1M).

–a*n* place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified then no movement occurs.

–d leave order of directory entries as is (default is to move sub-directories to the beginning of directories).

–v currently reports how many files were processed, and how big the source and destination freelists are.

–f*fsize*[:*isize*]
specify the *outputfs* file system and inode list sizes (in blocks). If the option (or :*isize*) is not given, the values from the *inputfs* are used.

Dcopy catches interrupts and quits and then reports on its progress. To terminate *dcopy* send a quit signal, and *dcopy* will no longer catch interrupts or quits.

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
..., ...	several comma-separated conversions

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 block 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 block 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 block sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

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

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256-character standard in the CACM Nov, 1968. The *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 [–**r**SID] [–**s**] [–**n**] [–**g**list] [–**m**[mrlist]] [–**y**[comment]] [–**p**] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see **–m** and **–y** keyletters below).

Keyletter arguments apply independently to each named file.

–r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *get*s for editing (**get –e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **–r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

–s Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

-g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

-m[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (MR) number *must* be supplied as the reason for creating the new delta.

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

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

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from MR number validation program, *delta* terminates (it is assumed that the MR numbers were not all valid).

-y[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the

standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form ?-file are explained in the "Source Code Control System User's Guide" in Section 9 of the *CTIX Programmer'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 SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile*(4). (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.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1),

rmdel(1), sccsfile(4).
CTIX Programmer's Guide, Section 9.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

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

SYNOPSIS

deroff [**-mx**] [**-w**] [files]

DESCRIPTION

Deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and .EN lines, and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *Deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **-m** option may be followed by an **m**, **s**, or **l**. The **-mm** option causes the macros 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), nroff(1), tbl(1), troff(1), spell(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. (As a special case, both the block device name and the swap device name are printed for the argument name / if swapping is done on the same disk section as the **root** file system.) Argument names must be full path names.

This command is most commonly used by **/etc/rc** (see *brc*(1M)) to construct a mount table entry for the **root** device.

EXAMPLE

The command:

```
/etc/devnm /usr
```

produces

```
dsk/c0d0s3 /usr
```

if **/usr** is mounted on **/dev/dsk/c0d0s3**.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M), setmnt(1M).

NAME

Devices – configuration file for uucp communications lines

SYNOPSIS

/usr/lib/uucp/Devices

DESCRIPTION

/usr/lib/uucp/Devices is a text file that contains configuration specifications for communications devices, such as modems or direct lines. Each line in the file describes a single device and how it communicates with a remote system. Comment lines begin with a pound sign (#). The UUCP system uses the **/usr/lib/uucp/Devices** file in conjunction with the **/usr/lib/uucp/Dialers** file to place a call.

Each line containes five or more fields delimited by spaces. The first field is the line type as specified in the **/usr/lib/uucp/Systems** file; for direct lines, the first field is the name of the remote system.

The remaining fields give the device name; the calling device indicator (such as for 801 calling units), if used; the speed, which may be specified as ANY; and the name of the caller as specified in the **/usr/lib/uucp/Dialers** file. The last field, the name of the caller, may be followed by a token format (containing **\D** or **\T**); pairs of these dialer name/token format fields can be repeated if more than one dialer must be used in succession to make the connection. If no token format is specified, a **\D** is used for a dialer name that references the **/usr/lib/uucp/Dialers** file; a **\T** is used for internal dialer types such as 801. Unused fields are replaced by a hyphen (–).

EXAMPLE

The following entry configures a 1200-baud intelligent modem on device contty for use with UUCP:

ACU contty - 1200 penril

FILES

/usr/lib/uucp/Devices
/usr/lib/uucp/Dialers
/usr/lib/uucp/Systems

SEE ALSO

uucp(1C), dial(3C), Dialers(5).
MightyFrame Administrator's Reference Manual.

NAME

Dialers – ACU/modem calling protocols

SYNOPSIS

/usr/lib/uucp/Dialers

DESCRIPTION

Dialers describes the call-placing protocols for intelligent modems, ACUs (automatic calling units), and other serial switched devices such as data switches. When a connection is requested via the UUCP system, CTIX looks for a description of the called system in the **/usr/lib/uucp/Systems** file, where the type of line is specified for connection to that system. CTIX then checks the **/usr/lib/uucp/Devices** file for a description of the line, its speed and its Dialers name. The Dialers name given in the **Devices** file corresponds to the first field of the **Dialers** file.

Dialers is a text file that contains the dialing script for the modems that are configured in the **Devices** file. Each description begins on a new line and has three or more fields, delimited by spaces.

The first field of the description is the name of the modem or device as specified in the **Devices** file.

The second field specifies the codes used by that particular modem for secondary dial tone (=) and pause (–); this field enables CTIX to translate from the standard 801 codes (= and –) to the special characters used by that particular device.

The remaining fields are the chat script that is necessary to establish communication with the modem.

The modem chat script is composed of command strings to the modem and response strings expected in return from the modem. The strings consist of ASCII and control characters that are recognized by the individual modem or device. Spaces delimit the end of a send or receive sequence. The first string is an expect string.

Several modems and switches are already provided in the **Dialers** file. Additional devices can be configured by studying the manufacturers' manuals to determine the appropriate send/receive sequences for other modems.

In the string sequences of the send/receive fields the following escape sequences represent control codes:

\ddd Octal number.

\c Suppress new line (valid only after **\r** or at the end of a field).

\d Delay (two seconds).

\D Substitute the telephone number (from the **/usr/lib/uucp/Systems** file or *cu*(1C)), without character translation.

\e Turn off echo checking.

\E Turn on echo checking (for slow devices).

\K Insert a BREAK.

\n New-line.

\p Pause (a slight delay of one-quarter to one-half second).

\r Carriage return.

\T Substitute the telephone number (from the **/usr/lib/uucp/Systems** file or *cu*(1C)), with character translation. Character translation interprets the 801 codes in the second field and expands any symbols found in the **/usr/lib/uucp/Dialcodes** file.

Comments delimited by a pound sign (#), spaces, or tabs are ignored. Any line terminated by a backslash (\) continues to the next line.

EXAMPLE

The following example establishes communication with a Ventel modem:

```
ventel =&-%      "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
```

The first field, "ventel," is the name of the modem that corresponds to a "ventel" caller type in the fifth or subsequent field of a **Devices** file entry. The second field describes the modem's convention for the secondary dial tone (&) and a pause (%) command. The remaining fields consist of five strings separated by spaces. The five strings are interpreted as follows:

1. The first expect string ("") is null.
2. Send to the modem a series of carriage returns to elicit a prompt.
3. The modem should respond with a dollar sign ($).
4. Send the telephone number (**\T**) to the modem.
5. Upon connection the modem should respond with the string 'ONLINE!'.

FILES

/usr/lib/uucp/Devices
/usr/lib/uucp/Dialcodes
/usr/lib/uucp/Systems

SEE ALSO

uucp(1C), dial(3C), Devices(5).
MightyFrame Administrator's Reference Manual.

NAME

df – report number of free disk blocks

SYNOPSIS

df [**–t**] [**–f**] [file-systems]

DESCRIPTION

Df prints out the number of free 512-byte blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/dsk/c0d0s1**) 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/dsk/*
/etc/mnttab

SEE ALSO

fs(4), mnttab(4).

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:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where $n1 = n2$ or $n3 = n4$, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

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 (.).

WARNINGS

Missing newline at end of file X

indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**–ex3**] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

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

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

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

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

Under the **–e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3. Option **–x** (**–3**) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to *file1*.

```
ed – file1 < script
```

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

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

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* or *troff*(1). *Name1* and *name2* are the old and new versions of the file. *Diffmk* generates *name3*, which contains the lines of *name2* plus inserted formatter "change mark" (**.mc**) requests. When *name3* is formatted, changed or inserted text is shown by | at the right margin of each line. The position of deleted text is shown by a single *.

If anyone is so inclined, *diffmk* can be used to produce listings of C (or other) programs with changes marked. A typical command line for such use is:

```
diffmk old.c new.c tmp; nroff macs tmp | pr
```

where the file **macs** contains:

```
.pl 1
.ll 77
.nf
.eo
.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), troff(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 [**–d**] [**–s**] [**–w***n*] dir1 dir2

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same contents.

–d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

–s Suppress messages about identical files.

–w *n* Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1).

```
 *          dump area
 *          down load image file
 *          Bootable program,
 *          size determined by a.out format. nblocks=1.
 */
    char   fpulled;          /* dismounted last time? */
    long   time;             /* time last came on line */
    struct gdswprt2 dsk2;    /* Drive specific parameters */
    char   minires[38];      /* for future mini/miti frame
                             enhancements */
    char   sysres[292];      /* custom system area */
    struct mntnam mntname[MAXSLICE];
                             /* names for auto mounting; null
                              * string means no auto mount
                              * not used in mitiframe */
    char   userres[256];     /* user area */
};

struct gdswprt {
    char   name[6];      /* printf name */
    ushort cyls;         /* the number of cylinders for this disk */
    ushort heads;        /* number of heads per cylinder */
    ushort psectrk;      /* number of physical sectors per track */
    ushort pseccyl;      /* number of physical sectors per cylinder */
    char   flags;        /* floppy density and high tech drive flags */
    char   step;         /* stepper motor rate to controller -
                            ST506 only */
    ushort sectorsz;     /* size of physical sectors (in bytes) */
};

struct gdswprt2 {
    short  wpccyl;           /* value to program for RWC/WPC -
                             ST506 only */
    ushort enetaddr[3];      /* Ethernet station address -
                              * MiniFrame only */
    unchar       gap1;       /* Gap size on SMD drives */
    unchar       gap2;
    char   filler[28];
};

struct partit{
    union {
        uint strk;          /* start track number (new style) */
        struct {
            ushort strk;  /* start track # */
            ushort nsecs; /* # logical blocks available to user */
        } old;
    } sz;
};
```

If a volume home block is valid, *magic* is equal to VHBMAGIC and the 32-bit sum of the volume home block's bytes is 0xFFFFFFFF (−1); *chksum* is the adjustment that makes the sum come out right.

Dsk describes the peculiarities of the disk, including deliberate deviations from the system standard. *Dsk.flags* the bitwise or of zero or more of the following constants:

FPDENSITY	(MiniFrame only) If on, the disk is double density; if off, the disk is single density.
FPMIXDENS	(MiniFrame only) If off, FPDENSITY specifies the density of the first track; if on, the first track is single density regardless of FPDENSITY.
HITECH	(ST506 only) If on, head select bit 3 is valid; if off, reduced write current is valid.
NEWPARTTAB	If off, the old style slice (partition) table is in use; if on, the new style slice table is in use.
RWCPWC	(ST506 only) If on, set reduced write current/write precompensation. HITECH selects write precompensation.
EXCHANGEABLE	If on, the disk is a floppy or removable hard disk cartridge. If off, the disk is a winchester.
FORMATEXTRA	If on, the SMD drive is formatted with an extra sector on each track. (This sector is ignored by CTIX but is required for some disk drives, notably the Eagle-XP.)

Dsk.step specifies a stepper motor rate for the ST506; use 14 in this field.

Partab divides the disk into slices (partitions).

Fpulled indicates whether an exchangeable disk was properly removed from the drive. The system sets this field to 1 when the disk is inserted in the drive. To clear *fpulled*, run *dismount*(1M); see that entry.

Mntname, *minires*, and *userres* are reserved for future use.

Resmap describes the files that share Slice 0 with the Volume Home Block. Provision is made for eight such files, but only five have been assigned slots in *resmap*. Each *resmap* entry gives the starting location (logical block number) and length (logical blocks). A length of zero indicates that the file is not provided. The first five entries in *resmap* describe:

1. The loader. When the system is reset or turned on, the boot prom loads the loader into the loader address and jumps execution to it. The function of the loader is to search for and load a program that will boot the system.

   On MightyFrame the loader searches the tape, onboard Winchester disks 0, 1, and 2, and the VME, in that order. On MiniFrame the loader searches the tape, the floppy disk, and Winchester disks 1 and 0, in that order.

   On each disk, the loader first checks for a standalone program. If the disk lacks a standalone program, the loader checks for a CTIX kernel, which must be a CTIX executable object file called **/unix** in the file system in slice 1. When the loader locates an appropriate program, it preserves the crash dump table, loads the program it found at the address it was linked at (0x0 if unknown) and executes it. If no disk contains an appropriate file, the loader continues searching until an appropriate disk is inserted.

2. The bad block table, which always begins at logical block 1 of the disk. Each logical block in the bad block table consists of a four-byte checksum followed by 127 bad block cells. The checksum is a value that makes the 32-bit sum of the logical block be 0xFFFFFFFF (−1). A bad block cell is defined by the following structure.

```
struct bbcell {
    ushort cyl;         /* the cylinder of the bad block */
    ushort badblk;      /* the physical sector address of
                           the bad block within the cylinder cyl */
    ushort altblk;      /* track number of alternate */
    ushort nxtind;      /* index into the cell array for next
                           bad block cell for this cylinder */
};
```

A single sequence of numbers, starting from zero, identifies the checksums and cells. In each cell in use, ***cyl*** identifies a cylinder that contains the bad block; ***badblk*** physical block offset within the cylinder of the bad block; ***altblk*** identifies the track that contains the alternate block; ***nextind*** (not used in MightyFrame) identifies the next cell for a bad block on the same cylinder or is zero if this is the last one.

3. The dump area. After Reset or Suicide, the Boot prom dumps processor registers, the memory map, a crash dump block, and the contents of physical memory, until it runs out of room in the dump area.

4. The down load image area. The down load images are described by a table at the beginning of the area. The area is described by the following array.

```
struct dldent {
        short d_strt;
        /* block displacement from down load index */
        short d_sz;
        /* # of blocks for this entry */
};
```

The image number is the index for *dldent*. *D_strt* is the offset in bytes of the image from the beginning of the down load image area; *d_sz* is the size in bytes of the image.

5. A bootable program, usually a diagnostic. This is the program the loader considers a substitute for the **/unix** file. The program must be in *a.out*(4) format with magic number 407 or be a simple memory image.

   If the fifth entry in *resmap* has a zero address but a nonzero length, the loader looks at the beginning of slice 1 for the program.

Slice 0 is called the Reserved Area. Only the volume home block and the files described by *resmap* can be in the Reserved Area. A formatted disk used by a working system certainly has at least one more slice.

Ioctl system calls use the following structure.

```
struct gdioctl {
    ushort status;              /* status */
    struct gdswprt params;      /* description of the disk */
    struct gdswprt2 params2;    /* more description of the disk */
    short  ctrltyp;             /* the type of disk controller */
    short  driveno;
};
```

Status is the bitwise or of the following constants.

VALID_VHB A valid Volume Header Block has been read.

DRV_READY The disk is on line.

PULLED Last removal of disk from drive was not preceded by proper dismount.

Params is a *gdswprt* structure, the same type used in the volume header block.

Dsktype is equal to

GD_WD1010 for Western Digital 1010 ST506 Controller

GD_WD2010 for Western Digital 2010 ST506 Controller

GD_WD2797 for Western Digital 2797 Floppy Disk Controller

GD_RAMDISK for RAM Disk Emulator

GD_SMD3200 for Interphase SMD3200 disk controller

CTIX understands the following disk *ioctl* calls.

ioctl(fd, GDIOCTYPE, 0)
Returns GDIOC if *fd* is a file descriptor for a disk special file.

ioctl(fd, GDGETA, gdctl_ptr)
Gdctl_ptr is a pointer to a *gdioctl* structure. *Ioctl* fills the structure with information about the disk.

ioctl(fd, GDSETA, gdctl_ptr)
Gdctl_ptr is a pointer to a *gdioctl* structure. *Ioctl* passes the description of the disk to the disk driver. This is primarily meant for reading disks created by other kinds of computers.

ioctl(fd, GDFORMAT, ptr)
Ptr points to formating information. The disk driver formats a track.

ioctl(fd, GDDISMNT)
Ioctl informs the driver that the user intends to remove the disk from the drive. When this system call successfully returns, the driver has flushed all data in the buffer cache and waited for all queued transfers to complete. The last transfer is to write out the Volume Home Block with the *fpulled* flag cleared. Once this call returns the drive is inaccessible until a new disk is inserted.

SEE ALSO
iv(1), mknod(1M), ioctl(2).

NAME

diskusg - generate disk accounting data by user ID

SYNOPSIS

/usr/lib/acct/diskusg [options] [files]

DESCRIPTION

Diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. *Diskusg* outputs lines on the standard output, one per user, in the following format:

uid login #blocks

where

uid — is the numerical user ID of the user.

login — is the login name of the user; and

#blocks — is the total number of 512-byte disk blocks allocated to this user.

Diskusg normally reads only the i-nodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

Diskusg recognizes the following options:

-s — the input data is already in *diskusg* output format. *Diskusg* combines all lines for a single user into a single line.

-v — verbose. Print a list on standard error of all files that are charged to no one.

-i *fnmlist* — ignore the data on those file systems whose file system name is in *fnmlist*. *Fnmlist* is a list of file system names separated by commas or enclose within quotes. *Diskusg* compares each name in this list with the file system name stored in the volume ID (see *labelit(1M)*).

-p *file* — use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.

-u *file* — write records to *file* of files that are charged to no one. Records consist of the special file name, the i-node number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk* (see *acct*(1M)) which generates total accounting records that can be merged with other accounting records. *Diskusg* is normally run in *dodisk* (see *acctsh*(1M)).

EXAMPLES

The following will generate daily disk accounting information:

```
for i in s1 s3; do
    diskusg /dev/rdsk/c0d0$i > dtmp.‘basename $i‘ &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

/etc/passwd	used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M), acct(4)
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

dismount – remove exchangeable disk

SYNOPSIS

dismount [**–f**]

DESCRIPTION

Dismount must be executed before physical removal of an exchangeable disk from its drive. For each disk that is labeled exchangeable (see *disk*(7)), *dismount* syncronizes and unmounts its mounted file systems, writes out its volume home block, bars further I/O, and clears its "pulled" flag. The last action prevents a warning message the next time the disk is placed in the drive.

Once *dismount* has been run, the exchangeable disk's drive is unusable until the dismounted disk is removed.

If a MiniFrame has two kinds of exchangeable disks, *dismount* dismounts them both. To restrict *dismount* to one disk, specify **–f** for floppy.

FILES

/etc/mnttab	mounted file system list
/dev/dsk/*	disk reserved area

SEE ALSO

fsck(1M), update(1), disk(7).

NAME

du – summarize disk usage

SYNOPSIS

du [**–ars**] [names]

DESCRIPTION

Du gives the number of 512-byte blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, . is used.

The optional argument **–s** causes only the grand total (for each of the specified *names*) to be given. The optional argument **–a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

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

drivers – loadable device drivers

DESCRIPTION

A loadable driver is equivalent to a fixed, linked-in device driver. It has access to all kernel subroutines and global data. After it is loaded, it is effectively part of the running kernel.

Differences between loadable and ordinary drivers involve their driver ID, init routine, release routine, and interrupt processing.

Init Routine

Loadable drivers may have an init routine that is executed when the driver is bound, and a release routine that is executed when the driver is unbound (see *lddrv*(1M) for a description of driver allocation and bind operations). Init routines check for the existence of hardware, initialize the hardware, put the interrupt service routine for the hardware into the interrupt chain, and do other similar tasks.

Release Routine

Release routines make sure the device or driver is idle, turn off the device, take the interrupt service routine out of the interrupt chain, and similar tasks. A typical action for a release routine to take when the device *is not* idle is to set an error code in **u.u_error** and return.

Driver ID

All drivers have a driver ID. Preloaded drivers have a driver ID of 0. Loaded drivers are given an ID when they allocate virtual space. The driver ID is automatically set when the driver is linked. The ID should never be modified by the driver itself; the ID is used to identify the driver to the system when making certain requests.

EXAMPLE

```
/* init, release, interrupt service routines */
/* for loadable device xyzzy */

#include <sys/drv.h>

#define XYZ_VECNO       0x60        /* interrupt vector number */
#define XYZ_BUSY        1           /* flags */
#define XYZ_OPEN        2
int xyzzint();                      /* interrupt service routine */

extern int DFLT_ID;
static int Drv_id = &DFLT_ID;       /* set drive ID */
int xy_base;
int xy_flags;
```

```
xy_init()
{
        if (set_vec(Drv_id, XYZ_VECNO, xyzzyint) < 0)
        {
                u.u_error = EBUSY;
                return;
        }
        .
        .
        <do hardware initialization>
        .
}
xy_release()
{
        if (xy_flags & (XY_BUSY | XY_OPEN))
        {
                u.u_error = EBUSY;
                return;
        }
        .
        .
        <turn off device>
        .
        reset_vec (Drv_id, XYZ_VECNO);
}
xyzzyint()
{
        .
        <clear interrupt>
        .
        <process interrupt>
        .
}
```

SEE ALSO

Writing MightyFrame Device Drivers.

NAME

dump – dump selected parts of an object file

SYNOPSIS

dump [**–acfghlorst**] [**–z** name] files

DESCRIPTION

The *dump* command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

–a	Dump the archive header of each member of each archive file argument.
–g	Dump the global symbols in the symbol table of an archive.
–f	Dump each file header.
–o	Dump each optional header.
–h	Dump section headers.
–s	Dump section contents.
–r	Dump relocation information.
–l	Dump line number information.
–t	Dump symbol table entries.
–z name	Dump line number entries for the named function.
–c	Dump the string table.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

–d number	Dump the section number or range of sections starting at *number* and ending either at the last section number or *number* specified by **+d**.
+d number	Dump sections in the range either beginning with first section or beginning with section specified by **–d**.
–n name	Dump information pertaining only to the named entity. This *modifier* applies to **–h**, **–s**, **–r**, **–l**, and **–t**.
–p	Suppress printing of the headers.
–t index	Dump only the indexed symbol table entry. The **–t** used in conjunction with **+t**, specifies a range of symbol table entries.

+t index — Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **-t** option.

-u — Underline the name of the file for emphasis.

-v — Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be used with all the above options except **-s** and **-o** options of *dump*.

-z name,number — Dump line number entry or range of line numbers starting at *number* for the named function.

+z number — Dump line numbers starting at either function *name* or *number* specified by **-z**, up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **-z** option may be replaced by a blank.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

SEE ALSO

a.out(4), ar(4).

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
\v	vertical tab
****	backslash
*****n*	the 8-bit character whose ASCII code is *n*, an octal number of no more than four digits, the first of which must be zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

NAME

ed, red – text editor

SYNOPSIS

ed [–] [file] ...

red [–] [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. If multiple *file* arguments are given, the % argument of the *e* command becomes useful. 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*. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty –tabs** or **stty tab3** mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10 and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this

mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports 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**–**9**] is equivalent to [**0123456789**]. The – loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [ ]**a**–**f**] matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly* *m* occurrences; \{*m*,\} matches *at least* *m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between* *m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \(and \) is a RE that matches whatever the unadorned RE matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression ^\(.*\)\1$ matches a line consisting of two repeated appearances of the same string.

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 RE*$ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x* addresses the line marked with the mark name character *x*, which must be 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 **l**, **n** or **p**, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)**a**
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.)**c**
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)**d**

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). If % is given in place of a file name, the next name on the command line argument list is used. The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *f* ile-name command changes

the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,$)g/*RE/command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted; the . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,$)G/*RE/*

In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an & causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address *'x* then addresses this line; . is unchanged.

(.,.)l

The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab*, *backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)m*a*

The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.)n

The *n*umber command prints the addressed

lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.)p

The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(\$)r *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "\$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE***/***replacement***/** or
(.,.)s/*RE***/***replacement***/g**

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced

by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters *n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(.,.)t*a*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,$)v/*RE*/*command list*

This command is the same as the global command *g* except that the *command list* is

executed with . initially set to every line that does *not* match the RE.

(1,$)V/*RE*/
This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the RE.

(1,$)w *file*
The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

($)=
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 CTIX System 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	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

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 *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands: it prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The – command-line option inhibits this feature.

SEE ALSO

grep(1), sed(1), sh(1), stty(1), fspec(4), regexp(5).
Other editors: vi(1), ex(1).

WARNINGS 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 \n in a RE does not match a new-line character.
The *l* command mishandles DEL.
Characters are masked to 7 bits on input.

Due to export restrictions, encryption features are not available.

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [**–r**] name ...

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor. The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

BRIEF INTRODUCTION

To edit the contents of an existing file you begin with the command "edit name" to the shell. *Edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

Edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a ".", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, line 2 will

become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say "s/*old*/*new*/" where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No **write** since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands

such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "'". The **z** command can also be given other following characters "z-" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form **/text/** to search forward for *text* or **?text?** to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form **/^text/** which searches for *text* at the beginning of a line. Similarly **/text$/** searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$-5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. *Edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing

"put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ex(1), vi(1).

NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable printers
disable [**-c**] [**-r**[reason]] printers

DESCRIPTION

Enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

Disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

-c Cancel any requests that are currently printing on any of the designated printers.

-r[*reason*] Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a reason, then a default reason will be used. *Reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

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(4), environ(5).

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 phototypesetter, while *neqn* is used for the same purpose with *nroff* on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent.

If no files are specified (or if – is specified as the last argument), these programs read 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 **–d***xy* or (more commonly) with **delim** *xy* between .EQ and .EN. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by **delim off**. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, 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 $e^{x^2+y^2}$ 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: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$

The keywords **from** and **to** introduce lower and upper limits: $\lim_{n \to \infty} \sum_{0}^{n} x_i$ is made with *lim from {n -> inf} sum from 0 to n x sub i*. Left and right brackets, braces, etc., of the right height are made with **left** and **right**:
left [x sup 2 + y sup 2 over alpha right] ~=~ 1
produces

$$\left[x^2 + \frac{y^2}{\alpha} \right] = 1.$$

Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and **" "** for nothing at all (useful for a right-side-only bracket). A **left** *thing* need not have a matching **right** *thing*.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**:
pile {a above b above c}
produces
a
b.
c
Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies. Matrices are made with **matrix**:

matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }

produces

x_i 1

y_2 2

In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~=~ n under* is $\ddot{\bar{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \vec{\vec{y}}$.

Point sizes and fonts can be changed with **size** *n* or **size** $\pm n$, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **-s***n* and **-f***n*.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **-p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define thing % replacement %

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** ($\sum$), **int** ($\int$), **inf** (∞), and shorthands such as >= ($\geq$), != ($\neq$), and -> ($\rightarrow$) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or GAMMA (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. *Troff*(1) four-character escapes such as **\(dd** (‡) and **\(bs** (⊘) 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 *troff*(1) when all else fails. Full details are given in the manual cited below.

SEE ALSO

cw(1), mm(1), mmt(1), nroff(1), tbl(1), troff(1), eqnchar(5), mm(5), mv(5).

BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold "12.3"**.
See also *BUGS* under *troff*(1).

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 and removed; the record is truncated if the read request is for less than the record's length.

An appropriate command to the console sends console information to the error record queue. See *console*(7).

FILES

/dev/error special file

SEE ALSO

errdemon(1M), console(7).

NAME

errdead – extract error records and status information from dump

SYNOPSIS

/etc/errdead [-a[e][f]] [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 demon *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.

Errdead understands the following options:

–a Instead of passing extracted records to *errpt*(1M), append them to **/usr/adm/errfile**, provided that the dump corresponds to the namelist and that the dump is newer than the error file.

–e Only valid if **–a** is also specified. Invoke *errdemon*(1M) when done. This is normally done in the *rc* script (see *brc*(1M)).

–f Only valid if **–a** is also specified. Write extracted records even if the dump is older than the error file.

The *dumpfile* specifies the file (or memory) that is to be examined; if not given, *errdead* looks for a dump area by scanning the available disks in the same order as does the bootstrap ROM. 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
/usr/adm/errfile	repository for error records
/etc/log/confile	console 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 demon

SYNOPSIS

/usr/lib/errdemon [-n] [-c file] [file]

DESCRIPTION

The error logging demon *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 demon 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). *Errdemon* can also extract console records; the **–n** option disables this, thus forcing all console reports to stay in a circular buffer in the kernel. The **–c** option allows specifying a console file. The default console file is **/etc/log/confile**. The error-logging demon is terminated by sending it a software kill signal (see *kill*(1)). Only the superuser may start the demon, and only one demon may be active at any time.

FILES

/dev/error	source of error records
/usr/adm/errfile	repository for error records
/etc/log/confile	console records
/dev/console	

DIAGNOSTICS

The diagnostics produced by *errdemon* are intended to be self-explanatory.

SEE ALSO

errpt(1M), errstop(1M), kill(1), err(7).

NAME

errpt – process a report of logged errors

SYNOPSIS

errpt [options] [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 unabled 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.

–d *devlist* — A detailed report is limited to data about devices given in *devlist*, where *devlist* can be one of two forms: a list of device identifiers separated from one another by a comma, or a list of device identifiers enclosed in double quotes and separated from one another

by a comma and/or more spaces. *Errpt* is familiar with the block devices **GD?** (0 to 15). Additional identifiers are **int**, **mem**, **QIC0** for 1/4-inch tape, and **TP0** for 1/2-inch tape, which include detailed reports of stray-interrupt, and **tty** and serial asynchronous terminals memory-parity type errors, respectively.

-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.

Logical blocks in the filesystem are 1024 bytes. Physical sector numbers are 512-byte blocks.

FILES

/usr/adm/errfile default error file

SEE ALSO

date(1), errdead(1M), errdemon(1M), errfile(4).

NAME

errstop – terminate the error-logging demon

SYNOPSIS

/etc/errstop [namelist]

DESCRIPTION

The error-logging demon *errdemon*(1M) is terminated by using *errstop*. This is accomplished by executing *ps*(1) to determine the demon'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). signal(2).

NAME

ex – text editor

SYNOPSIS

ex [–] [**–v**] [**–t** *tag*] [**–r**] [**–R**] [+*command*] [**–l**] name ...

DESCRIPTION

Ex is the root of a family of editors: *ex* and *vi*. *Ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base *terminfo*(4) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1).

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the

editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of filenames, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, filenames in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming filenames and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join** (**j**) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

–	Suppress all interactive-user feedback. This is useful in processing editor scripts.
–v	Invokes *vi*.

-t *tagfR*	Edit the file containing the *tag* and position the editor at its definition.
-r *file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
-R	*Readonly* mode set, prevents accidentally overwriting the file.
+*command*	Begin editing by executing the specified editor search or positioning *command*.
-l	**LISP** mode; indents appropriately for lisp code, the () {} [[and]] commands in *vi* are modified to have meaning for *lisp*.

The *name* argument indicates files to be edited.

Ex States

Command	Normal and initial state. Input prompted for by :. Your kill character cancels partial command.
Insert	Entered by **a** **i** and **c**. Arbitrary text may be entered. Insert is normally terminated by line having only . on it, or abnormally with an interrupt.
Visual	Entered by **vi**, terminates with **Q** or ^\.

Ex command names and abbreviations

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**			unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	window	**z**
insert	**i**	rewind	**rew**	escape	**!**
join	**j**	set	**se**	lshift	**<**
list	**l**	shell	**sh**	print next	CR
map		source	**so**	resubst	**&**
mark	**ma**	stop	**st**	rshift	**>**
move	**m**	substitute	**s**	scroll	**^D**

Ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
-	previous	*'x*	marked with *x*

+*n*	*n* forward	''	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment var.
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp		( ) { } are s-exp's
list		print ^I for tab, $ at end
magic		. [* special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in *vi*
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any char in *str*
[↑*str*]	... not in *str*
[*x*-*y*]	... between *x* and *y*
*	any number of preceding

AUTHOR

Vi and *ex* are based on software developed by The University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/usr/lib/terminfo	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), edit(1), grep(1), sed(1), vi(1), curses(3X), term(4), terminfo(4).
CTIX Programmer's Guide.

CAVEATS AND BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '–' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

Due to export restrictions, encryption features are not available.

NAME

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [–tabstop] [–tab1,tab2,...,tabn] [file ...]
unexpand [**–a**] [file ...]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default only leading blanks and tabs are reconverted to maximal strings of tabs. If the **–a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The 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. # '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 expression.

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

factor – factor a number

SYNOPSIS

factor [number]

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^{56} (about 7.2×10^{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.

DIAGNOSTICS

"Ouch" for input out of range or for garbage input.

NAME

ff – list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

Ff reads the i-list and directories of the *special* file, assuming it to be a file system, saving i-node data for files which match the selection criteria. Output consists of the path name for each saved i-node, plus any other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, –*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

–I	Do not print the i-node number after each path name.
–l	Generate a supplementary list of all path names for multiply linked files.
–p *prefix*	The specified *prefix* will be added to each generated path name. The default is ..
–s	Print the file size, in bytes, after each path name.
–u	Print the owner's login name after each path name.
–a *n*	Select if the i-node has been accessed in *n* days.
–m *n*	Select if the i-node has been modified in *n* days.
–c *n*	Select if the i-node has been changed in *n* days.
–n *file*	Select if the i-node has been modified more recently than the argument *file*.
–i *i-node-list*	Generate names for only those i-nodes specified in *i-node-list*.

EXAMPLES

To generate a list of the names of all files on a specified

file system:

```
ff -I /dev/rdsk/c0d0s1
```

To produce an index of files and i-numbers which are on a file system and have been modified in the last 24 hours:

```
ff -m -1 /dev/c0d0s1 > /log/incbackup/usr/tuesday
```

To obtain the path names for i-nodes 451 and 76 on a specified file system:

```
ff -i 451,76 /dev/rdsk/c0d1s3
```

SEE ALSO

finc(1M), find(1), frec(1M), ncheck(1M).

BUGS

Only a single path name out of any possible ones will be generated for a multiply linked i-node, unless the **-l** option is specified. When **-l** is specified, no selection criteria apply to the names generated. All possible names for every linked file on the file system will be included in the output.

On very large file systems, memory may run out before *ff* does.

NAME

file – determine file type

SYNOPSIS

file [**–c**] [**–f** ffile] [**–m** mfile] arg ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0 (see *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.

File uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

The **–m** option instructs *file* to use an alternate magic file.

The **–c** flag causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **–c**.

SEE ALSO

ar(1), ld(1).

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The *fcntl*(2) function provides for control over open files. The *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
#define   O_WRONLY      1
#define   O_RDWR        2
#define   O_NDELAY      04      /* Non-blocking I/O */
#define   O_APPEND      010     /* append
                                 (writes guaranteed at the end) */
#define   O_SYNC        020     /* synchronous write option */
#define   O_DIRECT      020000  /* perform direct I/O */
#define   O_NODIRECT    040000  /* disable direct I/O */

/* Flag values accessible only to open(2) */
#define   O_CREAT       00400   /* open with file create
                                 (uses third open arg)*/
#define   O_TRUNC       01000   /* open with truncation */
#define   O_EXCL        02000   /* exclusive open */

/* fcntl(2) requests */
#define   F_DUPFD       0       /* Duplicate fildes */
#define   F_GETFD       1       /* Get fildes flags */
#define   F_SETFD       2       /* Set fildes flags */
#define   F_GETFL       3       /* Get file flags */
#define   F_SETFL       4       /* Set file flags */
#define   F_GETLK       5       /* Get blocking file locks */
#define   F_SETLK       6       /* Set or clear file locks and fail
                                       on busy */
#define   F_SETLKW      7       /* Set or clear file locks and wait
                                       on busy */

/* file segment locking control structure */
struct flock {
        short     l_type;
        short     l_whence;
        long      l_start;
        long      l_len;     /* if 0 then until EOF */
        int       l_pid;     /* returned with F_GETLK */
```

```
/* file segment locking types */
#define   F_RDLCK          01        /* Read lock */
#define   F_WRLCK          02        /* Write lock */
#define   F_UNLCK          03        /* Remove locks */
```

SEE ALSO

fcntl(2), open(2).

NAME

finc – fast incremental backup

SYNOPSIS

finc [selection-criteria] file-system raw-tape

DESCRIPTION

Finc selectively copies the input *file-system* to the output *raw-tape*. The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by *labelit* (see *volcopy*(1M)). The selection is controlled by the *selection-criteria*, accepting only those i-nodes/files for whom the conditions are true.

It is recommended that production of a *finc* tape be preceded by the *ff* command, and the output of *ff* be saved as an index of the tape's contents. Files on a *finc* tape may be recovered with the *frec* command.

The argument **n** in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where $+n$ means more than n, $-n$ means less than n, and n means exactly n. A day is defined as a 24 hours.

–a *n*	True if the file has been accessed in *n* days.
–m *n*	True if the file has been modified in *n* days.
–c *n*	True if the i-node has been changed in *n* days.
–n *file*	True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a tape consisting of all files from file-system **/usr** modified in the last 48 hours:

```
finc -m -2 /dev/dsk/c0d0s1 /dev/rmt0
```

SEE ALSO

cpio(1), ff(1M), frec(1M), volcopy(1M).

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 descriptions, the argument *n* is used as a decimal integer where $+n$ means more than n, $-n$ means less than n and n means exactly n.

–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.

–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, respectively.

–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*[**c**] — True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.

–atime *n* — True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.

-mtime *n* — True if the file has been modified in *n* days.

-ctime *n* — True if the file has been changed in *n* days.

-exec *cmd* — True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current path name.

-ok *cmd* — Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

-print — Always true; causes the current path name to be printed.

-cpio *device* — Always true; write the current file on *device* in *cpio* (4) format (5120-byte records).

-newer *file* — True if the current file has been modified more recently than the argument *file*.

-inum *n* — True if the current file is inode number *n*.

-depth — Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.

(*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 (**!** is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**-o** is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not

been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \)
-atime +7 -exec rm {} \;
```

FILES

/etc/passwd, /etc/group

SEE ALSO

chmod(1), cpio(1), sh(1), test(1), stat(2), cpio(4), fs(4).

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [-columns] [file ...]

DESCRIPTION

Fold produces a folded version of its input, inserting newlines so that none of its output lines is wider than *columns*. If *columns* is omitted, folding is done at 80 columns.

If tabs are present in the input, *columns* should be a multiple of eight.

SEE ALSO

expand(1)

WARNING

Overstriking can be spoiled by folding.

NAME

frec – recover files from a backup tape

SYNOPSIS

/etc/frec [**–p** path] [**–f** reqfile] raw-tape i-number:name ...

DESCRIPTION

Frec recovers files from the specified *raw-tape* backup tape written by *volcopy*(1M) or *finc*(1M), given their *i-numbers*. The data for each recovery request will be written into the file given by *name*.

The **–p** option allows you to specify a default prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified, i.e., that do not begin with / or ./. If any directories are missing in the paths of recovery *names* they will be created.

–p *path* — Specifies a prefixing *path* to be used to fully qualify any names that do not start with / or ./.

–f *reqfile* — Specifies a file which contains recovery requests. The format is i-number:newname, one per line.

EXAMPLES

To recover a file, i-number 1216 when backed-up, into a file named **junk** in your current working directory:

```
frec /dev/rmt0 1216:junk
```

To recover files with i-numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/joe/a.c**:

```
frec -p /usr/src/cmd /dev/rmt0 14156:a
 1232:b 3141:/usr/joe/a.c
```

SEE ALSO

cpio(1), ff(1M), finc(1M), volcopy(1M).

BUGS

While paving a path (i.e., creating the intermediate directories contained in a pathname) *frec* can only recover i-node fields for those directories contained on the tape and requested for recovery.

NAME

fsck, dfsck – file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [**–y**] [**–n**] [**–s***c:s*] [**–s**] [**–S***c:s*] [**–S**] [**–t** file] [**–q**] [**–D**] [**–f**] [**–p**] [**–bB**] [**–O**] [**M**] [file-systems]

/etc/dfsck [options1] filsys1 ... – [options2] filsys2 ...

DESCRIPTION

Fsck

Fsck audits and interactively repairs inconsistent conditions for CTIX system files. If the file system is consistent, 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 some 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. Upon completion *fsck* reports the number of used and free 1024-byte blocks and the number of files in the filesystem.

Modifying a mounted file system requires special precautions by *fsck*, because a single *sync (2)* will undo all of *fsck's* repair work. To prevent this, *fsck* performs a *syslocal*(2) **RESYNC.** The system call forces CTIX to reread the superblock from the disk.

Fsck has more consistency checks than its predecessors *check*, *dcheck*, *fcheck*, and *icheck* combined.

The following options 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.

–s*c:s*

–s Ignore the actual free list or bit map and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent.

If *c:s* is given on a standard file system, the free list is organized with *c* blocks per cylinder and *s* blocks skipped. If *c:s* is omitted, the values originally specified to *mkfs* are used. If these values were not specified, then the value *400:7* is used.

-S*c:s*

-S Conditionally reconstruct the free list or bit map. This option is like **-s** above except that the free list or bit map 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 or bit map 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.

-q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list or bit map salvaged.

-D Directories are checked for consistency. Useful after system crashes. The following inconsistencies are sought:

- Entries with null names but nonzero i-numbers.
- Entries that are not padded to full size with nulls.
- Invalid . and .. entries.
- Names that contain "/".
- Final blocks that are not cleared past end-of-file.

-f Fast check. Check block and sizes (Phase 1) and check the free list or bit map (Phase 5). The free list or bit map will be reconstructed (Phase 6) if it is necessary.

-p Preen file systems only; intended for auto boot. No operator input is prompted for. Instead, *fsck*

applies standard fixes whenever the fix doesn't involve loss of data. Only the following problems are subject to this kind of fix:

Unreferenced i-nodes.
Link counts in i-nodes too large.
Missing blocks in the free list.
Blocks in the free list also in files.
Counts in the super block wrong.

Any problem not of this type causes *fsck* to terminate with an error status. The startup script that runs *fsck* (normally **/etc/bcheckrc**) can specify the **-p** option to *fsck* and make a normal boot contingent upon a normal *fsck* return status.

-b or **-B**
Resync file system after modifying (if file system was mounted).

-M Convert file system to new bit map free list format.

-O Convert file system to old free list format.

Both **-M** and **-O** imply **-s**.

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 i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   I-node number out of range.
8. Super Block checks:
   More than 65536 i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. If it is empty, *fsck* will silently remove them. *Fsck* will force the reconnection of nonempty directories. The name assigned is the i-node 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. The **lost+found** directory is normally created by running *mklost+found*(1M) just after the file system is created with *mkfs*(1M).

Checking the raw device is almost always faster and should almost always be used.

Dfsck

Dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A – is the separator between the file system groups.

The *dfsck* program permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Do not use *dfsck* to check the *root* file system.

NOTE

The **–b** option should nearly always be used.
The raw device should always be used with mounted file systems.

FILES

/etc/checklist	contains default list of file systems to check.
/etc/checkall	optimizing *dfsck* shell file.

SEE ALSO

clri(1M), init(1M), mklost+found(1M), ncheck(1M), checklist(4), fs(4).
MiniFrame Administrator's Reference Manual.
MightyFrame Administrator's Reference Manual.

BUGS

I-node numbers for . and .. in each directory should be checked for validity.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

If **-p** was specified and preening was inadequate, a nonzero status is returned.

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:

#	absolute address
i	convert from i-number to i-node address
b	convert to byte address
d	directory slot offset
+,–,*,/	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=–	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility

F	buffer status
X	hexadecimal or octal address flip-flop (default is hexadecimal)
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes
s or **S**	print as superblock
x	print as hexadecimal words
h	print as hexadecimal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a

directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 – 12)
at	access time
mt	modification time
maj	major device number
min	minor device number
si	#inodes field in superblock
sf	#blks field in superblock
sd0	s_dinfo[0] in superblock
sd1	s_dinfo[1] in superblock
=BS	set a blank superblock with file system type 1K and a magic number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.
2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how

	several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current i-node as directory entries.
512.ps	prints the superblock

SEE ALSO

fsck(1M), dir(4), fs(4).

NAME

fsplit – split f77, ratfor, or efl files

SYNOPSIS

fsplit options files

DESCRIPTION

Fsplit splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Procedure *X* is put in file *X*.**f**, *X*.**r**, or *X*.**e** depending on the language option chosen, with the following exceptions: *main* is put in the file *MAIN*.[**efr**] and unnamed *blockdata* segments in the files *blockdataN*.[**efr**] where *N* is a unique integer value for each file.

The following *options* pertain:

–f (default) Input files are *f77*.

–r Input files are *ratfor*.

–e Input files are *Efl*.

–s Strip *f77* input lines to 72 or fewer characters with trailing blanks removed.

SEE ALSO

csplit(1), split(1).

NAME

ftp – file transfer program

SYNOPSIS

ftp [**-v**] [**-d**] [**-i**] [**-n**] [**-g**] [host]

DESCRIPTION

Ftp is the user interface to the ARPANET standard File Transfer Protocol. The program copies files to and from a remote node. It is more general than *rcp*(1N), because a File Transfer Protocol server is available under a wider range of operating systems.

The client node with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands, the prompt "ftp>" is displayed.

COMMANDS

The following commands are recognized by *ftp*. Each machine session begins with one or more **open** commands and and ends with one or more **close** or a single **bye** command.

! Invoke a shell on the local machine.

append *local-file* [*remote-file*]
Append *local-file* to a file on the remote machine. If *remote-file* is left unspecified, the remote file is named after the local file. File transfer uses the current setting for **type**.

ascii Set the file transfer type to network ASCII . This is the default type.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary Set the file transfer type to support binary image transfer.

bye Terminate the FTP session with all the remote servers and exit *ftp*.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close Terminate the FTP session with the current remote server, and return to the command interpreter.

copy *host1:file1 host2:file2*
Copy *file1* of remote host *host1* to *file2* of remote host *host2* . Connection to *host1* and *host2* must be opened prior to this command. The current setting for **type** must be the same for both remote servers.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified, it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string "-->".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, place the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, output comes to the terminal.

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine. The current setting for **type** is used while transferring the file.

glob
Toggle file name globbing. With file name globbing enabled, each local file or pathname is processed for shell metacharacters. Remote files specified in mutiple item commands such as **mput** are globbed by the remote server. If globbing is disabled, all files and pathnames are treated literally.

hash
Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of *command*. If no argument

is given, *ftp* prints a list of the known commands.

lcd [*directory*] Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, the output is sent to the terminal.

mdelete *remote-files*
Delete the specified files on the remote machine. If globbing is enabled, the specification of remote files will first be expanded using **ls**.

mdir *remote-files local-file*
Obtain a directory listing of multiple files on the remote machine and place the result in *local-file*.

mget *remote-files*
Retrieve the specified files from the remote machine and place them in the current local directory. If globbing is enabled, the specification of remote files will first be expanded using **ls**.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Obtain an abbreviated listing of multiple files on the remote machine and place the result in *local-file*.

mput *local-files* Transfer multiple local files from the current local directory to the current working directory on the remote machine.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP

server at that port. If the auto-login option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below). Connection to *host* becomes the current connection. Note that multiple connections can be made with the **open** command. The current connection can be changed by using the **open** command for an already connected host.

prompt Toggle interactive prompting (on by default). Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off, any **mget** or **mput** will transfer all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file. File transfer uses the current setting for **type**.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2* ...
The arguments specified are sent, verbatim, to the remote FTP server. A single FTP reply code is expected in return.

recv *remote-file* [*local-file*]
A synonym for **get**.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server as well.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for **put**.

sendport Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status Show the current status of *ftp*.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*]
Set the file transfer type to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII .

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. Unless *ftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*] A synonym for **help**.

Command arguments which have embedded spaces may be quoted with quote (") marks.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name "-" is specified, the standard input (for reading) or standard output (for writing) is used.

2) If the first character of the file name is "|", the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ""| ls -lt"". A particularly useful example of this mechanism is: "dir |more".

3) Failing the above checks, if "globbing" is enabled, local file names with shell metacharacters are expanded.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of "ascii", "image" (binary), "ebcdic", and "local byte size" (for PDP-10s and PDP-20s mostly). *Ftp* supports the ASCII and image types of file transfer.

Ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

The **-v** (verbose on) option forces *ftp* to show all responses from the remote server, as well as report on data transfer statistics.

The **-n** option restrains *ftp* from attempting "auto-login" upon initial connection. If auto-login is enabled, *ftp* will check the *.netrc* file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will use the login name on the local machine as the user identity on the remote machine, and prompt for a password and, optionally, an account with which to login.

The **-i** option turns off interactive prompting during multiple file transfers.

The **-d** option enables debugging.

The **-g** option disables file name globbing.

WARNINGS

Many FTP server implementations do not support the experimental operations such as print working directory.

Aborting a file transfer does not work correctly; if one attempts this, the local *ftp* will likely have to be killed by hand.

SEE ALSO

rcp(1N).

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [**–d**] [**–l**] [**–t***timeout*]

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Protocol server process. It is normally executed by the startup file, **/etc/rc**.

If the **–d** option is specified, each socket created will have debugging turned on (SO_DEBUG). With debugging enabled, the system will trace all TCP packets sent and received on a socket.

If the **–l** option is specified, each FTP session is logged on the standard output. This allows a line of the form

/etc/ftpd –l > /tmp/ftplog

to be used to conveniently maintain a log of FTP sessions.

The FTP server will timeout an inactive session after 60 seconds. If the **–t** option is specified, the inactivity timeout period will be set to *timeout*.

The FTP server currently supports the following FTP requests; case is not distinguished.

Request	**Description**
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -lg")
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PASV	set the server in passive mode
PORT	specify data connection port
QUIT	terminate session
RETR	retrieve a file
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory

XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining FTP requests specified in Internet RFC 765 are recognized, but not implemented.

Ftpd interprets file names according to the "globbing" conventions used by the shell.

Ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */etc/ftpusers*, if that file exists.

3) If the user name is "anonymous" or "ftp", an anonymous ftp account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password; by convention this is given as the client host's name.

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" home directory be constructed with care; the following rules are recommended.

$HOME	Make the home directory owned by "ftp" and unwritable by anyone.
$HOME/bin	Make this directory owned by the superuser and unwritable by anyone. The program *ls*(1) must be present to support the list commands. This program should have mode 111.
$HOME/etc	Make this directory owned by the superuser and unwritable by anyone. The files *passwd*(5) and *group*(5) must be present for the *ls* command to work properly. These files should be mode 444.
$HOME/pub	Make this directory mode 777 and owned by "ftp." Users should then place files

which are to be accessible via the anonymous account in this directory.

SEE ALSO

ftp(1N).

"File Transfer Protocol," RFC 765 in *Internet Protocol Transition Workbook,* March 1982. Network Information Center, SRI International, Menlo Park, CA 94025.

WARNINGS

There is no support for aborting commands.

The anonymous account is inherently dangerous and should be avoided when possible.

The server must run as the superuser to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the superuser only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**–ku**] files [–] [[**–ku**] files]

DESCRIPTION

Fuser lists the process IDs of the processes using the *files* specified as arguments. For block special devices, all processes using any file on that device are listed. The process ID is followed by **c**, **p** or **r** if the process is using the file as its current directory, the parent of its current directory (only when in use by the system), or its **root** directory, respectively. If the **–u** option is specified, the login name, in parentheses, also follows the process ID. In addition, if the **–k** option is specified, the SIGKILL signal is sent to each process. Only the super-user can terminate another user's process (see *kill*(2)). Options may be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

EXAMPLES

fuser –ku /dev/dsk/c0d01s1
: When run by the superuser, terminates all processes that are preventing cartridge file systems from being unmounted, listing the process ID and login name of each process as it is killed.

fuser –u /etc/passwd
: Lists process IDs and login names of processes that have the password file open.

fuser –ku /dev/dsk/c0d0s1 –u /etc/passwd
: Does both of the above examples in a single command line.

FILES

/unix	for namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M), ps(1), kill(2), signal(2).

NAME

fwtmp, wtmpfix – manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**–ic**]
/usr/lib/acct/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 is not currently available on MiniFrame. *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 performed, *acctcon1* will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to **/etc/wtmp**. The first record is the old date denoted by the string **old time** placed in the line field and the flag OLD_TIME placed in the type field of the <**utmp.h**> structure. The second record specifies the new date and is denoted by the string **new time** placed in the line field and the flag NEW_TIME placed in the type field. *Wtmpfix* uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, *wtmpfix* will check the validity of the name field to ensure that it consists soley of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to INVALID and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctcon1* will fail when processing connect accounting records.

FILES

/etc/wtmp
/usr/include/utmp.h

SEE ALSO

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

NAME

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*(4)), 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:

- **c***n* Select character set *n*, *n* between 0 and 5 (see the *HP7221A Plotter Operating and Programming Manual*, *Appendix A*).
- **p***n* Select pen numbered *n*, *n* between 1 and 4 inclusive.
- **r***n* Window on GPS region *n*, *n* between 1 and 25 inclusive.
- **s***n* Slant characters *n* degrees clockwise from the vertical.
- **u** Window on the entire GPS universe.
- **xd***n* Set x displacement of the viewport's lower left corner to *n* inches.
- **xv***n* Set width of viewport to *n* inches.
- **yd***n* Set y displacement of the viewport's lower left corner to *n* inches.
- **yv***n* 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.
- **r***n* Display GPS region *n*, *n* between 1 and 25 inclusive.
- **u** Display the entire GPS universe.

SEE ALSO

ged(1G), graphics(1G), gps(4).

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.
- **r***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.

   $*n* enters the previous *point* numbered *n*.

   >*x* labels the last *point* entered with the upper case letter *x*.

   $*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	[**–e**cho,**s**tyle,**w**eight] *points*
Box	[**–e**cho,**s**tyle,**w**eight] *points*
Circle	[**–e**cho,**s**tyle,**w**eight] *points*
Hardware	[**–e**cho] *text points*
Lines	[**–e**cho,**s**tyle,**w**eight] *points*
Text	[**–a**ngle,**e**cho,**h**eight,**m**id-point,**r**ight-point,**t**ext,**w**eight] *text points*

Edit commands:

Delete	(– (**u**niverse or **v**iew) or *points*)
Edit	[**–a**ngle,**e**cho,**h**eight,**s**tyle,**w**eight] (– (**u**niverse or **v**iew) or *points*)
Kopy	[**–e**cho,**p**oints,**x**] *points pivot destination*
Move	[**–e**cho,**p**oints,**x**] *points pivot destination*
Rotate	[**–a**ngle,**e**cho,**k**opy,**x**] *points pivot destination*
Scale	[**–e**cho,**f**actor,**k**opy,**x**] *points pivot destination*

View commands:

coordinates	*points*
erase	
new-display	
object-handles	(– (**u**niverse or **v**iew) or *points*)
point-handles	(– (**l**abelled-points or **u**niverse or **v**iew) or *points*)
view	(– (**h**ome or **u**niverse or **r**egion) or [**–x**] *pivot destination*)
x	[**–v**iew] *points*
zoom	[**–o**ut] *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 specifed 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 *keyletter*[*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*n*	Angle of *n* degrees.
echo	When true, echo additions to the display buffer.
factor*n*	Scale factor is *n* percent.
height*n*	Height of *text* is *n* universe-units ($0 \leq n < 1280$).
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*type*	Line style set to one of following *types*: **so** solid; **da** dashed; **dd** dot-dashed; **do** dotted

	ld	long-dashed

text When false, *text* strings are outlined rather than drawn.

weight*type* Sets line weight to one of following *types*:

n	narrow
m	medium
b	bold

Area options:

home Reference the home-window.

out Reduce magnification.

region*n* 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** connect 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 <**cr**>. Multiple lines of

text may be entered by preceding a **cr** with a backslash (i.e., **\cr**). The **T**ext 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 **H**ardware 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 <**cr**>. This is useful to reference a single *point.* If only one *point* is entered, the location of the <**cr**> 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 **u**niverse 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 **M**ove)

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 **k**opy 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 **f**actor percent. If the **k**opy 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 **p**oint-handles)
labels object-handles (and/or point-handles) that lie within the defined area with **O** (or **P**). Point-handles identifies labeled points when the **l**abelled-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 **h**ome, **u**niverse, and **r**egion display particular windows in the universe.

x indicates the center of a defined area. Option **v**iew indicates the center of the screen.

zoom
decreases (**z**oom **o**ut) 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 **Q**uit
exit from *ged*. **Q**uit 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 CTIX system command.

? lists *ged* commands.

SEE ALSO

gdev(1G), graphics(1G), sh(1), gps(4).

NAME

get – get a version of an SCCS file

SYNOPSIS

get [**–r**SID] [**–c**cutoff] [**–i**list] [**–x**list] [**–w**string] [**–a**seq-no.] [**–k**] [**–e**] [–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.

–r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the **–e** keyletter is also used), as a function of the SID specified.

–c*cutoff* *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **–c7502** is equivalent to **–c750228235959**. Any number of non-

numeric characters 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*(1C) command:

```
~!get "-c%E% %U%" s.file
```

-e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The **-e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *get*s for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get -e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **-e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **-k** keyletter in place of the **-e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **-e** keyletter is used.

-b Used with the **-e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)
Note: A branch *delta* may always be created from a non-leaf *delta*.

-i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved"

column of Table 1.

-x*list* A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **-i** keyletter for the *list* format.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **-k** keyletter is implied by the **-e** keyletter.

-l[**p**] Causes a delta summary to be written into an *l-file*. If **-lp** is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **-s** keyletter is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **-m** keyletter generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

-t Used to access the most recently created ("top") delta in a given release (e.g., **-r1**), or release and level (e.g., **-r1.2**).

-w *string* Substitute *string* for all occurrences of **%W%** when *get*ing the file.

-a*seq-no.* 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 1. Determination of SCCS Identification String

SID* Specified	−b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

\# Successor.

† The **−b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of − means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is

interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing ***identification keywords*** with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file (see ***admin***(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file (see ***admin***(1)).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see ***admin***(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by ***what***(1).
%W%	A shorthand notation for constructing ***what***(1) strings for CTIX system program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing ***what***(1) strings for non-CTIX system program

files.
%A% = %Z%%Y% %M% %I%%Z%

FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
* otherwise.

b. A blank character if the delta was applied or was not applied and ignored;
* if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was not applied:
"I": Included.
"X": Excluded.
"C": Cut off (by a **−c** keyletter).

d. Blank.

e. SCCS identification (SID).

f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.

h. Blank.

i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an **-e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **-e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **-i** keyletter argument if it was present, followed by a blank and the **-x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

admin(1), delta(1), help(1), prs(1), what(1), sccsfile(4).
CTIX Programmer's Guide, Section 9.

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 does not, then only one file may be named when the **-e** keyletter is used.

NAME

getopt – 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. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The positional parameters ($1 $2 . . .) of the shell are reset so that each option is preceded by a – and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)        FLAG=$i; shift;;
        -o)             OARG=$2; shift 2;;
        --)             shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

sh(1), getopt(3C).

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

getty – set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**-h**] [**-t** timeout] line [speed [type [linedisc]]]
/etc/getty -c file

DESCRIPTION

Getty is a program that is invoked by ***init***(1M). It is the second process in the series, (***init-getty-login-shell***) that ultimately connects a user with the CTIX system. Initially *getty* generates a system identification message from the values returned by the ***uname***(2) system call. Then, if **/etc/issue** exists, it outputs this to the user's terminal, followed finally by the login message field for the entry it is using from **/etc/gettydefs**. *Getty* reads the user's login name and invokes the ***login***(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **-h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **-t** flag plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, ***speed***, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* at what speed to initially run, what the login message should look like, what the inital tty settings are, and what speed to try next should the user indicate that the speed is inappropriate. (by typing a <***break***> character). The default ***speed*** is 9600 baud. The optional third argument, ***type***, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands the following types:

none	default
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard HP45
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type

to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, LDISC0.

When given no optional arguments, *getty* sets the *speed* of the interface to 9600 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, newline characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

In addition to the standard UNIX system erase and kill characters (**#** and **@**), *getty* also understands **\b** and **^U**. If the user uses a **\b** as an erase, or **code–U** as a kill character, *getty* sets the standard erase character and/or kill character to match.

Getty also understands the "standard" ESS protocols for erasing, killing and aborting a line, and terminating a line. If *getty* sees the ESS erase character, **_**, or kill character, **$**, or abort character, **&**, or the ESS line terminators, **/** or **!**, it arranges for this set of characters to be used for these functions.

Finally, *login* is called with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment (see *login*(1)).

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1C), init(1M), login(1), ioctl(2), gettydefs(4), inittab(4), tty(7).

BUGS

While *getty* does understand simple single character quoting conventions, it is not possible to quote the special control characters that *getty* uses to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. Therefore it is not possible to login via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. They will always be interpreted as having their special meaning as described above.

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 terminal, it returns a NULL pointer. 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(4).

DIAGNOSTICS

Returns the NULL pointer if *name* is not found.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get network entry

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetent ( )

struct netent *getnetbyname (name)
char *name;

struct netent *getnetbyaddr (net)
long net;

setnetent (stayopen)
int stayopen

endnetent ( )
```

DESCRIPTION

Getnetent, *getnetbyname*, and *getnetbyaddr* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network data base, **/etc/networks**.

```
struct netent {
        char    *n_name;        /* official name of net */
        char    **n_aliases;    /* alias list */
        int     n_addrtype;     /* net number type */
        long    n_net;          /* net number */
};
```

The members of this structure are:

n_name	The official name of the network.
n_aliases	A zero-terminated list of alternate names for the network.
n_addrtype	The type of the network number returned; currently only AF_INET.
n_net	The network number. Network numbers are returned in machine byte order.

Getnetent reads the next line of the file, opening the file if necessary.

Setnetent opens and rewinds the file. If the *stayopen* flag is non-zero, the network data base will not be closed after each call to *getnetent* (either directly, or indirectly through one of the other getnet calls).

Endnetent closes the file.

Getnetbyname and *getnetbyaddr* sequentially search from the beginning of the file until a matching net name or

Endhostent closes the file.

Gethostbyname and *gethostbyaddr* sequentially search from the beginning of the file until a matching host name or host address is found, or until EOF is encountered. Host addresses are supplied in network order.

FILES

/etc/hosts

SEE ALSO

hosts(4N).
CTIX Internetworking Manual.

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. Only the Internet address format is currently understood.

NOTE

This command is for use with a special version of the CTIX kernel that supports networking protocols.

NAME

gethostname – get name of current host

SYNOPSIS

gethostname (name, namelen)
char *name;
int namelen;

DESCRIPTION

Gethostname returns the standard host name for the current processor, as previously set by *setuname*(1M). The parameter *namelen* specifies the size of the *name* array. The returned name is null-terminated unless insufficient space is provided.

RETURN VALUE

If the call succeeds, a value of 0 is returned. If the call fails, then a value of –1 is returned and an error code is placed in the global location *errno*.

ERRORS

The following errors may be returned by these calls:

[EFAULT]	The *name* or *namelen* parameter gave an invalid address.
[EPERM]	The caller was not the super-user.

SEE ALSO

setuname(1M).
CTIX Internetworking Manual.

BUGS

Host names are limited to 9 characters.

NOTE

This command is for use with a special version of the CTIX kernel that supports networking protocols.

net address is found, or until EOF is encountered. Network numbers are supplied in host order.

FILES

/etc/networks

SEE ALSO

networks(4N).
CTIX Internetworking Manual.

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. Only Internet network numbers are currently understood. Expecting network numbers to fit in no more than 32 bits is probably naive.

NOTE

This command is for use with a special version of the CTIX kernel that supports networking protocols.

NAME

getopt – get option letter from argument vector

SYNOPSIS

```
int getopt (argc, argv, optstring)
int argc;
char **argv, *opstring;

extern char *optarg;
extern int optind, opterr;
```

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*. This error message may be disabled by setting *opterr* to a non-zero value.

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:

```
main (argc, argv)
int argc;
char **argv;
{
        int c;
        extern char *optarg;
        extern int optind;
        .
        .
        .
        while
```

NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocol entry

SYNOPSIS

```
#include <netdb.h>

struct protoent *getprotoent ( )

struct protoent *getprotobyname (name)
char *name;

struct protoent *getprotobynumber (proto)
int proto;

setprotoent (stayopen)
int stayopen

endprotoent ( )
```

DESCRIPTION

Getprotoent, *getprotobyname*, and *getprotobynumber* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol data base, **/etc/protocols**.

```
struct protoent {
        char    *p_name;      /* official name of protocol */
        char    **p_aliases;  /* alias list */
        long    p_proto;      /* protocol number */
};
```

The members of this structure are:

p_name The official name of the protocol.

p_aliases A zero-terminated list of alternate names for the protocol.

p_proto The protocol number.

Getprotoent reads the next line of the file, opening the file if necessary.

Setprotoent opens and rewinds the file. If the *stayopen* flag is non-zero, the network data base will not be closed after each call to *getprotoent* (either directly, or indirectly through one of the other getproto calls).

Endprotoent closes the file.

Getprotobyname and *getprotobynumber* sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until EOF is encountered.

FILES

/etc/protocols

SEE ALSO

protocols(4N).
CTIX Internetworking Manual.

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. Only the Internet protocols are currently understood.

NOTE

This command is for use with a special version of the CTIX kernel that supports networking protocols.

NAME

getservent, getservbyport, getservbyname, setservent, endservent – get service entry

SYNOPSIS

```
#include <netdb.h>

struct servent *getservent ( )

struct servent *getservbyname (name, proto)
char *name, *proto;

struct servent *getservbyport (port, proto)
int port; char *proto;

setservent (stayopen)
int stayopen

endservent ( )
```

DESCRIPTION

Getservent, *getservbyname*, and *getservbyport* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, **/etc/services**.

```
struct    servent {
    char  *s_name;      /* official name of service */
    char  **s_aliases;  /* alias list */
    long  s_port;       /* port service resides at */
    char  *s_proto;     /* protocol to use */
};
```

The members of this structure are:

s_name — The official name of the service.

s_aliases — A zero-terminated list of alternate names for the service.

s_port — The port number at which the service resides. Port numbers are returned in network byte order.

s_proto — The name of the protocol to use when contacting the service.

Getservent reads the next line of the file, opening the file if necessary.

Setservent opens and rewinds the file. If the *stayopen* flag is non-zero, the network data base will not be closed after each call to *getservent* (either directly, or indirectly through one of the other getserv calls).

Endservent closes the file.

Getservbyname and *getservbyport* sequentially search from the beginning of the file until a matching protocol

name or port number is found, or until EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

/etc/services

SEE ALSO

getprotoent(3N), services(4N).
CTIX Internetworking Manual.

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. Expecting port numbers to fit in a 32-bit quantity is probably naive.

NOTE

This command is for use with a special version of the CTIX kernel that supports networking protocols.

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 surrounded with quotes ", in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

–a	Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **–x**).
–b	Break (disconnect) the graph after each label in the input.
–c	Character string given by next argument is default label for each point.
–g	Next argument is grid style, 0 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, do not erase before plotting.
–x [l]	If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.
–y [l]	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 is no 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 prefixes 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/bin/graf:/rbin:/usr/rbin

and the restricted shell, *rsh*, 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 CTIX system 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(4).

NAME

greek – select terminal filter

SYNOPSIS

greek [**–T**terminal]

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 $TERM (see *environ*(5)). The following *terminal*s are recognized currently:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	TEKTRONIX 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	TEKTRONIX 4014.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), eqn(1), hp(1), mm(1), nroff(1), tplot(1G), environ(5), term(5).

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.

–i Ignore upper/lower case distinction during comparisons.

–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 (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we do not know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to BUFSIZ characters; longer lines are truncated. (BUFSIZ is defined in **/usr/include/stdio.h**.)

Egrep does not recognize ranges, such as **[a–z]**, in character classes.

If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

gutil – graphical utilities

SYNOPSIS

command-name [options] [files]

DESCRIPTION

Below is a list of miscellaneous device independent utility commands found in **/usr/bin/graf**. 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*(4).

bel – send bel character to terminal

cvrtopt [=***s****tring* **f***string* **i***string* **t***string*] [*args*] – options converter

Cvrtopt 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., –r1.1) or are two-letters long must be described through options to *cvrtopt*.

Cvrtopt is usually used with *set* in the following manner as the first line of a shell procedure:

**set – `cvrtopt** =[*options*] $@`

Options to *cvrtopt* are:

s*string*	*String* accepts string values.
f*string*	*String* accepts floating point numbers as values.
i*string*	*String* accepts integers as values.
t*string*	*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 [**–r***n* **u**] [GPS *files*] – GPS to *plot*(4) filter

Gtop transforms a GPS into *plot*(4)

commands displayable by *plot* filters. GPS objects are translated if they fall within the window that circumscribes the first *file* unless an *option* is given.
Options:

r*n* translate objects in GPS **r**egion *n*.

u translate all objects in the GPS **u**niverse.

pd [*plot*(5) *files*] – *plot*(4) dump
Pd prints a human readable listing of *plot*(4) format graphical commands.

ptog [*plot*(5) *files*] – *plot*(4) to GPS filter
Ptog transforms *plot*(4) 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 on-line documentation
Whatis prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. The command **whatis** * prints out every description.
Option:

o just print command **o**ptions

yoo *file* – pipe fitting
Yoo is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that, without *yoo*, this is not usually successful as it causes a read and write on the same file simultaneously.

SEE ALSO
graphics(1G), gps(4), plot(4).

NAME

hd – hexadecimal and ascii file dump

SYNOPSIS

/usr/local/bin/hd *file*

DESCRIPTION

Hd prints a hexadecimal listing of *file*, side by side with an ASCII listing.

SEE ALSO

od(1).

NAME

head – give first few lines

SYNOPSIS

head [*–count*] [*file* ...]

DESCRIPTION

Head gives the first *count* lines of each of the specified files. If no files are specified, *head* reads the standard input. If you omit *count*, *head* prints the first 10 lines.

SEE ALSO

tail(1).

NAME

help – ask for help

SYNOPSIS

help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1 — Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **ge6**, for message 6 from the *get* command).

type 2 — Does not contain numerics (as a command, such as **get**)

type 3 — Is all numeric (e.g., **212**)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

FILES

/usr/lib/help	directory containing files of message text.
/usr/lib/help/helploc	file containing locations of help files not in **/usr/lib/help**.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

hinv – hardware inventory

SYNOPSIS

/etc/hinv option
/etc/hinv hardware-item

DESCRIPTION

Hinv provides hardware configuration information. It is used in one of two ways. For the first way, an *option* is given and the result is printed on ***stdout***. The other way involves asking about a particular hardware item and *hinv* existing with 1 if it exists and 0 otherwise.

Option is one of the following:

–p	print hardware configuration. Items are printed one per line.
–c	print CPU type
–f	print FPU type
–s	print system type
–u	print maximum number of users
–m	print total physical memory in bytes.

Hardware-item is one of the following:

68881	68881 floating-point processor
iop	terminal accelerator board
422	422 cluster board
vme	VME interface board
sn	RS-232 board *n*
serial	gives number of serial ports present
disks	gives number and types of disks present
eeprom	VME EEPROM valid for UNIX

BUGS

Hinv does not know about VME cards.

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* output. A typical use is:

nroff –h files . . . | hp

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

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*.

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), eqn(1), greek(1), nroff(1), tbl(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 in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text "disappear"; in particular, tables generated by *tbl*(1) that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to *hp* is piped through *col*(1).
Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hyphen – find hyphenated words

SYNOPSIS

hyphen [files]

DESCRIPTION

Hyphen finds all the hyphenated words ending lines in *files* and prints them on the standard output. If no arguments are given, the standard input is used; thus, *hyphen* may be used as a filter.

EXAMPLE

The following will allow the proofreading of *nroff* hyphenation in *textfile*.

```
mm textfile | hyphen
```

SEE ALSO

mm(1), nroff(1).

BUGS

Hyphen cannot cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

NAME

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).

NAME

ifconfig – configure network interface parameters

SYNOPSIS

/etc/ifconfig interface [address] [parameters]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or configure network interface parameters. *Ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address. The *interface* parameter is a string of the form "name unit", e.g. "en0", while the address is either a host name present in the host name data base, *hosts*(4), or a DARPA Internet address expressed in the Internet standard "dot notation".

The following parameters may be set with *ifconfig*:

up	Mark an interface "up".
down	Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface.
trailers	Enable the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver.
–trailers	Disable the use of a "trailer" link level encapsulation.
arp	Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addreses and 10Mb/s Ethernet addresses.
–arp	Disable the use of the Address Resolution Protocol.

Ifconfig displays the current configuration for a network interface when no optional parameters are supplied.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

intro(4N), netstat(1).

NAME

includes – determine C language preprocessor include files

SYNOPSIS

includes [option ...] file ...

DESCRIPTION

Includes determines the **#include** files necessary to compile a C language source file using the C language preprocessor *cpp*(1). *Includes* is based on *cpp*(1) and takes the same options. Multiple source files may be named on the command line. However, instead of producing preprocessed code, it produces on standard output a list of the **#include** file dependencies (directly or nested) of the named source files.

The output format is suitable for direct use in a *makefile* to be used by the *make*(1) command. For each named source file, the **#include** files are listed, one per line, preceded by the name of the source file (with the last letter of its name changed to the letter 'o'). The two names are separated by the two characters ": ". For example, if source file **pgm.c** depends only on the **#include** file **incl.h**, then the output of **includes** for the source file **pgm.c** would be:

```
pgm.o: incl.h
```

The following *options* to *includes* are recognized:

–P This option has no affect.

–C This option has no affect.

–U*name*
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system:	ibm, gcos, os, tss, unix
hardware:	interdata, pdp11, u370, u3b, vax, mc68k, mc68000, mc68010, mc68020
system variants:	RES, RT

–D*name*
–D*name*=*def*
Define *name* as if by a **#define** directive. If no =*def* is given, *name* is defined as 1.

–I*dir* Change the algorithm for searching for **#include** files whose names do not begin with / to look in *dir* before looking in the directories on

the standard list. Thus, **#include** files whose names are enclosed in " " will be searched for first in the directory of the *ifile* argument, then in directories named in **–I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the *ifile* argument is not searched.

Two special names are understood by *includes*. The name __LINE__ is defined as the current line number (as a decimal integer) as known by *includes*, and __FILE__ is defined as the current file name (as a C string) as known by *includes*. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives understood by *includes* start with lines begun by #. The directives are:

#define *name token-string*
Replace subsequent instances of *name* with *token-string*.

#define *name*(*arg, ..., arg*) *token-string*
Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list.

#undef *name* Cause the definition of *name* (if any) to be forgotten from now on.

#include "*filename*"
#include <*filename*>
Include at this point the contents of *filename* (which will then be run through *includes*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **–I** option above for more detail.

#line *integer-constant "filename"*
This directive has no affect.

#endif
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.

#ifdef *name* The lines following will be processed if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#ifndef *name* The lines following will not be processed if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**.

#if *constant-expression*
Lines following will be processed if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the **?:** operator, the unary –, **!**, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *includes* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

#else Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will be processed. And vice versa.

The test directives and the possible **#else** directives can be nested.

FILES

/usr/include standard directory for **#include** files

SEE ALSO

cc(1), cpp(1), m4(1).

DIAGNOSTICS

The error messages produced by *includes* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NAME

init, telinit – process control initialization

SYNOPSIS

/etc/init [**0123456SsQq**]

/etc/telinit [**0123456sSQqabc**]

DESCRIPTION

Init

Init is a general process spawner. Its primary role is to create processes from a script stored in the file **/etc/inittab** (see *inittab*(4)). This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a *run-level* at any given time. A *run-level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run-levels* is defined in the *inittab* file. *Init* can be in one of eight *run-levels*, **0–6** and **S** or **s**. The *run-level* is changed by having a privileged user run **/etc/init** (which is linked to **/etc/telinit**). This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was rebooted, telling it which *run-level* to change to.

Init is invoked inside the CTIX system as the last step in the boot process. The first thing *init* does is to look for **/etc/inittab** and see if there is an entry of the type *initdefault* (see *inittab*(4)). If there is, *init* uses the *run-level* specified in that entry as the initial *run-level* to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a *run-level* from the virtual system console. If an **S** (**s**) is entered, *init* goes into the *SINGLE USER* level. This is the only *run-level* that does not require the existence of a properly formatted *inittab* file. If **/etc/inittab** does not exist, then by default the only legal *run-level* that *init* can enter is the *SINGLE USER* level. In the *SINGLE USER* level the virtual console terminal **/dev/syscon** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the *SINGLE USER run-level* one of two options can be elected. First, if the shell is terminated (via an end-of-file), *init* will reprompt for a new *run-level*. Second, the *init* or *telinit* command can signal *init* and force it to change the *run-level* of the system. *Init* always trys to relink */dev/syscon* to a reasonable terminal before

opening it. It invokes *conlocate*(1M) to do this.

When *init* prompts for the new *run-level*, the operator may enter only one of the digits **0** through **6** or the letters **S** or **s**. If **S** is entered *init* operates as previously described in *SINGLE USER* mode with the additional result that **/dev/syscon** is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the previous system console, saying where the virtual terminal has been relocated.

When *init* comes up initially and whenever it switches out of *SINGLE USER* state to normal run states, it sets the *ioctl*(2) states of the virtual console, **/dev/syscon**, to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever *SINGLE USER* mode is entered. If this file does not exist when *init* wants to read it, a warning is printed and default settings are assumed.

If a **0** through **6** is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be re-prompted. If this is the first time *init* has entered a *run-level* other than *SINGLE USER*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run-level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run-level*.

Run-level **2** is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multi-user environment.

In a multi-user environment, the *inittab* file is usually set up so that *init* will create a process for each terminal on the system.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a child death signal, telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and if it exists (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp** if such a file exists.

To spawn each process in the *inittab* file, *init* reads each entry and for each entry which should be respawned, it

forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's *run-level*. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however, *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response the **init Q** or **init q** command can wake *init* to re-examine the *inittab* file.

If *init* receives a *powerfail* signal (*SIGPWR*) and is not in *SINGLE USER* mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the *run-levels* permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure. It is important to note that the powerfail entries should not use devices that must first be initialized after a power failure has occurred.

When *init* is requested to change *run-levels* (via *telinit*), *init* sends the warning signal (SIGTERM) to all processes that are undefined in the target *run-level*. *Init* waits 20 seconds before forcibly terminating these processes via the kill signal (SIGKILL).

Telinit

Telinit, which is linked to */etc/init*, is used to direct the actions of *init*. It takes a one-character argument and signals *init* via the kill system call to perform the appropriate action. The following arguments serve as directives to *init*.

0–6 tells *init* to place the system in one of the *run-levels* **0–6**.

a,**b**,**c** tells *init* to process only those **/etc/inittab** file entries having the **a**, **b** or **c** *run-level* set.

Q,**q** tells *init* to re-examine the **/etc/inittab** file.

s,**S** tells *init* to enter the single user environment. When this level change is effected, the virtual system teletype, **/dev/syscon**, is changed to the terminal from which the command was executed.

A directive to change to *run-level* **6** receives special priority. Ordinarily, a *run-level* change

received while *init* is re-examining *inittab* does not take effect until the re-examination is complete. But a directive to change to *run-level* **6** received while *init* is waiting on a *bootwait* entry is effected as soon as the command in the *bootwait* entry finishes. This special case permits a *bootwait* command to use *telinit* to stop the system initialization process before users get access to the system. *Run-level* **6** then handles the transition to single-user state: see */etc/profile*.

Telinit can only be run by someone who is super-user or a member of group **sys**.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/syscon
/dev/systty
/etc/profile

SEE ALSO

conlocate(1M), getty(1M), login(1), sh(1), who(1), kill(2), inittab(4), profile(4), utmp(4).

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.

NAME

install – install commands

SYNOPSIS

/etc/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 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 (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **–s** option.

–f *dirb* — Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to **755** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **–o** or **–s** options.

-i	Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options other than **-c** and **-f**.
-n *dirc*	If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options other than **-c** and **-f**.
-o	If *file* is found, this option saves the "found" file by copying it to OLD*file* in the directory in which it was found. This option is useful when installing a normally text busy file such as **/bin/sh** or **/etc/getty**, where the existing file cannot be removed. 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

cpset(1M), make(1).

NAME

ipcrm – remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

Ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

–q *msqid* removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

–m *shmid* removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

–s *semid* removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

–Q *msgkey* removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.

–M *shmkey* removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

–S *semkey* removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

NAME

ipcs – report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

Ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

−q Print information about active message queues.

−m Print information about active shared memory segments.

−s Print information about active semaphores.

If any of the options **−q**, **−m**, or **−s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed.

−b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

−c Print creator's login name and group name. See below.

−o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

−p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

−t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on

shared memory, last *semop*(2) on semaphores.) See below.

-a Use all print *options*. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

-C *corefile* Use the file *corefile* in place of **/dev/kmem**.

-N *namelist* The argument will be taken as the name of an alternate *namelist* (**/unix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:
- **q** message queue;
- **m** shared memory segment;
- **s** semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:
The first two characters are:
- **R** if a process is waiting on a *msgrcv*;
- **S** if a process is waiting on a *msgsnd*;
- **D** if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
- **C** if the associated shared memory segment is to be cleared when the first attach is executed;
- **-** if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

- **r** if read permission is granted;
- **w** if write permission is granted;
- **a** if alter permission is granted;
- – if the indicated permission is *not* granted.

OWNER	(all)	The login name of the owner of the facility entry.
GROUP	(all)	The group name of the group of the owner of the facility entry.
CREATOR	(a,c)	The login name of the creator of the facility entry.
CGROUP	(a,c)	The group name of the group of the creator of the facility entry.
CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	The number of messages currently outstanding on the associated message queue.
QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p)	The process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	The time the last message was sent to the associated queue.
RTIME	(a,t)	The time the last message was received from the associated queue.

CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2).

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

intro – introduction to special files

SYNOPSIS

```
#include <sys/socket.h>
                        /* internetworking only */
#include <net/route.h>
#include <net/if.h>
```

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and CTIX System device drivers. The names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding CTIX system device driver are discussed where applicable.

INTERNETWORKING

Entries that describe network protocol use are marked **(7N)**. These protocols are available only with a special version of the CTIX kernel that supports internetworking. For further information, see the *CTIX Inernetworking Manual.*

All network protocols are associated with a specific protocol-family. A ***protocol-family*** provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol-family may support multiple methods of addressing, though the current protocol implementations do not. A protocol-family is normally comprised of a number of protocols, one per ***socket***(2N) type. It is not required that a protocol-family support all socket types. A protocol-family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in ***socket***(2N). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol-family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol-family/network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism. For example, a protocol

supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families and/or address formats. The SYNOPSIS section of each network interface entry gives a sample specification of the related drivers for use in providing a system description to the *config*(1M) program. The DIAGNOSTICS section lists messages which may appear on the console and in the system error log **/usr/adm/messages** due to errors in device operation.

PROTOCOLS

The system currently supports only the DARPA Internet protocols fully. Raw socket interfaces are provided to IP protocol layer of the DARPA Internet, to the IMP link layer (1822), and to Xerox PUP-1 layer operating on top of 3Mb/s Ethernet interfaces. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

ADDRESSING

Associated with each protocol family is an address format. The following address format is supported:

```
#define AF_INET      2
        /* internetwork: UDP, TCP, etc. */
```

ROUTING

The network facilities provide limited packet routing. A simple set of data structures comprise a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. A user process, the routing demon, maintains this data base with the aid of two socket specific *ioctl*(2) commands, SIOCADDRT and SIOCDELRT. The commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by the superuser.

A routing table entry has the following form, as defined in **<net/route.h>**:

```
struct  rtentry {
                u_long      rt_hash;
                struct      sockaddr rt_dst;
                struct      sockaddr rt_gateway;
                short       rt_flags;
                short       rt_refcnt;
                u_long      rt_use;
                struct      ifnet *rt_ifp;
};
```

with *rt_flags* defined from,

```
#define RTF_UP          0x1
        /* route usable */
#define RTF_GATEWAY     0x2
        /* destination is a gateway */
#define RTF_HOST        0x4
        /* host entry (net otherwise) */
```

Routing table entries come in three types: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). When the system is booted, each network interface that is autoconfigured installs a routing table entry when it wishes to have packets sent through it. Normally the interface specifies the route through it is a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (i.e., the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (*rt_refcnt* is nonzero), the resources associated with it will not be reclaimed until further references to it are released.

The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a nonexistant entry, or ENOBUFS if insufficient resources were available to install a new route.

User processes read the routing tables through the **/dev/kmem** device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple

routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

INTERFACES

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it.

At boot time each interface which has underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address, it is expected to install a routing table entry so that messages may be routed through it. Most interfaces require some part of their address specified with an SIOCSIFADDR ioctl before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware-specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (e.g. 10Mb/s Ethernets), the entire address specified in the ioctl is used.

The following *ioctl* calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an *ifrequest* structure as its parameter. This structure has the form

```
struct  ifreq {
        char    ifr_name[16];
                /* name of interface (e.g. "ec0") */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                short   ifru_flags;
        } ifr_ifru;
#define ifr_addr ifr_ifru.ifru_addr
                /* address */
#define ifr_dstaddr     ifr_ifru.ifru_dstaddr
                /* other end of p-to-p link */
#define ifr_flags ifr_ifru.ifru_flags
                /* flags */
};
```

SIOCSIFADDR
: Set interface address. Following the address assignment, the "initialization" routine for the interface is called.

SIOCGIFADDR
: Get interface address.

SIOCSIFDSTADDR
: Set point-to-point address for interface.

SIOCGIFDSTADDR
: Get point-to-point address for interface.

SIOCSIFFLAGS
: Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.

SIOCGIFFLAGS
: Get interface flags.

SIOCGIFCONF
: Get interface configuration list. This request takes an *ifconf* structure (see below) as a value-result parameter. The *ifc_len* field should be initially set to the size of the buffer pointed to by *ifc_buf*. On return it will contain the length, in bytes, of the configuration list.

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
struct  ifconf {
        int     ifc_len;
                /* size of associated buffer */
        union {
                caddr_t  ifcu_buf;
                struct   ifreq *ifcu_req;
        } ifc_ifcu;
#define ifc_buf  ifc_ifcu.ifcu_buf
                /* buffer address */
#define ifc_req  ifc_ifcu.ifcu_req
                /* array of structures returned */
};
```

SEE ALSO
: config(1M), ioctl(2), socket(2N), intro(7).

NAME

console – console terminal

DESCRIPTION

The special file **/dev/console** designates a standard destination for system diagnostics. The kernel writes its diagnostics to this file, as does any user process with messages of systemwide importance. Unless CTIX is configured with the kernel debugger, **console** is not associated with a terminal; console messages are written to **/etc/log/confile**. If **console** is associated with a physical terminal (configured with the kernel debugger), then console messages appear on that terminal.

Note that *inittab*(4) does not normally post a *getty* on **console** because it has no source for interactive input.

Console messages are saved in a circular buffer. Reading **console** retrieves the messages and removes them from the buffer.

If CTIX is configured with the kernel debugger (see *config*(1M)), then **tty000** is associated with the console. This means that console messages also go to **tty000** and that a Control-B on **tty000** starts the kernel debugger.

The size of the console circular buffer is configured with the *config*(1M) parameter **cbufsz**. The default is 4096 bytes.

The following *ioctl*(2) commands are acceptd:

ioctl(fd, CONERR);
: *Fd* must be open to **console**. All console output is to be duplicated in the error message queue. See *err (7)*.

ioctl(fd, CONBUF);
: *Fd* must be open to **console**. No console output is to be duplicated in the error message queue. This is the initial condition.

ioctl(fd, CON_SET, port)
: *Fd* must be open to *console*. *Port* is the minor device number of the RS-232 line that will be the new debugger console; *port* must be a valid RS-232 channel. The function returns the number of the new debugger console port.

ioctl(fd, CON_LOC)
: *Fd* must be open to *console*. The function returns the number of the current debugger console port.

NAME

iv – initialize and maintain volume

SYNOPSIS

iv –iuostdwlvq *special* [*descriptionfile*]

DESCRIPTION

Iv initializes and maintains a disk volume. *Special* and *descriptionfile* specify the disk and a description file for it; these are described below. *Iv* does one of five operations, specified by the following options:

–i Completely initialize a volume. This consists of five phases:

1. Initialize *iv's* internal Volume Home Block, based on *descriptionfile* and the disk type. If the disk can support bad block handling (all types, except floppy disks on MiniFrame systems), create an internal Bad Block Table. Put bad block data from *descriptionfile* and volume's existing Bad Block Table (if any) in internal Bad Block Table.
2. Format medium.
3. Perform a surface check. If the disk can support bad block handling, add bad blocks to the Bad Block Table. If the disk cannot support bad block handling, the first bad spot causes the disk to be rejected.
4. Write out the Volume Home Block. This has the effect of dividing the volume into slices (partitions).
5. Allocate and write out the files that share the Reserved Area (slice 0) with the Volume Home Block. If the disk can support bad block handling, one of these files is the Bad Block Table. Other files are specified in *descriptionfile*.

–u Update the Volume Home Block. This is the same as **–i**, except that the second and third phases (medium formatting and surface check) are skipped.

–o Output a Volume Home Block and partition 0 to any file; requires a *descriptionfile*. The following command produces a dump tape:

iv -o /dev/rmt0 /usr/lib/iv/desc.tdump

-s Surface test. Any bad blocks discovered are added to the bad block table.

-t Tell volume description. Display volume home block in human-readable form. No description file is needed. The volume's contents are not affected.

-d Description file display. A description file that describes the current state of the volume is written to the standard output. If the Reserved Area contains a loader, the **loader** keyword's value is written as **/usr/lib/iv/loader**. If the Reserved Area contains a down load image area, the Down Load Area Description lists files whose names are of the form

/usr/lib/iv/ws*xxx*.*yyy*

where *xxx* is the numeric device identification; and *yyy* is **422** if *xxx* is even, **232** if *xxx* is odd.

The **-f** option, equivalent to **-u**, is provided for compatibility with older versions of *iv*. It should not be used, as it may disappear in future releases.

In addition to the single operation option (**-i**, **-u**, **-s**, **-t**, or **-d**) you can specify any or all of the following options:

-v Verbose display output. If the display includes the Volume Home Block, also include the bad block table.

-l A normal surface test consists of a single pass over the disk; **-l** specifies ten passes.

-w A normal surface test pass consists of a read pass; **-w** specifies a write pass before each read pass.

-q Print the size of the disk (in megabytes).

File Parameters

Special is the character special file for slice zero on the drive. This name takes the form **/dev/rdsk/c***n***d***t***s0,** where *n* is the controller number and *t* is the drive number.

Descriptionfile is a text file that describes the volume. It is required by the **-i** and **-u** options. The description file consists of five parts:

- general description
- reserved area description

- bad blocks description
- partition table description
- down load area description

Each description is separated from the next by a line that contains only a single dollar sign ($). Specifics for each of the five descriptions are given under separate headings below.

General Description

Each line in the General Description begins with a keyword. Some keywords are followed by values; the value is separated from the keyword by spaces or tabs. For example:

```
ecc
cylinders       1024
```

Each keyword is only used once. Here are the valid keywords.

type Mandatory, unless the volume is already initialized in the appropriate format. Value is disk type: "HD" for onboard ST506 hard disk, "RD" for RAM disk, "V3200" for SMD controller, and "FD" for floppy disk (MiniFrame only).

name Mandatory, unless the volume is already initialized in the appropriate format. Value is the volume name. Any characters except spaces or tabs are permitted in the volume name; the serial number of the disk is the recommended volume name. The actual name in the Volume Home Block is always exactly six characters; *iv* right truncates names that are too long and right pads with nulls names that are too short.

cylinders Mandatory, unless the volume is already initialized in the appropriate format. Value is the number of cylinders on the disk.

heads Mandatory, unless the volume is already initialized in the appropriate format. Value is the number of heads on the disk.

sectors Mandatory, unless the volume is already initialized in the appropriate format. Value is the number of physical sectors per track.

steprate Mandatory for ST506, unless the volume is already initialized in the appropriate format. Value is a number that is passed to the disk controller. The normal steprate for ST506 drives is 14; 0 can be used for slower drives. See the disk manufacturer's documentation for further information.

exchangeable If this keyword is present, the disk can be removed from its drive.

hitech (ST506 drives only) If this keyword is present, the reduced write current line to the disk is used for head-select bit 3 to allow more than eight heads.

precomp (ST506 drives only) The value is $c/16$, where c is the cylinder at which precompensation should start. See the disk manufacturer's documentation for further information.

ecc The disk has been prepared to function in ECC mode.

enetaddr (MiniFrame only) Ethernet address of the machine (boot drive only).

gap1
gap2 Gap size for SMD drives. See the disk manufacturer's documentation for further information.

Reserved **Area Description**

The Reserved Area Description describes the files that share slice zero with the volume home block. Each line in the Reserved Area Description consists of a keyword followed by one or more parameters; one or more tabs or spaces separates keywords and parameters from each other. Here are the valid keywords and their meanings. (A logical block is 1024 bytes long.)

loader Describes the loader area. The first, mandatory, parameter is the full pathname of an a.out file to put in the loader area. The second, optional, parameter is the size of the loader area in logical blocks. If the second parameter is missing, the size of the a.out file is used.

badblocktable Describes the bad block table. The first,

mandatory, parameter is the size of the bad block table in logical blocks. The second, optional, parameter is only used when an existing bad block table contains errors; this parameter is "empty" to clear the bad block table, missing otherwise.

dump Describes area to contain dump after crash. The only, mandatory, parameter, specifies the size of the dump area in logical blocks.

downloadarea
Describes area to contain system images for downloading. The only, mandatory, parameter, specifies the size of the download area in logical blocks. (The files actually put in this area are described separately. See the Down Load Area Description heading, below.)

program Describes the bootable (standalone) program area. There are three ways to specify this area. If there are two parameters, the program area is allocated in slice zero; the first parameter must be the full pathname of the file to be copied to the program area, and the second parameter must be the size of the program area in logical blocks. If there is one parameter, the program area is allocated in slice one; the parameter must be the full pathname of the file to be copied to the program area. If no program area is desired, omit the **program** line from the Reserved Area Description.

All lines valid for the Reserved Area Description are optional. However, the bad block table is mandatory on a volume which supports bad block handling; the loader area is mandatory on a volume which is to hold an operating system; and a dump area is recommended on a volume which is to hold an operating system.

Bad Block Description
The Bad Block Description explicitly specifies up to 889 bad blocks to be added to the bad block table. *Iv* merges specified bad block information with information already in the bad block table (if there already is one) and bad block information discovered through the

surface test.

Each bad block entry is a single line. There are two forms:

s

where *s* is a sector number;

c h b

where *c* is a cylinder number, *h* is a head number, and *b* is a byte number. Both forms condemn a single sector, the second the sector that contains the specified byte.

The last sector on each track serves as a bad block alternate. *Iv* chooses the alternates in a way that minimizes extra seeking for alternate blocks.

Partition Table Description

The Partition Table Description specifies where the slices (partitions) on the disk are to begin and end. Each line in the Description specifies the starting logical block of a slice. Start blocks must be on even boundaries. Except for overlapping partitions, slices must be listed in ascending numeric order, and the beginning of a slice defines the end of the previous slice.

If necessary, overlapping partitions can be specified. A $ following any block number indicates that the slice extends to the end of the disk, beyond the next boundary number. Any slice with a starting block number that is larger than its successor must extend to the end of the disk (and must therefore be followed by the $ parameter).

For example, the following description specifies five slices; the fifth slice extends from the second slice to the end of the disk:

```
0
16
20016
40016   $
16      $
```

The following example is also possible, although of doubtful utility:

```
0
16
20016   $
40016   $
16
30016   $
```

In this example six slices are specified. The third, fourth, and sixth slices extend to the end of the disk. The fifth slice, however, starts at 16 and ends at 30015 (inclusive); it includes all the second slice, but only part of the third slice.

The first logical block boundary number in the Description must always be 0. The last slice in the Description always extends to the end of the disk ($ is optional).

Note: **Do not use Partition Table Descriptions from other versions of CTIX that specify partitions by track numbers, rather than by logical block boundaries.**

There can be at most 16 slices on a disk.

It is a fatal error to specify a slice 1 that does not leave enough room in slice 0 for the Volume Home Block and the slice 0 files.

Down Load Area Description

The Down Load Area Description specifies system images to be included in the Down Load Area. Each line in the Description consists of a numeric device identification, the size of the system image in logical blocks, and the full path name of the file to be copied into the down load area; the three parts of the line are separated by one or more spaces or tabs.

EXAMPLES

Here is an example of a disk description file for a nonbootable disk.

```
#       MAXTOR 85 MB disk
type            HD
name            Serno
cylinders       1024
heads           8
sectors         17
steprate        14
hitech
ecc
$
$
0
8
$
$
```

The following file describes a bootable SMD.

```
type            V3200
name            Serno
cylinders       1489
heads           11
sectors         33
ecc
gap1            16
gap2            16
$
badblocktable   3
dump            1024
downloadarea    300
loader          /usr/lib/iv/loader
$
$
0
1360
17360
25360
45360
85360
125360
165360 $
$
100     /usr/lib/iv/ws100.422
200     /usr/lib/iv/ws200.422
$
```

The following file describes a bootable Hitachi drive.

```
type            HD
name            Serno
cylinders       823
heads           10
sectors         17
steprate        14
hitech
ecc
$
badblocktable   1
dump            1024
downloadarea    300
loader          /usr/lib/iv/loader
$
$
0
1346
17730
25922
```

```
46402
$
100     /usr/lib/iv/ws100.422
200     /usr/lib/iv/ws200.422
$
```

The following file describes a drive without a dump area.

```
type            HD
name            Serno
cylinders       645
heads           7
sectors         17
steprate        14
precomp         80
hitech
ecc
$
badblocktable   1
downloadarea    300
loader          /usr/lib/iv/loader
$
$
0
328
12328
18328   $
$
100     /usr/lib/iv/ws100.422
200     /usr/lib/iv/ws200.422
$
```

FILES

/dev/rdsk/* – disk character special files
/usr/lib/iv/desc.* – prototype description files.

SEE ALSO

update(1), disk(7).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.
"WD2010-05 Winchester Disk Controller" in *Storage Management Products Handbook*. Irvine, Calif.: Western Digital Corp., 1984.

WARNINGS

The **–i**, **–u**, and **–s** operations are dangerous or fatal to existing volume data. Always precede these operations with a backup.

When a new bad block is itself an alternate block, *iv* may produce messages that appear spurious but are actually correct. If the bad block is already in use as an alternate, the "added bad block" message can appear

twice for one block.
Do not run *mkfs*(1M) on an overlapping partition.

NAME

issue – issue identification file

DESCRIPTION

The file **/etc/issue** contains the *issue* or project identification to be printed as a login prompt. This is an ASCII file which is read by program *getty* and then written to any terminal spawned or respawned from the **/etc/inittab** file.

FILES

/etc/issue

SEE ALSO

login(1).

NAME

ldfcn – common object file access routines

SYNOPSIS

```
#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>
```

DESCRIPTION

The common object file access routines are a collection of functions for reading an object file that is in common object file form. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type LDFILE, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function *ldopen*(3X) allocates and initializes the LDFILE structure and returns a pointer to the structure to the calling program. The fields of the LDFILE structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

LDFILE *ldptr;

TYPE(ldptr)	The file magic number, used to distinguish between archive members and simple object files.
OPTR(ldptr)	The file pointer returned by *fopen* and used by the standard input/output functions.
OFFSET(ldptr)	The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.
HEADER(ldptr)	The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

(1) functions that open or close an object file

NAME

join – relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is –, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

–a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

–e *s* Replace empty output fields by string *s*.

–j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.

–o *list* Each output line comprises the fields specifed in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.

–t*c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been

sorted in ASCII collating sequence on the group ID fields.

```
join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd
/etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

With default field separation, the collating sequence is that of **sort -b**; with **-t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk*(1) are wildly incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

NAME

kill – terminate a process

SYNOPSIS

kill [–signo] PID ...

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 numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless 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

killall – kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

Killall is a procedure used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

Killall is chiefly used to terminate all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* (see *kill*(1)) to all remaining processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), kill(1), ps(1), shutdown(1M), signal(2).

NAME

ld – link editor for common object files

SYNOPSIS

ld [**–e** epsym] [**–f** fill] [**–l**x] [**–m**] [**–r**] [**–s**] [**–t**] [**–M**] [**–o** outfile] [**–u** symname] [**–L** dir] [**–x**] [**–Z**] [**–N**] [**–n**] [**–F**] [**–v**] [**–VS** num] [**–G**] [**–w**] filenames

DESCRIPTION

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *ld* combines them, producing an object module that can either be executed or used as input for a subsequent *ld* run. The output of *ld* is left in **a.out**. This file is executable if no errors occurred during the load. If any input file, *filename*, is not an object file, *ld* assumes it is either an ASCII file containing link editor directives or an archive library.

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. The library (archive) symbol table (see *ar*(4)) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is unimportant.

The following options are recognized by *ld*.

–e epsym
: Set the default entry point address for the output file to be that of the symbol *epsym*.

–f fill
: Set the default fill pattern for "holes" within an output section as well as initialized *bss* sections. The argument *fill* is a two-byte constant.

–l*x*
: Search a library named **lib***x***.a,** where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **–l** is significant. By default, libraries are located in

/lib

and

/usr/lib.

However, if the shell variable LIBROOT is set, the value of LIBROOT is prepended to **/lib** and **/usr/lib** before searching the libraries.

-m Produce a map or listing of the input/output sections on the standard output.

-o outfile
Produce an output object file by the name *outfile*. The name of the default object file is **a.out**.

-r Retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor will not complain about unresolved references, and the output file will not be executed.

-s Strip line number entries and symbol table information from the output object file.

-t Turn off the warning about multiply-defined symbols that are not the same size.

-u symname
Enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-x Do not preserve local (non-.globl) symbols in the output symbol table; enter only external and static symbols. This option saves some space in the output file.

-Z Do not bind anything to address zero. This option will allow runtime detection of null pointers.

-L dir Change the algorithm of searching for **lib***x***.a** to look in *dir* before looking in /lib.

-M Output a message for each multiply-defined external definition. However, if the objects being loaded include debugging information, extraneous output is produced (see the **-g** option in *cc*(1)).

-N Put the data section immediately following the text in the output file. The result is a plain executable file, indicated by magic number 0407 in the operating system header.

-n Put the data section at the next segment boundary following the text section. The result is a shared text file, indicated by magic number 0410 in the operating system header.

-z Like **-n** but permits demand paged execution. This type of file is indicated by magic number 0413 in the operating system header.

-F Like **-z** but takes less disk space and can page faster into memory. This type is also indicated by magic number 0413 in the operating system header. It is distinguished by having virtual text and data starting addresses that are equal to the file offsets of the text and data sections, modulo 4096. The **-F** option is on by default.

-V Output a message giving information about the version of ld being used.

-VS num
Use **num** as a decimal version number identifying the **a.out** file that is produced. The version stamp is stored in the optional header.

-G Change the symbol name look-up algorithm as follows: if two names do not initially match, then if one of them is exactly eight characters, then a match is attempted only on the first eight characters. The purpose of this is to allow compatibility between object modules that have been created with and the old C compiler and with the new C compiler, which allows variable names more than eight characters long. A warning message is issued in such cases.

-w If **-G** is used, do not print warnings about symbols that partially matched.

FILES

/lib/lib*x*.a	libraries
/usr/lib/lib*x*/*fR.a*	libraries
a.out	output file
/lib/ifile.0407	default **-N** directive file
/lib/ifile.0410	default **-n** directive file
/lib/ifile.0413	default **-z** directive file
/lib/ifile.0413-F	default **-F** directive file

SEE ALSO
as(1),cc(1), exit(2), end(3C), a.out(4),ar(4).

CAVEATS
Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should insure the following properties for programs:

- C defines a zero pointer as null. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the data space.

NAME

/etc/lddrv/lddrv – manage loadable drivers

SYNOPSIS

lddrv [**–m** master] [**–abdqsuv**] [devname] [subdevs]]
lddrv –a [**v**] [**–m** master] [**–o** dfile] devname [subdevs]
lddrv –d [**v**] [**–m** master] devname
lddrv –b [**v**] [**–m** master] devname
lddrv –u [**v**] [**–m** master] devname
lddrv –q [**v**] [**–m** master] devname
lddrv –s [**v**] [**–m** master]

DESCRIPTION

Lddrv allocates/deallocates space for a specified driver, loads/unloads a specified driver, and returns the status of specified driver(s).

The **v** argument prints verbose information on the screen. The **–m** option allows overriding the default master file (**/etc/master**). Use **–o** *dfile* to specify the name of the file that contains the driver's executable code; if omitted, executable code is placed in the file *devname*. The *devname* argument is the name of the driver.

Devname must correspond to the first field in the *master* file. In addition, the relocatable driver code must be in a file named *devname*.**o**. More than one major device may be plugged with a single invocation of *lldrv* by specifying up to three subdevices.

The options are:

–a Allocate space for and load the driver.

–d Unload the driver and deallocate its space.

–b Load (bind) the driver.

–u Unload the driver.

–q Return the status of a particular loadable driver.

–s Return the status of all loadable drivers.

EXAMPLES

A status report for all drivers could look like:

```
DEVNAME  ID  BLK  CHAR    SIZE     ADDR FLAGS
   lipc   0   -     -   0x5000 0x3dd000 ALLOC BOUND
   plp    1   -     6   0x1000 0x3e2000 ALLOC BOUND
```

FILES

/etc/master	default master file
/etc/drvtbl	loadable driver table
/etc/lddrv	contains *lddrv* and loadable drivers

SEE ALSO

syslocal(2), master(4), drivers(7).

NAME

ldeeprom – load EEPROM

SYNOPSIS

ldeeprom [**–s** <system file>]

DESCRIPTION

Ldeeprom is used to load the electrically erasable programmable read-only memory on the VME interface card.

Ldeeprom reads the **/etc/system** file, generates a VME description file, and outputs it to the EEPROM.

The **–s** option can be used to specify a file to be used instead of the **/etc/system** file.

SEE ALSO

system(4), vme(7).

NAME

lex – generate programs for simple lexical tasks

SYNOPSIS

lex [**–rctvn**] [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 *n* number of positions is *n* (default 2000)

%n *n* number of states is *n* (500)

%t *n* number of parse tree nodes is *n* (1000)

%a *n* number of transitions is *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).
CTIX Programmer's Guide, Section 17.

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 [option] ... file ...

DESCRIPTION

Lint attempts to detect features of the C program file that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **–c** or the **–o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

Lint will take all the **.c**, **.ln**, and **llib-l***x***.ln** (specified by **–l***x*) files and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is not used, the second pass of *lint* checks this list of files for mutual copatibility. When the **–c** option is used, the **.ln** and the the **llib-l***x***.ln** files are ignored.

Any number of *lint* options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

–a Suppress complaints about assignments of long values to variables that are not long.

–b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints.)

–h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

-u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program.)

-v Suppress complaints about unused arguments in functions.

-x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

-l*x* Include additional lint library **llib-l***x***.ln**. For example, you can include a lint version of the math library **llib-lm.ln** by inserting **-lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These line libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

-n Do not check compatibility against either the standard or the portable lint library.

-p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

-c Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for inter-function compatibility.

-o lib Cause *lint* to create a lint library with the name **llib-l***lib***.ln**. The **-c** option nullifies any use of the **-o** option. The lint library produced is the input that is given to *lint*'s second pass. The **-o** option simply causes this file to be saved in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **-x** option is suggested. The **-v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint commants" (see below).

The **-D**, **-U**, and **-I** options of *cpp*(1) and the **-g** and **-O** options of *cc*(1) are also recognized as separate

arguments. The **-g** and **-O** options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc*(1) command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code.

/*VARARGS*n*/
: suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
: turns on the **-v** option for the next function.

/*LINTLIBRARY*/
: at the beginning of a file shuts off complaints about unused functions 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, if the **-c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **-c** and the **-o** options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the **-c** option. Each of these invocations produces a **.ln** file which corresponds to the **.c** file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the **-c** option), listing all the **.ln** files with the needed **-l***x* options. This will print all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the

source files that have been modified since the last time the set of source files were *lint*ed.

FILES

/usr/lib	the directory where the lint libraries specified by the **−lx** option must exist
/usr/lib/lint[12]	first and second passes
/usr/lib/llib-lc.ln	declarations for C Library functions (binary format; source is in **/usr/lib/llib-lc**)
/usr/lib/llib-port.ln	declarations for portable functions (binary format; source is in **/usr/lib/llib-port**)
/usr/lib/llib-lm.ln	declarations for Math Library functions (binary format; source is in **/usr/lib/llib-lm**)
/usr/tmp/*lint*	temporaries

SEE ALSO

cc(1), cpp(1), make(1).

BUGS

exit(2) *longjmp*(3C), and other functions that do not return are not understood; this causes various lies.

need not be allocated to the file in order to be locked, as such locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent sections occur, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

F_LOCK and F_TLOCK requests differ only by the action taken if the resource is not available. F_LOCK will cause the calling process to sleep until the resource is available. F_TLOCK will cause the function to return a −1 and set *errno* to [EACCESS] error if the section is already locked by another process.

F_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an [EDEADLK] error is returned and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by accessing another process's locked resource. Thus calls to *lock* or *fcntl* scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The *alarm*(2) command may be used to provide a timeout facility in applications which require this facility.

ERRORS

The *lockf* utility will fail if one or more of the following are true:

[EBADF]	*Fildes* is not a valid open descriptor.
[EACCESS]	*Cmd* is F_TLOCK or F_TEST and the section is already locked by another process.
[EDEADLK]	*Cmd* is F_LOCK or F_TLOCK and a deadlock would occur. Also the *cmd* is either of the above or F_ULOCK and

the number of entries in the lock table would exceed the number allocated on the system. (Note that this differs from EDEADLOCK.)

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.

CAVEATS

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package is the most common source of unexpected buffering.

SEE ALSO

close(2), creat(2), fcntl(2), intro(2), open(2), read(2), write(2).

NAME

login – sign on

SYNOPSIS

login [name [env-var . . .]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file." (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure **/etc/profile** is performed (or **/etc/cprofile** for *csh*), the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and the file **.profile** (or **.cshrc** and **.login** for *csh*) in the working directory is executed, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is – followed by the last component of the interpreter's pathname (i.e., **–sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used. If this field is "*", then a *chroot*(2) is done to the directory named in the directory field of the entry.

At that point *login* is re-executed at the new level which must have its own root structure, including **/etc/login** and **/etc/passwd**.

The basic *environment* (see *environ*(5)) is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

L*n*=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/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
.profile	user's login profile

SEE ALSO

mail(1), newgrp(1), sh(1), su(1), passwd(4), profile(4), environ(5).

DIAGNOSTICS

Login incorrect if the user name or the password cannot be matched.
No shell, cannot open password file, or *no directory*: consult your system administrator.
No utmp entry. You must exec "login" from the lowest level "sh". if you attempted to execute *login* as a

command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname – 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(5).

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1). Note that the link editor *ld*(1) is capable of multiple passes over an archive in the portable archive format (see *ar*(4)) and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar cr library `lorder *.o | tsort`
```

FILES

*symref, *symdef temporary files

SEE ALSO

ar(1), ld(1), tsort(1), ar(4).

BUGS

Object files whose names do not end with **.o**, even when contained in library archives, are overlooked. Their global symbols and references are attributed to some other file.

NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

lp [**–c**] [**–d**dest] [**–m**] [**–n**number] [**–o**option] [**–s**] [**–t**title] [**–w**] files
cancel [ids] [printers]

DESCRIPTION

Lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name – stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

Lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

–c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the **–c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **–c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

–d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable LPDEST (if it is set). Otherwise, a default

destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

-m Send mail (see *mail(1)*) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

-n*number* Print *number* copies (default of 1) of the output.

-o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **-o** keyletter more than once. For more information about what is valid for *options*, see *Models* in *lpadmin*(1M).

-s Suppress messages from *lp*(1) such as "request id is ...".

-t*title* Print *title* on the banner page of the output.

-w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lpstat(1), lpadmin(1M), lpsched(1M), mail(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

lpadmin – configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin **–p** *printer* [*options*]
/usr/lib/lpadmin **–x** dest
/usr/lib/lpadmin **–d**[dest]

DESCRIPTION

Lpadmin configures LP spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination. *Lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **–p**, **–d** or **–x** options must be present for every legal invocation of *lpadmin*.

–d[*dest*] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **–d**.

–x*dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **–x**.

–p*printer* names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.

The following *options* are only useful with **–p** and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

–c*class* inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.

–e*printer* copies an existing *printer's* interface program to be the new interface program for *P*.

–h indicates that the device associated with *P* is hardwired. This *option* is assumed when creating a new printer unless the **–l** *option* is supplied.

−i*interface* establishes a new interface program for *P*. *Interface* is the pathname of the new program.

−l indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*(1M), disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

−m*model* selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP software (see *Models* below).

−r*class* removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.

−v*device* associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by the LP administrator, *lp*. Note that there is nothing to stop an administrator from associating the same *device* with more than one *printer*. If only the **−p** and **−v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the **−v** option and one of the **−e**, **−i** or **−m** options must be supplied. Only one of the **−e**, **−i** or **−m** options may be supplied. The **−h** and **−l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and _ (underscore).

Models.

Model printer interface programs are supplied with the LP software. They are shell procedures which interface between *lpsched*(1M), and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **−m**. Models should have 644 permission if owned by lp and bin, or 664 permission if owned by bin and bin. Alternatively, LP administrators may modify copies of models and then use *lpadmin* **−i** to associate them with printers. The following list describes the *models* and lists the options which they may be given on the *lp* command line using the **−o** keyletter:

dumb interface for a line printer without special functions and protocol. Form feeds are assumed. This is a good model to copy and modify for printers which do not have models.

1640 DIABLO 1640 terminal running at 1200 baud, using XON/XOFF protocol. Options:

−12 12-pitch (10-pitch is the default)
−f do not use the *450*(1) filter. The output has been pre-processed by either *450*(1) or the *nroff*(1) 450 driving table.

hp Hewlett-Packard 2631A line printer at 2400 baud. Options:

−c compressed print
−e expanded print

prx Printronix P300 or P600 printer using XON/XOFF protocol at 1200 baud.

EXAMPLES

1. Assuming there is an existing Hewlett-Packard 2631A line printer named *hp2*, it will use the **hp** model interface after the command:

```
/usr/lib/lpadmin −php2 −mhp
```

2. To obtain compressed print on *hp2*, use the command:

```
lp −dhp2 −o−c files
```

3. A DIABLO 1640 printer called *st1* can be added to the LP configuration with the command:

```
/usr/lib/lpadmin −pst1 −v/dev/tty20 −m1640
```

4. An *nroff*(1) document may be printed on *lp* in any of the following ways:

```
nroff −T450 files | lp −dst1 −of
nroff −T450−12 files | lp −dst1 −of
nroff −T37 files | col | lp −dst1
```

5. The following command prints the password file on *st1* in 12-pitch:

```
lp −dst1 −o12 /etc/passwd
```

NOTE: the **−12** option to the **1640** model should never be used in conjunction with *nroff*(1).

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1), nroff(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

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.

Lpr is a simple alternative to the *lp*(1) system. The same system should not use both.

Lpr uses a CTIX demon to manage spooling.

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.

FILES

/etc/passwd	user's identification and accounting data
/usr/lib/lpd	line printer daemon
/usr/spool/lpd/*	spool area
/etc/rc	initialization for *lp* or *lpr* spooling system

SEE ALSO

MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

lpsched, lpshut, lpmove – start/stop the LP request scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

Lpsched schedules requests taken by *lp*(1) for printing on line printers.

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp*(1) will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1M)) for the new destination when moving requests.

FILES

/usr/spool/lp/*

/etc/rc invocation of *lpsched*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

lpset – set parallel line printer options

SYNOPSIS

lpset [**–i** indent] [**–c** columns] [**–l** lines]

DESCRIPTION

Lpset sets the translation options for the parallel printer interface. It is normally run in */etc/rc*. The options set the indent (**–i**), number of columns (**–c**), and lines per page (**–l**); the interpretation of these options by the interface is described under *lp*(7).

With no options, *lpset* reports the current values. Initially, the values are: an indent of 4, 132 columns, 66 lines per page.

FILES

/dev/lp

SEE ALSO

lp(7).

DIAGNOSTICS

Self explanatory, except that "parallel printer not properly connected" may actually mean that the printer is in use.

NAME

lpstat – print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

Lpstat prints information about the current status of the LP line printer system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *Lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
-u"user1, user2, user3"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat -o
```

prints the status of all output requests.

-a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

-c[*list*] Print class names and their members. *List* is a list of class names.

-d Print the system default destination for *lp*.

-o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

-p[*list*] Print the status of printers. *List* is a list of printer names.

-r Print the status of the LP request scheduler

-s Print a status summary, including the status of the line printer scheduler, the system default destination, a list of class names and their members, and a list of printers and their associated devices.

-t Print all status information.

-u[*list*] Print status of output requests for users. *List* is a list of login names.

-v[*list*] Print the names of printers and the pathnames of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls – list contents of directory

SYNOPSIS

ls [**–RadCxmlnogrtucpFbqisf**] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are four listing formats:

- Multicolumn format. This is the default when the standard output is a terminal. By default this format sorts names down the page; the **–x** option controls this. Choice of multicolumn format is controlled manually by the **–C** option.
- Simple (one entry per line) format. This is the default when the standard output is not a terminal. Each line consists of a file name together with whatever additional information is requested by options.
- Long format. See the the **–l** option.
- Stream format. See the **–m** option.

The number of columns used in multicolumn and stream format is taken from an environment variable, COLUMNS. If this variable is not set, the *terminfo* database is used to determine the number of columns, based on the environment variable TERM. If this information cannot be obtained, 80 columns are used.

There are an unbelievable number of options:

–R Recursively list subdirectories encountered.

–a List all entries; usually entries whose names begin with a period (.) are not listed.

–d If an argument is a directory, list only its name (not its contents); often used with **–l** to get the status of a directory.

–C Multi-column output with entries sorted down the columns.

-x Multi-column output with entries sorted across rather than down the page.

-m Stream output format.

-l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.

-n The same as **-l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

-o The same as **-l**, except that the group is not printed.

-g The same as **-l**, except that the owner is not printed.

-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

-t Sort by time modified (latest first) instead of by name.

-u Use time of last access instead of last modification for sorting (with the **-t** option) or printing (with the **-l** option).

-c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**-t**) or printing (**-l**).

-p Put a slash (/) after each filename if that file is a directory.

-F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.

-b Force printing of non-graphic characters to be in the octal \ddd notation.

-q Force printing of non-graphic characters in file names as the character (?).

-i For each file, print the i-number in the first column of the report.

-s Give size in 512-byte blocks, including indirect blocks, for each entry.

-f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **-l, -t, -s,** and **-r,** and

turns on **−a;** the order is the order in which entries appear in the directory.

The mode printed under the **−l** option consists of 10 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 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	to get user IDs for **ls −l** and **ls −o**.
/etc/group	to get group IDs for **ls −l** and **ls −g**.

/usr/lib/terminfo/*
to get terminal information.

SEE ALSO
chmod(1), find(1).

BUGS
Unprintable characters in file names may confuse the columnar output options.

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.

–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 the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is

the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define — the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **$*n*** in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is like $*, but each argument is quoted (with the current quotes).

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 CTIX system 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 new-line. 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.

decr	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 CTIX system 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( )')

errprint — 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

cc(1), cpp(1).
CTIX Programmer's Guide, Section 10.

NAME

mc68k, pdp11, u3b, u3b5, vax – provide truth value about your processor type

SYNOPSIS

mc68k

pdp11

u3b

u3b5

vax

DESCRIPTION

These commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

The commands that do not apply will return a false (non-zero) value. These commands are often used within *make*(1) makefiles and shell procedures to increase portability.

SEE ALSO

sh(1), test(1), true(1), false(1), make(1).

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

mail [**–epqr**] [**–f** file]

mail [**–t**] persons

rmail [**–t**] 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:

–e causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

–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.

–r causes messages to be printed in first-in, first-out order.

–f*file* 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 >. The **–t** option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of **dead.letter**.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). 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 **a!b!cde** as a recipient's name causes the message to be sent to user **b!cde** on system **a**. System **a** will interpret that destination as a request to send the message to user **cde** on system **b**. This might be useful, for instance, if the sending system can access system **a** but not system **b**, and system **a** has access to system **b**. *Mail* will not use *uucp* if the remote system is the local system name (i.e., localsystem!user).

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment. In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

Rmail only permits the sending of mail; ***uucp***(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the *mailfile*
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), uucp(1C), write(1).

BUGS

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

mailx – interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Incoming mail is stored in a standard file for each user, called the system *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages remain in this file until forcibly removed.

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

–d	Turn on debugging output. Neither particularly interesting nor recommended.
–e	Test for presence of mail. *Mailx* prints nothing and exits with a successful return code if there is mail to read.
–f *[filename]*	Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.
–F	Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
–h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.
–H	Print header summary only.

-i	Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
-n	Do not initialize from the system default *Mailx.rc* file.
-N	Do not print initial header summary.
-r *address*	Pass *address* to network delivery software. All tilde commands are disabled.
-s *subject*	Set the Subject header field to *subject*.
-u *user*	Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.
-U	Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable.

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **set** and **unset** commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the **alias** command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a 'current' message, marked by a '>' in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces, which may include:

n	Message number **n**.
.	The current message.
^	The first undeleted message.
$	The last message.
*	All messages.
n–m	An inclusive range of message numbers.
user	All messages from **user**.
/string	All messages with **string** in the subject line (case ignored).
:c	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **Copy**, **edit**, **followup**, **Followup**, **hold**, **mail**, **preserve**, **reply**, **Reply**, **shell**, and **visual**. Any errors in the start-up file cause the remaining lines in the file to be ignored.

COMMANDS

The following is a complete list of *mailx* commands:

!*shell-command*

Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

comment

Null command (comment). This may be useful in *.mailrc* files.

=

Print the current message number.

?

Prints a summary of commands.

alias *alias name ...*
group *alias name ...*

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates *name ...*

Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **al**ternates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]
chdir [*directory*]

Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*

Copy messages to the file without marking the messages as saved. Otherwise equivalent to the **s**ave command.

Copy [*msglist*]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the **S**ave command.

delete [*msglist*]
Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [*header-field* ...]
ignore [*header-field* ...]
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **Print** and **Type** commands override this command.

dp [*msglist*]
dt [*msglist*]
Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a **delete** command followed by a **print** command.

echo *string* ...
Echo the given strings (like *echo*(1)).

edit [*msglist*]
Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed*(1).

exit
xit
Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

file [*filename*]
folder [*filename*]
Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:
% the current *mailbox*.
%**user**
the *mailbox* for **user**.
the previous file.

& the current *mbox*.
Default file is the current *mailbox*.

folders
Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]
Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **F**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]
Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **f**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]
Prints the header summary for the specified messages.

group *alias name* ...
alias *alias name* ...
Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]
Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the **z** command.

help
Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
Holds the specified messages in the *mailbox*.

if *s*|*r*
mail-commands
else
mail-commands
endif
Conditional execution, where *s* will execute following *mail-commands*, up to an **else** or **endif**, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore *header-field* ...
discard *header-field* ...
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **Print** and **Type** commands override this command.

list
Prints all commands available. No explanation is given.

mail *name* ...
Mail a message to the specified users.

mbox [*msglist*]
Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **exit** and **quit** commands.

next [*message*]
Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]
Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]
Preserve the specified messages in the *mailbox*.

Print [*msglist*]
Type [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

print [*msglist*]
type [*msglist*]
Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit
Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a filename, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]
respond [*message*]
Reply to the specified message, including all other recipients of the message. If "record" is set to a filename, the response is saved at the

end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]
Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the **C**opy, **f**ollowup, and **F**ollowup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]
save [*msglist*] *filename*
Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the **e**xit and **q**uit commands).

set
set *name*
set *name=string*
set *name=number*
Define a variable called *name*. The variable may be given a null, string, or numeric value. **S**et by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell
Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [*msglist*]
Print the size in characters of the specified messages.

source *filename*
Read commands from the given file and return to command mode.

top [*msglist*]
Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*]
Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See **exit** and **quit**.

Type [*msglist*]
Print [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

type [*msglist*]
print [*msglist*]
Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

undelete [*msglist*]
Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name ...*
Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version
Prints the current version and release date.

visual [*msglist*]
Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*
Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the **save** command.

xit
exit

Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **q**uit).

z[+|-]

Scroll the header display forward or backward one screen-full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~! *shell-command*

Escape to the shell.

~.

Simulate end of file (terminate message input).

~: *mail-command*

~_ *mail-command*

Perform the command-level request. Valid only when sending a message while reading mail.

~?

Print a summary of tilde escapes.

~A

Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~a

Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~b *name* ...

Add the *names* to the blind carbon copy (Bcc) list.

~c *name* ...

Add the *names* to the carbon copy (Cc) list.

~**d**

Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~**e**

Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~**f** [*msglist*]

Forward the specified messages. The messages are inserted into the message, without alteration.

~**h**

Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~**i** *string*

Insert the value of the named variable into the text of the message. For example, ~**A** is equivalent to '~**i** Sign.'

~**m** [*msglist*]

Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~**p**

Print the message being entered.

~**q**

Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~**r** *filename*
~< *filename*
~< !*shell-command*

Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell

command and is executed, with the standard output inserted into the message.

~s *string* ...
Set the subject line to *string*.

~t *name* ...
Add the given *names* to the To list.

~v
Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w *filename*
Write the partial message onto the given file, without the header.

~x
Exit as with ~q except the message is not saved in *dead.letter*.

~| *shell-command*
Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*
The user's base of operations.

MAILRC=*filename*
The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **unset** command may be used to erase variables.

allnet
All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also

the **alt**ernates command and the "metoo" variable.

append
Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**

askcc
Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub
Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint
Enable automatic printing of messages after **d**elete and **u**ndelete commands. Default is **noautoprint**.

bang
Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang**.

cmd=*shell-command*
Set the default command for the **p**ipe command. No default value.

conv=*conversion*
Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **-U** command line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (*pg*(1) by default). Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is $HOME/dead.letter.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot
Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*
The command to run when the **edit** or **~e** command is used. Default is *ed*(1).

escape=*c*
Substitute *c* for the ~ escape character.

folder=*directory*
The directory for saving standard mail files. User specified file names beginning with a plus (+) are expanded by preceding the filename with this directory name to obtain the real filename. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header
Enable printing of the header summary when entering *mailx*. Enabled by default.

hold
Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore
Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof
Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the ~. command. Default is **noignoreeof**. See also "dot" above.

keep
When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave
Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*
The name of the file to save messages which have been read. The **xit** command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo
If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*
The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop
When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the pathname is absolute. Default is **nooutfolder**. See "folder" above and the **S**ave, **C**opy, **f**ollowup, and **F**ollowup commands.

page
Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*
Set the *command mode* prompt to *string*. Default is "? ".

quiet
Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save
Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*
Sets the number of lines in a screen-full of headers for the **headers** command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is *mail*(1).

sendwait
Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*
The name of a preferred command interpreter. Default is *sh*(1).

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
The variable inserted into the text of a message when the ~**a** (autograph) command is given. No default (see also ~**i** (TILDE ESCAPES)).

Sign=*string*
The variable inserted into the text of a message when the ~**A** command is given. No default (see also ~**i** (TILDE ESCAPES)).

toplines=*number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. Default is *vi*(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO
mail(1), pg(1), ls(1).

BUGS
Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail*(1) (the standard mail delivery program).

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [**–f** makefile] [**–p**] [**–i**] [**–k**] [**–s**] [**–r**] [**–n**] [**–b**] [**–e**] [**–m**] [**–t**] [**–d**] [**–q**] [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 - *makefile* argument pair may appear.

Make updates a target only if its dependents 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. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common

file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The first one or two characters in a command can be the following: **-**, **@**, **-@**, or **@-**. If **@** is present, printing of the command is suppressed. If **-** is present, *make* ignores an error. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a **@**. The **-n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion 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 the initial character sequence of the command contains **-**. the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old makefiles (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null or implicit) command associated with them. The previous version of *make* assumed, if no command was specified explicitly, that the command was null.

Interrupt and quit cause the target to be deleted unless the target is a dependent of 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. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros 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, **./** is generated. The only macro excluded from this alternative form is **$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules 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:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o
.y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

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:

```
make -fp - 2>/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*(4)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x***.c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .y .l .s

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library.

Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the LIB macro has been previously defined.) The expression **$(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 *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        ar rv $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        ar rv lib $?
        rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile and s.[Mm]akefile

SEE ALSO

cc(1), cd(1), lex(1), sh(1), yacc(1), printf(3S), sccsfile(4).

BUGS

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty. File names with the

characters = : @ will not work. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*. The syntax **(lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c~)** does not work.

NAME

man, manprog – print entries in this manual

SYNOPSIS

man [options] [section] titles

/usr/lib/manprog file

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 ($8.5'' \times 11''$).

–s Typeset the entry in the small format ($6'' \times 9''$).

–T4014 Display the typeset output on a Tektronix 4014 terminal using *tc*(1).

–Ttek Same as **–T4014**.

–Tvp Print the typeset output on a Versatec printer; this option is not available at all sites.

–T*term* Format the entry using *nroff* and print it on the standard output (usually, the terminal); *term* is the terminal type (see *term*(5) 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 $TERM (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 **12** 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* other than **−d**, **−c**, and **−y** 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*, *nroff*, or the *man*(5) macro package.

When using *nroff*, *man* examines the environment variable $TERM (see *environ*(5)) and attempts to select options to *nroff*, as well as filters, that adapt the output to the terminal being used. The **−T***term* option overrides the value of $TERM; in particular, one should use **−Tlp** 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*(5).

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) (*neqn* for *nroff*) and *tbl*(1), respectively; if *eqn* or *neqn* are invoked, they will automatically read the file **/usr/pub/eqnchar** (see *eqnchar*(5)).

The *man* command executes *manprog* that takes a file name as its argument. *Manprog* calculates and returns a string of three register definitions used by the formatters identifying the date the file was last modified. The returned string has the form:

−rd*day* **−rm***month* **−ry***year*

and is passed to *nroff* which sets this string as variables for the *man* macro package. Months are given from 0 to 11, therefore month is always 1 less than the actual month. The *man* macros calculate the correct month. If the *man* macro package is invoked as an option to *nroff*/*troff* (i.e., *nroff −man file*), then the current day/month/year is used as the printed date.

FILES

/usr/man/u_man/man[1-8]	directories for source files
/usr/man/local/man[1-8]/*	local additions
/usr/lib/manprog	calculates modification dates of entries

SEE ALSO

cw(1), eqn(1), nroff(1), tbl(1), tc(1), troff(1), environ(5), man(5), term(5).

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, mkdirs – make a directory

SYNOPSIS

mkdir dirname ...
mkdirs dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask*(1)). Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

Mkdirs creates the specified directory by using *mkdir*, as well as all nonexistent parent directories. All diagnostics are those of *mkdir*. If no directories are made, *mkdir* is silent and has an exit status of 0.

SEE ALSO

sh(1), rm(1), umask(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special [**–O**] blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs special [**–O**] proto [gap blocks/cyl]
/etc/mkfs special

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. The command waits 10 seconds before starting to construct the file system. 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. This is the number of *physical* (512-byte) disk blocks the file system will occupy. The boot program is left uninitialized. If the optional number of i-nodes is not given, the default is the number of *logical* blocks divided by 4 (rounded down); i-nodes are allocated in groups of 16.

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. The second token is a number specifying the size of the created file system in *physical* disk blocks. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the file system. The maximum number of i-nodes configurable is 65500. 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 **–bcd** 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 IDs of the owner of the file.

If the file is a regular file, the next token is a pathname whence the contents 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 numbers. If the file is a directory, ***mkfs*** makes the entries . and .. and then reads a list of names and (recursively) files 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 *gap* and the number of *blocks/cyl* can be specified. The default gap size is 7. The default blocks/cylinder is 400. The default will be used if the supplied *gap* and *blocks/cyl* are considered illegal values or if a short argument count occurs.

The **-O** option makes a file system with a free list instead of a bit map.

Special must be a disk slice. The third form of the *mkfs* command extracts the slice size from the volume home block and creates a file system the same size; this third option cannot be used where there are overlapping partitions. The number of i-nodes is the number of logical blocks divided by 4. Optimal values for gap size and blocks/cylinder are calculated; these may not be 7 and 400.

SEE ALSO

chmod(1), dir(4), fs(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

BUGS

If a prototype is used, there is no way to specify links or cluster size.

NAME

mkhosts – make node name commands

SYNOPSIS

/etc/mkhosts

DESCRIPTION

Mkhosts makes the simplified forms of the *rcmd*(1N) and *rlogin*(1N) commands. For each node listed in **/etc/hosts**, *mkhosts* creates a link to **/usr/local/bin/rcmd** in **/usr/hosts**. Each link's name is the same as the node's official name in **/etc/hosts**.

SEE ALSO

rcmd (1N), rlogin(1N).

NAME

mkifile – make an ifile from an object file

SYNOPSIS

mkifile a.out ifile

DESCRIPTION

Mkifile takes an object module and writes in *ifile* a line of the form

symbol = 0x*value*

For each external symbol in the object module this ifile can be used as an argument to *ld*(1) as an absolute symbol table against which other modules may be linked. *Mkifile* is used with loadable drivers to provide the symbols for the currently running CTIX.

SEE ALSO

ld(1), lddrv(1M), ldeeprom(1M).

NAME

mklost+found – make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

A directory *lost+found* is created in the current directory and a number of empty files are created therein and then removed so that there will be empty slots for *fsck*(M). This command should be run immediately after first mounting a newly created file system.

SEE ALSO

fsck(1M), mkfs(1M)

BUGS

Should be done automatically by ***mkfs***.

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).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

mktpy, mvtpy – install or relocate a PT or GT local printer

SYNOPSIS

/usr/local/bin/mktpy [name1 [name2 [tty]]]
/usr/local/bin/mvtpy name tty n

DESCRIPTION

To install a PT or GT local printer initially, the system administrator must use the *lpadmin*(1) command. The administrator *must* use the **–l** flag to ensure that the spooling system knows that the printer is attached to a login terminal. If this is not done, the printers will be attached to random devices at boot time, and behavior is undefined.

Mktpy is used to inform the *lp*(1) spooling system of the location of a printer that is attached to a PT or GT and to *enable*(1) the printer to accept print requests. *Mktpy* installs a printer on an RS-422 terminal's local port. *Mvtpy* updates a *mktpy* installation by changing the device association when the terminal's device number changes. The device number may change each time the system or the terminal is powered off and on.

The *mktpy* command accepts the names of one or two printers as arguments. If no arguments are given, it will prompt for the name of two printers. As an argument, the null string " must be used in the first argument position if only one printer exists and is attached to the second port. In response to a prompt, a <cr> indicates that no printer is attached to the indicated port.

If *mktpy* is run and the indicated printer is already attached to another tty device, the location of that printer is not changed and a warning message is printed.

If the command is being executed to inform the system of a printer attached to a terminal other than the terminal from which the command is being run, then the printers must be specified as arguments (with a null string, if necessary, as a place holder). The tty line to which the printer's host terminal is attached must be specified as the third argument.

The *mvtpy* command is most commonly used when a PT or GT has been powered off and then on again and the tty number of the PT or GT has changed. The system needs to be informed of the tty line that should receive the print requests. Before running *mvtpy*, the new tty number must be determined. *Name1* is the name of the printer, *name2* is the tty number of the new tty, and *n* is

the port on the PT or GT to which the printer is attached; *n* must be either 1 or 2. *Mvtpy* processes these arguments and issues an *lpadmin*(1) command to move the specified printer to the indicated tty line. Diagnostics are from *lpadmin*.

For the convenience of users, a **mktpy** login is provided. The *mktpy* login executes *mktpy* for the terminal where it is executed and prompts for printer names.

EXAMPLES

To install two printers on the same tty:

```
mktpy DIABLO EPSON
```

To install a printer on the second port of another tty:

```
mktpy " QUME tty256
```

To tell the system that the tty number of the terminal to which a printer is attached has changed:

```
mvtpy QUME tty260
```

NOTES

Only PTs or GTs that are on cluster lines can have printers attached to the serial ports. GTs have two serial ports and PTs have one serial port. Printers attached to PTs are considered to be attached to the first port. For each terminal **/dev/ttynnn** on the system, two other character special files exist, **/dev/tpa***nnn* and **/dev/tpb***nnn*. These devices refer to the first and second serial ports respectively on the matching terminal.

FILES

/dev/tty*nnn*	Name of tty
/dev/tpa	First serial port of /dev/tty nnn
/dev/tpb	Second serial port of /dev/tty*nnn*
/tmp/mkt[12]*	Temporary files to hold printer status for mktpy
/tmp/mvt*	Temporary file to hold printer status for mvtpy
/dev/tp/*	

SEE ALSO

accept(1), disable(1), enable(1), lp(1), lpadmin(1), lpsched(1), reject(1), tp(7).

DIAGNOSTICS

Generally self explanatory. "LP doesn't know *xxxx*" if the printer has not been created yet. Check spelling of printer name or see system administrator.

WARNINGS

The command

mktpy " " ttyxxx

has no effect; it does not prompt for input.

Printing to the second port of a PT is not recommended.

NAME

mm, osdd, checkmm – print/check documents formatted with the MM macros

SYNOPSIS

mm [options] [files]

osdd [options] [files]

checkmm [files]

DESCRIPTION

Mm can be used to type out documents using *nroff* and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn* (see *neqn*(1)) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff* and MM are generated, depending on the options selected.

Osdd is equivalent to the command **mm –mosd**. For more information about the OSDD adapter macro package, see *mosd*(5).

Options for *mm* are given below. Any other arguments or flags (e.g., **–rC3**) are passed to *nroff* or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

–T*term* Specifies the type of output terminal; for a list of recognized values for *term*, type **help term2**. If this option is *not* used, *mm* will use the value of the shell variable $TERM from the environment (see *profile*(4) and *environ*(5)) as the value of *term*, if $TERM is set; otherwise, *mm* will use **450** as the value of *term*. If several terminal types are specified, the last one takes precedence.

–12 Indicates that the document is to be produced in 12-pitch. May be used when $TERM is set to one of **300**, **300s**, **450**, and **1620**. (The pitch switch on the DASI 300 and 300s terminals must be manually set to **12** if this option is used.)

–c Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, **1620**, and **X**.

–e Causes *mm* to invoke *neqn*; also causes *neqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).

-t Causes *mm* to invoke *tbl*(1).
-E Invokes the **-e** option of *nroff*.
-y Causes *mm* to use the non-compacted version of the macros (see *mm*(5)).

As an example (assuming that the shell variable $TERM is set in the environment to **450**), the two command lines below are equivalent:

```
mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3
```

Mm reads the standard input when **-** is specified instead of any file names. (Mentioning other files together with **-** leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm -
```

Checkmm is a program for checking the contents of the named *files* for errors in the use of the Memorandum Macros, missing or unbalanced *neqn* delimiters, and **.EQ/.EN** pairs. Note: The user need not use the *checkeq* program (see *neqn*(1)). Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

HINTS

1. *Mm* invokes *nroff* with the **-h** flag. With this flag, *nroff* assumes that the terminal has tabs set every 8 character positions.
2. Use the **-o***list* option of *nroff* to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of the **-e**, **-t**, and **-** options, *together* with the **-o***list* option of *nroff* may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.
3. If you use the **-s** option of *nroff* (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The **-s** option of *nroff* does not work with the **-c** option of *mm*, or if *mm* automatically invokes *col*(1) (see **-c** option above).
4. If you lie to *mm* about the kind of terminal its output will be printed on, you'll get (often subtle) garbage; however, if you are redirecting output into a file, use the **-T37** option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

col(1), env(1), neqn(1), greek(1), nroff(1), tbl(1), profile(4), mm(5), term(5).

DIAGNOSTICS

mm "mm: no input file" if none of the arguments is a readable file and *mm* is not used as a filter.

checkmm "Cannot open *filename*" if file(s) is unreadable. The remaining output of the program is diagnostic of the source file.

NAME

mm – the MM macro package for formatting documents

SYNOPSIS

mm [options] [files]

nroff –mm [options] [files]

nroff –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 eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **–mm** option causes *nroff* 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 the package
/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), nroff(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

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*; *mmt* uses the MM macro package, while *mvt* uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing 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); also causes *eqn* to read the **/usr/pub/eqnchar** file (see *eqnchar*(5)).

–t Causes these commands to invoke *tbl*(1).

–Tvp Directs the output to a Versatec printer; this option is not available at all 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*(5)). 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 **–o***list* 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 **–o***list* 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(4), environ(5), mm(5), mv(5).

DIAGNOSTICS

"m[mv]t: no input file" if none of the arguments is a readable file and the command is not used as a filter.

NAME

more, page – text perusal

SYNOPSIS

more [**–cdflsu**] [*–n*] [*+linenumber*] [*+/pattern*] [name ...]

page [**–cdflsu**] [*–n*] [*+l*] [*+/pattern*] [name ...]

DESCRIPTION

More and *page* display text a screenful at a time. *Page* clears the screen before each screenful; otherwise *page* and *more* are identical. Henceforth we refer just to *more*.

When the screen is full, *more* prints the string "--More--". If input is a file, *more* indicates how much of the file has been read. To display the next screenful, type a space. To display a list of commands, type an "h".

More treats underlining and form feeds (^L) specially; otherwise it passes along its input unmodified. If your terminal has an underline mode or some other standout mode, *more* uses this mode to display underlined text. If a file begins with a form feed, *more* clears the screen before displaying the file. Subsequent form feeds cause *more* to pause.

If the standard output is not a terminal, *more* does not pause between screenfuls.

To make *more* pause in the middle of a screenful, type QUIT (normally code–\). Some text is lost when you do this, and you may terminate whatever program is piping to *more*.

These are the options.

–*n* Display *n* lines instead of a screenful.

–c Avoid scrolling. *More* begins each screenful at the top of the screen and erases each line as it is needed.

–d Prompts user with "Hit space to continue, Rubout to abort" at the end of each screenful.

–f Count file lines, rather than screen lines.

–l No special treatment for form feeds.

–s Suppress all but one of each sequence of blank lines. Useful with *nroff*.

–u No special handling of underlining.

+*linenumber*
Begin displaying the file at line number *l*.

+/*pattern*
Search for *pattern* and begin displaying the file two lines before it. *Pattern* is a regular expression; it follows the same rules as does a search in *ex*(1).

If the program is invoked as *page*, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and *k* – 1 rather than *k* – 2 lines are printed in each screenful, where *k* is the number of lines the terminal can display.

To supply options automatically, put the options in the **MORE** environment variable. Here's an example for the Bourne Shell:

MORE = '-s -d'

More looks at the **TERM** environment variable to find what kind of terminal you're using and at the **TERMCAP** environment variable to find how the terminal works. If **TERMCAP** is not set, *more* examines */etc/termcap*.

More resets terminal modes. This permits character (as opposed to line) commands and limits echoing of commands.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the *-c* mode of operation, the *csh* command *setenv MORE -c* or the *sh* command sequence *MORE='-c' ; export MORE* would cause all invocations of *more* , including invocations by programs such as *man* and *msgs* , to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i <space>
display *i* more lines, (or another screenful if no argument is given)

^D display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d same as ^D (code-D)

i z same as typing a space except that *i*, if present, becomes the new window size.

i s skip *i* lines and print a screenful of lines

i f skip *i* screenfuls and print a screenful of lines

q or Q Exit from *more*.

= Display the current line number.

v Start up the editor *vi* at the current line.

h Help command; give a description of all the *more* commands.

i /expr search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

i n search for the *i*-th occurrence of the last regular expression entered.

' (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i :n skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i :p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If

more is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q
exit from *more* (same as q or Q).

(dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally code-\). *More* will stop sending output, and will display the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

AUTHOR

Eric Schienbrood, minor revisions by John Foderaro and Geoffrey Peck

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

csh(1), man(1), script(1), sh(1), termcap(4), environ(5).

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

setmnt(1M), mount(2), mnttab(4).

DIAGNOSTICS

Mount issues a warning if the file system to be mounted is currently mounted 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 generally unwise to mount garbage file systems.

NAME

mvdir – move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

Mvdir moves directories within a file system. *Dirname* must be a directory; *name* must not exist unless it is a directory. *Name* may be a subset of *dirname*, provided the directory is not moved onto itself (**mvdir /x/y /x** is possible, but **/x/y** cannot be moved to **/x/y/z**).

Only super-user can use *mvdir*.

SEE ALSO

mkdir(1).

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/etc/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. 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

bcheck(1M), 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

netman – form-based network management

SYNOPSIS

/etc/netman

DESCRIPTION

Netman helps the system administrator monitor and configure the network. It is a form- and menu-based program that can run on a wide variety of terminals. *Netman* arranges for the installation and removal of servers, modifies system and personal network configuration files, and displays network information. Any user can use ***netman***, but if not run by the superuser, certain privileged options will not appear on the "Administration" menu.

Netman requires that the TERM environment variable be set to the terminal's type. TERM is used as an index into **/etc/termcap** (see *termcap*(4)) and **/usr/lib/kmap** (see *kmap*(4)).

Netman displays a series of pop-up windows. It explains keyboard usage for each window as needed, but it is worth knowing some standard keys in advance. The following table gives basic function keys, their alternatives for terminals without labeled function keys, and their standard meanings:

Key	*Alternate*	*Meaning*
↓	Return	Move selection cursor down or to next item. Wraps from end to beginning.
↑	(none)	Move selection cursor up or to previous item. Wraps from beginning to end. In text windows, scroll down text.
Next	Line Feed	Next window. If selection cursor present, choose selected option. If no selection cursor, return to previous window. In text windows, scroll up text.
Cancel	Control-X	Return to previous window. Use this to cancel an option you've already selected. If you Cancel the "Network Management" window, *netman* terminates.

Help	ESC ?	Print help window for current window.

An invalid keystroke prints a window showing which keystrokes are valid for the current window.

At any time, the interrupt character (normally generated by the Delete key) terminates *netman*.

Here is the hierarchy of windows:

Network Management

Machine status

- Displays status of network nodes.

Network Users

- Displays users on various network nodes. The display is actually a menu: you can choose a user and display the processes he or she is running.

Administration

- Modify configuration of the network.

Add an Equivalent User

- Declare another user equivalent to yourself.

Delete an Equivalent User

- Remove a user from your list of equivalent users. Use the arrow keys to step through the list or use the Bound key to display the complete list.

Add an Equivalent Machine

- Declare another node equivalent to yours: a user on the other machine is equivalent to a user with the same name on your machine. Only the superuser has access to this option.

Delete an Equivalent Machine

- Remove a node from your list of equivalent nodes. Use the arrow

keys to step through the list or use the Bound key to display the complete list. Only the superuser has access to this option.

Add a New Network Service

- Turn a server on. Choose from the Remote Login Server (supports *rlogin*(1N)), the Remote Command Server (support *rcmd*(1N) and *rcp*(1N)), Remote Status Server (supports *rwho*(1N) and *netstat*(1N)), the TELNET Protocol Server (supports *telnet*(1N)), or the File Transfer Protocol Server (supports *ftp*(1N)). Only the superuser has access to this option.

Remove a Current Network Service

- Turn a server off. Converse of "Add a Current Network Service."

Add a New Host

- Add to the list of nodes known to the local node. See *hosts*(4N) for an explanation of the various fields.

Change a Host Entry

- Change or delete the description of one of the nodes on the network.

Network Interface Statistics

- List statistics for local node's interface with network.

  Active Connections
  Network Interfaces
  Memory Usage
  Routing Tables
  Protocol Statistics

FILES

/etc/rc
/etc/hosts
/etc/hosts.equiv.
$HOME/.rhosts

NAME

netstat – show network status

SYNOPSIS

netstat [**–Aaimnrs**] [*interval*] [*system*] [*core*]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. The options have the following meanings:

–A show the address of any associated protocol control blocks; used for debugging

–a show the state of all sockets; normally sockets used by server processes are not shown

–i show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown)

–m show statistics recorded by the memory management routines (the network manages a "private share" of memory)

–n show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically)

–s show per-protocol statistics

–r show the routing tables

The arguments *system* and *core* allow substitutes for the defaults **/unix** and **/dev/kmem**.

If an *interval* is specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol.

Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the data bases **/etc/hosts** and **/etc/networks**, respectively. If a symbolic name for an address is unknown, or if the **–n** option is specified, the address is printed in the Internet "dot format"; refer to *rhosts*(4N)

for more information regarding this format. Unspecified, or "wildcard," addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. The refcnt field gives the current number of active uses of the route. Connection-oriented protocols normally hold on to a single route for the duration of a connection, while connectionless protocols obtain a route then discard it. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces, and a column for the interface with the most traffic since the system was last rebooted. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

hosts(4), networks(4), protocols(4), services(4).

NAME

newform – change the format of a text file

SYNOPSIS

newform [**–s**] [**–i**tabspec] [**–o**tabspec] [**–b**n] [**–e**n] [**–p**n] [**–a**n] [**–f**] [**–c**char] [**–l**n] [files]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **–s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "**–e**15 **–l**60" will yield results different from "**–l**60 **–e**15". Options are applied to all *files* on the command line.

–i*tabspec* Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be **––**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to **–8**. A *tabspec* of **–0** expects no tabs; if any are found, they are treated as **–1**.

–o*tabspec* Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **–i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **–8**. A *tabspec* of **–0** means that no spaces will be converted to tabs on output.

–l*n* Set the effective line length to *n* characters. If *n* is not entered, **–l** defaults to 72. The default line length without the **–l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **–i** to expand tabs to spaces).

–b*n* Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **–l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **–b** with no *n*

is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

newform -l1 -b7 file-name

The **-l1** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated.

-e*n* Same as **-b***n* except that characters are truncated from the end of the line.

-c*k* Change the prefix/append character to *k*. Default character for *k* is a space.

-p*n* Prefix *n* characters (see **-c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

-a*n* Same as **-p***n* except characters are appended to the end of a line.

-f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **-o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to

column 72), and the leading digits placed starting at column 73, the command would be:

newform -s -i -l -a -e filename

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*Newform* was called with a bad option.
not -s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

EXIT CODES

0 - normal execution
1 - for any error

SEE ALSO

csplit(1), tabs(1), fspec(4).

BUGS

Newform normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

Newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was **-o--**, and was preceded by either a **-o--** or a **-i--**, the tab specification format line will be incorrect.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [group]

DESCRIPTION

Newgrp changes a user's group identification. The user remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (**PS1**) other than $ (default) and has not exported **PS1**. After an invocation of *newgrp* , successful or not, their **PS1** will now be set to the default prompt string $. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry.

If the first argument to *newgrp* is a –, the environment is changed to what would be expected if the user actually logged in again.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

csh(1), login(1), sh(1), group(4), passwd(4), environ(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(4), environ(5).

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

nl [**–h**type] [**–b**type] [**–f**type] [**–v**start#] [**–i**incr] [**–p**] [**–l**num] [**–s**sep] [**–w**width] [**–n**format] [**–d**delim] file

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

–b*type* Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: **a**, number all lines; **t**, number lines with printable text only; **n**, no line numbering; **p***string*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is **t** (text lines numbered).

–h*type* Same as **–b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

–f*type* Same as **–b***type* except for footer. Default for logical page footer is **n** (no lines

numbered).

-p Do not restart numbering at logical page delimiters.

-v*start#* *Start#* is the initial value used to number logical page lines. Default is **1**.

-i*incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

-s*sep* *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

-w*width* *Width* is the number of characters to be used for the line number. Default *width* is **6**.

-n*format* *Format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

-l*num* *Num* is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). Default is **1**.

-d*xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **-d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nlist – get entries from name list

SYNOPSIS

#include <nlist.h>

int nlist (file-name, nl)
char *file-name;
struct nlist *nl;

DESCRIPTION

Nlist examines the name list in the executable file whose name is pointed to by *file-name*, and selectively extracts a list of values and puts them in the array of nlist structures pointed to by *nl*. The name list *nl* consists of an array of structures containing names of variables, types and values. The list is terminated with a null name; that is, a null string is in the name position of the structure. Each variable 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. The type field will be set to 0 unless the file was compiled with the –g option. If the name is not found, both entries are set to 0. See *a.out*(4) for a discussion of the symbol table structure.

This function 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.

NOTES

The **<nlist.h>** header file is automatically included by **<a.out.h>** for compatability. However, if the only information needed from **<a.out.h>** is for use of *nlist*, then including **<a.out.h>** is discouraged. If **<a.out.h>** is included, the line "#undef n_name" may need to follow it.

SEE ALSO

a.out(4).

DIAGNOSTICS

All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

Nlist returns –1 upon error; otherwise it returns 0.

NAME

ocurse – optimized screen functions

SYNOPSIS

#include <ocurse.h>

DESCRIPTION

Ocurse is the old Berkeley curses library that uses *termcap*(4).

These functions optimally update the screen.

Each *curses* program begins by calling *initscr* and ends by calling *endwin*.

Before a program can change a screen, it must specify the changes. It stores changes in a variable of type **WINDOW** by calling *curses* functions with the variable as argument. Once the variable contains all the changes desired, the program calls *wrefresh* to write the changes to the screen.

Most programs need only a single **WINDOW** variable. *Ocurse* provides a standard **WINDOW** variable for this case and a group of functions that operate on it. The variable is called *stdscr*; its special functions have the same names as the general functions minus the initial w.

FILES

/usr/include/ocurse.h	header file
/usr/lib/libocurse.a	curses library
/usr/lib/libtermcap.a	termcap library, used by curses

SEE ALSO

Ken Arnold. *Screen Updating and Cursor Movement Optimization: A Library Package.* Berkeley, Calif.: University of California.

stty(2), setenv(3), termcap(4).

FUNCTIONS

addch(ch)	Add a character to *stdscr*.
addstr(str)	Add a string to *stdscr*.
box(win,vert,hor)	Draw a box around a window.
crmode()	Set cbreak mode.
clear()	Clear *stdscr*.
clearok(scr,boolf)	Set clear flag for *scr*.
clrtobot()	Clear to bottom on *stdscr*.
clrtoeol()	Clear to end of line on *stdscr*.
delch()	Delete a character.
deleteln()	Delete a line.
delwin(win)	Delete *win*.

NAME

nm – print name list of common object file

SYNOPSIS

nm [**–o**] [**–x**] [**–h**] [**–v**] [**–n**] [**–e**] [**–f**] [**–u**] [**–V**] [**–T**] file-names

DESCRIPTION

The *nm* command displays the symbol table of each common object file *file-name*. *File-name* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

Name The name of the symbol.

Value Its value expressed as an offset or an address depending on its storage class.

Class Its storage class.

Type Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (eg., **char**[n][m]). Note that the object file must have been compiled with the **–g** option of the *cc*(1) command for this information to appear.

Size Its size in bytes, if available. Note that the object file must have been compiled with the **–g** option of the *cc*(1) command for this information to appear.

Line The source line number at which it is defined, if available. Note that the object file must have been compiled with the **–g** option of the *cc*(1) command for this information to appear.

Section For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The output of *nm* may be controlled using the following options:

–o Print the value and size of a symbol in octal instead of decimal.

–x Print the value and size of a symbol in hexadecimal instead of decimal.

-h Do not display the output header data.

-v Sort external symbols by value before they are printed.

-n Sort external symbols by name before they are printed.

-e Print only external and static symbols.

-f Produce full output. Print redundant symbols (.text, .data and .bss), normally suppressed.

-u Print undefined symbols only.

-V Print the version of the nm command executing on the standard error output.

-T By default, *nm* prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **-T** option causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm name -e -v** and **nm -ve name** print the static and external symbols in *name*, with external symbols sorted by value.

FILES

/usr/tmp/nm??????

CAVEATS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **-v** and **-n** options should be used only in conjunction with the **-e** option.

SEE ALSO

as(1), cc(1), ld(1), a.out(4), ar(4).

DIAGNOSTICS

"nm: name: cannot open"
if *name* cannot be read.

"nm: name: bad magic"
if *name* is not an appropriate common object file.

"nm: name: no symbols"
if the symbols have been stripped from *name*.

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, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
tbl ofile | eqn | nroff > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

WARNINGS

nohup command1; command2
: *nohup* applies only to *command1*

nohup (command1; command2)
: is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio -o <list >/dev/rmt/1m&
```

while

```
nohup cpio -o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file *errors*.

NAME

nroff – format text

SYNOPSIS

nroff [options] [files]

DESCRIPTION

Nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers.

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:

–o*list* Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end. (See *BUGS* below.)

–n*N* Number first generated page *N*.

–s*N* Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). This option does not work if the output of *nroff* is piped through *col*(1). When *nroff* halts between pages, an ASCII BEL is sent to the terminal.

–r*aN* Set register *a* (which must have a one-character name) to *N*.

–i Read standard input after *files* are exhausted.

–q Invoke the simultaneous input-output mode of the **.rd** request.

–z Print only messages generated by **.tm** (terminal message) requests.

–m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.

–c*name* Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.**[**nt**].[**dt**].*name* and **/usr/lib/macros/ucmp.**[**nt**].*name*.

–k*name* Compact the macros used in this invocation of *nroff*, placing the output in files [**dt**].*name* in the current directory

–T*name* Prepare output for specified terminal. Known *name*s are **37** for the (default) TELETYPE

Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300s** for the DASI 300s, **300** for the DASI 300, **450** for the DASI 450, **lp** for a (generic) ASCII line printer, **382** for the DTC-382, **4000A** for the Trendata 4000A, **832** for the Anderson Jacobson 832, **X** for a (generic) EBCDIC printer, 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.

-h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

-u*n* Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*

SEE ALSO

col(1), cw(1), eqn(1), greek(1), mm(1), tbl(1), troff(1), mm(5).

BUGS

Nroff 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* generates may be off by one day from your idea of what the date is.

When *nroff* is used with the **-o***list* option inside a pipeline it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

od – octal dump

SYNOPSIS

od [**–bcdosxf**] [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 unsigned decimal.

–o Interpret words in octal.

–s Interpret 16-bit words in signed decimal.

–x Interpret words in hexadecimal.

– f Interpret bytes in hexadecimal with ASCII listing at side.

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

dump(1), hd(1).

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [**–f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.**z** with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 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, except that *pcat* can not be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.z use:

pcat name.z

or just:

pcat name

To make an unpacked copy, say *nnn*, of a packed file named *name*.z (without destroying *name*.z) use the command:

pcat name >nnn

Pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the .z) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

Unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.z (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the "unpacked" name already exists;
if the unpacked file cannot be created.

SEE ALSO

cat(1).

NAME

passwd – change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes or installs a password associated with the login *name*.

Ordinary users may change only the password which corresponds to their login *name*.

Passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has "aged" sufficiently. If "aging" is insufficient the new password is rejected and *passwd* terminates; see *passwd*(4).

Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

Each password must have at least six characters. Only the first eight characters are significant.

Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means upper and lower case letters.

Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

One whose effective user ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password

by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd – password file
/etc/opasswd – password file before password was changed

SEE ALSO

login(1), id(1), su(1), passwd(4).

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* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a 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	combine pairs of lines into lines

SEE ALSO

grep(1), cut(1), pr(1).

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

path – locate executable file for command

SYNOPSIS

path [-options] *command*

DESCRIPTION

Path is a quick way to discover which executable file is behind a shell command. It searches each directory mentioned in your **PATH** environment variable until it finds an executable file called *command*.

Any options specified are passed to *ls*(1).

WARNING

The shell (*sh*(1)) hashes the location of certain commands. Therefore, path and type (shell built-in) may give different results.

SEE ALSO

ls(1).

NAME

pg - file perusal filter for soft-copy terminals

SYNOPSIS

pg [*-number*] [**-p** *string*] [**-cefns**] [*+linenumber*] [*+/pattern/*] [files...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a soft-copy terminal. (The file name - and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

-number
: An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

-p *string*
: Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

-c
: Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

-e
: Causes *pg* *not* to pause at the end of each file.

-f
: Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *-f* option inhibits *pg* from splitting lines.

-n Normally, commands must be terminated by a <*newline*> character. This option causes an automatic end of command as soon as a command letter is entered.

-s Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+*linenumber*
Start up at *linenumber*.

+/*pattern*/
Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be

terminated by a <*newline*>, even if the **-*n*** option is specified.

i/*pattern*/
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^
i?*pattern*?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n**
: Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p**
: Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w**
: Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
: Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <*newline*>, even if the **-*n*** option is specified.

h
: Help by displaying an abbreviated summary of available commands.

q or **Q**
: Quit *pg*.

!*command*
: *Command* is passed to the shell, whose name is

taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the *-n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/*
: Terminal information data base

/tmp/pg*
: Temporary file when input is from a pipe

SEE ALSO

crypt(1), ed(1), grep(1), more(1), terminfo(4).

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options (e.g., *crypt*(1)), terminal settings may not be restored correctly.

NAME

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.

The below *options* may appear singly or be combined in any order:

+*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.

–e*ck* Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).

–i*ck* In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

−n*ck* Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **−m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

−w*k* Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).

−o*k* Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

−l*k* Set the length of a page to *k* lines (default is 66).

−h Use the next argument as the header to be printed instead of the file name.

−p Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

−f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on failure to open files.

−t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.

−s*c* Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

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

protocols – list of Internet protocols

DESCRIPTION

The file **/etc/protocols** lists known DARPA Internet protocols. Each line describes a single protocol and consists of the following blank separated fields:

name number aliases ...

where

name is the official name of the protocol.

number is the protocol number.

aliases ... is a blank-separated list of local aliases for the protocol.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Protocol names and numbers are specified by the SRI Network Information Center. Do not change this file unless you are familiar with DARPA Internet internals.

FILES

/etc/protocols

SEE ALSO

CTIX Internetworking Manual.

NOTE

This command is for use with a special version of the CTIX kernel that supports networking protocols.

NAME

reloc – relocation information for a common object file

SYNOPSIS

#include <reloc.h>

DESCRIPTION

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format.

```
struct      reloc
{
    long    r_vaddr ;
                        /* (virtual) address of reference */
    long    r_symndx ;
                        /* index into symbol table */
    short   r_type ;    /* relocation type */
} ;


/*
 * All generics
 *   reloc. already performed to symbol in the same section
 */
#define  R_ABS          0

/*
 *Motorola Processors 68000, 68010, and 68020
 *
 *
 */
#define R_DIR24         04
#define R_REL24         05
#define R_OPT16         014
#define R_IND24         015
#define R_IND32         016
#define R_RELBYTE       017
#define R_RELWORD       020
#define R_RELLONG       021
#define R_PCRBYTE       022
#define R_PCRWORD       023
#define R_PCRLONG       024
```

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated.

NAME

prof – display profile data

SYNOPSIS

prof [**–tcan**] [**–ox**] [**–g**] [**–z**] [**–h**] [**–s**] [**–m** mdata] [prog]

DESCRIPTION

Prof interprets a profile file produced by the *monitor*(3C) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t, c, a,** and **n** determine the type of sorting of the output lines:

–t Sort by decreasing percentage of total time (default).

–c Sort by decreasing number of calls.

–a Sort by increasing symbol address.

–n Sort lexically by symbol name.

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

–o Print each symbol address (in octal) along with the symbol name.

–x Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

–g Include non-global symbols (static functions).

–z Include all symbols in the profile range (see *monitor*(3C)), even if associated with zero number of calls and zero time.

–h Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

–s Print a summary of several of the monitoring parameters and statistics on the standard error output.

–m mdata
Use file *mdata* instead of **mon.out** as the input profile file.

A program creates a profile file if it has been loaded with the **-p** option of *cc*(1). This option to the *cc* command arranges for calls to *monitor*(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the **-p** option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, "mon.out" is produced in the directory current when the program terminates. If PROFDIR = string, "string/pid.progname" is produced, where *progname* consists of argv[0] with any path prefix removed, and *pid* is the program's process id. If PROFDIR = nothing, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the MARK macro (see *prof*(5)).

FILES

mon.out for profile
a.out for namelist

SEE ALSO

cc(1), exit(2), profil(2), monitor(3C), prof(5).

WARNING

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely, however.

BUGS

Only programs that call *exit*(2) or return from *main* will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the **-p** option *cc*(1) to invoke profiling imposes a limit of 600 (300 on the PDP-11) functions that may have call counters established during program execution. For more counters you must call *monitor*(3C) directly. If this limit is exceeded, other data will be

overwritten and the **mon.out** file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

NAME

prfld, prfstat, prfdc, prfsnap, prfpr – operating system profiler

SYNOPSIS

/etc/prfld [namelist]
/etc/prfstat on
/etc/prfstat off
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [namelist]]

DESCRIPTION

Prfld, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* form a system of programs to facilitate an activity study of the CTIX operating system. A kernel configured with kernel profiling must be used.

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.

Prfdc 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. *Prfdc* will store the counters into *file* every *period* minutes and will turn off at *off_hour* (valid values for *off_hour* are **0–24**). *Prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

Prfpr formats the data collected by *prfdc* 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(7).

NAME

prs – print an SCCS file

SYNOPSIS

prs [**–d**[dataspec]] [**–r**[SID]] [**–e**] [**–l**] [**–c**[date-time]] [**–a**] files

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

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

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

–d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

–r[*SID*] Used to specify the *S*CCS *ID*entification (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed. **–e** and **–l** keyletters. The format for the date is: mm/dd/yy [hh:mm:ss].

–e Requests information for all deltas created *earlier* than and including the delta designated via the **–r** keyletter or the date given by the **–c** option.

–l Requests information for all deltas created *later* than and including the delta designated via the **–r** keyletter or the date given by the **–c** option.

[**–c**[cutoff]] Cutoff date-time, in

the form:

YY[MM[DD[HH[MM[SS]]]]]

-c[*date-time*] 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 in the form:

"-c77/2/2 9:22:25".

-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*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords.

A tab is specified by **\t** and carriage return/new-line is specified by **\n**. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS: ...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS: ...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS: ...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R: ...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S

:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

prs –d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

Users and/or user IDs for s.file are:
xyz
131
abc

prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case:*

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the **–a** keyletter.

FILES

/tmp/pr?????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).
CTIX Programmer's Guide, Section 9.

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, or under window management, processes associated with the current window. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

–e	Print information about all processes.
–d	Print information about all processes, except process group leaders.
–a	Print information about all processes, except process group leaders and processes not associated with a terminal.
–f	Generate a *full* listing. (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/kmem**.
–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 system *namelist* file in place of examining the running system.
–t *termlist*	Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's

file name (e.g., **tty004**) or if the device's file name starts with **tty**, just the digit identifier (e.g., **004**).

-p *proclist* Restrict listing to data about processes whose process ID numbers are given in *proclist*.

-u *uidlist* Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the **-f** option is used, in which case the login name will be printed.

-g *grplist* Restrict listing to data about processes whose process groups are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (*full* or *long*) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

F (l) Flags (octal and additive) associated with the process:
- 1 in core;
- 2 system process;
- 4 locked in core (e.g., for physical I/O);
- 10 being swapped;
- 20 being traced by another process;
- 40 another tracing flag.

S (l) The state of the process:
- 0 non-existent;
- S sleeping;
- W waiting;
- R running;
- I intermediate;
- Z terminated;
- T stopped;
- X growing.

UID (f,l) The user ID number of the process owner; the login name is printed under the **-f** option.

PID (all) The process ID of the process; it is possible to kill a process if you know this datum.

PPID (f,l) The process ID of the parent process.

C	(f,l)	Processor utilization for scheduling.
PRI	(l)	The priority of the process; higher numbers mean lower priority.
NI	(l)	Nice value; used in priority computation.
SZ	(l)	The size in pages (4K bytes) of the core image of the process.
RSZ	(l)	The resident size in pages (4K bytes) of the core image of the process.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
STIME	(f)	Starting time of the process.
TTY	(all)	The controlling terminal for the process. The **t** prefix implies a real terminal; the **s** prefix implies a shell layer; and the **p** prefix implies a virtual terminal.
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/swap	the default swap devices
/etc/passwd	supplies UID information
/dev	searched to find terminal ("tty") names

SEE ALSO

acctcom(1), 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 the file *output* that can be processed with a text formatter to produce a permuted index of file *input* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of each line. *Ptx* output is in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where **.xx** is assumed to be an *nroff* or *troff*(1) macro provided by the user, or provided by the *mptx*(5) macro package. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed. *Tail* and *head*, at least one of which is always the empty string, are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line.

The following *options* can be applied:

–f Fold 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.

–o *only* Use as keywords only the words given in the *only* file.

–i *ignore* Do not use as keywords any words given in the *ignore* file. If the **–i** and **–o** options are missing, use **/usr/lib/eign** as the *ignore* file.

–b *break* Use the characters in the *break* file to separate words. Tab, new-line, and space characters are *always* used as break

characters.

-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
/usr/lib/tmac/tmac.ptx

SEE ALSO

nroff(1), troff(1), mm(5), mptx(5).

BUGS

Line length counts do not account for overstriking or proportional spacing.
Lines that contain tildes (~) are botched, because *ptx* uses that character internally.

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/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 criteria for determining a valid login name is derived from *passwd*(4). 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(4), passwd(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

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 your system administrator.

NAME

qinstall - install and verify software using the mkfs(1) proto file database

SYNOPSIS

/usr/local/bin/qinstall -c [**-esoim**] proto root
/usr/local/bin/qinstall -g [**-p** prefix] [**-t** #] root
/usr/local/bin/qinstall -r [**-q**] proto root from_rfile to_rfile

DESCRIPTION

Qinstall is used to package software on distribution media, to install software, and to verify the correctness of the installation. Output from *qinstall* goes to standard output. *Root* must be a full pathname or ".".

The following options are recognized by *qinstall*:

-V output additional, verbose messages to *stderr*.

-c check whether files under *root* match files in *proto* for owner and permission. This option is used primarily to verify the correctness of an installation, but it is also used in the software distribution packaging process.

- **-o** print omissions from *root*.
- **-s** set permissions and owners to be correct if incorrect
- **-e** print extra files not found in *proto*
- **-i** ignore differences in special files
- **-m** check mounted file systems as well

-g generate a proto file from *root*. This option is used in the software distribution packaging process.

- **-p** add specified prefix to path names
- **-t** specify number of tabs to indent

-r replace file *to_rfile* in *root* with contents of *from_rfile*, keeping permissions as in *proto*. *To_rfile* path must be a full path name as in the proto file, and will be offset with *root*. Multiple from/to pairs may be specified. This option is used to install customizable software.

- **-q** query before replace. Options are to replace *to_rfile* with *from_rfile*, to save *from_rfile*, to ignore *to_rfile*, to perform an *sdiff*(1) between the two files or to replace *to_rfile* with the previous diff.

EXAMPLE

A sample proto file created with the **-g** option follows. (**qinstall -g . > ../proto**)

```
/mkboot
0 0
d--777 2 2
install    d--775 0 0
    IsamRel         ---444 0 0 /install/IsamRel
    $
usr        d--775 2 2
    include         d--775 2 2
        iserc.h     ---444 2 2 /usr/include/iserc.h
        isam.h      ---444 2 2 /usr/include/isam.h
        $
    lib    d--775 2 2
        isam        d--775 2 2
            IsamConfig      ---755 2 2 /usr/lib/isam/IsamConfig
            IsamCreate      ---755 2 2 /usr/lib/isam/IsamCreate
            IsamProtect     ---755 2 2 /usr/lib/isam/IsamProtect
            IsamReorg       ---755 2 2 /usr/lib/isam/IsamReorg
            IsamStat        ---755 2 2 /usr/lib/isam/IsamStat
            IsamStop        ---755 2 2 /usr/lib/isam/IsamStop
            IsamTransfer    ---755 2 2 /usr/lib/isam/IsamTransfer
            IxFilter        ---755 2 2 /usr/lib/isam/IxFilter
            IxSpec          ---755 2 2 /usr/lib/isam/IxSpec
            isam            ---755 2 2 /usr/lib/isam/isam
            $
        libisam.a           ---444  2 2 /usr/lib/libisam.a
        $
    $
$
```

SEE ALSO

qlist(1), ctinstall(1), mkfs(1).

BUGS

Qinstall invoked with the **-m** option on an inconsistent file system produces error messages of the form "filename: cannot stat".

NAME

qlist - print out file lists from proto file; set links based on lines in proto file.

SYNOPSIS

/usr/local/bin/qlist -m [**-d** dir] [**-o**] [**-p** prefix] proto
/usr/local/bin/qlist -l dir [**-p** prefix] proto
/usr/local/bin/qlist -s proto root

DESCRIPTION

Qlist is used in the distribution software packaging process and in the software installation process. It makes lists of files from proto files created by *qinstall*(1). Lists are based on the files' group identifiers and types. *Qlist* also sets links based on lines in the proto file during software installation.

Qlist understands extended proto files, in which a line beginning with **:L** indicates that the first file named is a link to the second file. Other lines beginning with **:** are comments. The last field on a line in an extended proto file is a group identifier of 9 or fewer characters, such as "WP" for the Word Processor product. The following symbols appearing immediately after the group identifier designate the file's type and have the following meanings:

+ designates a customizable file, such as **/etc/passwd**. This type of file is one which the user may or may not want to install over his existing version. This type of file can be installed with the -rq option of *qinstall*(1).

- designates a zero-length file. The specified file should not be used when updating an existing system; rather, it should be used for raw, or first installs only.

@ implies an update but no query from *qinstall*(1). This symbol is used for files required by the installation tools for installation and for possible text busy files.

< designates an optional file, or a file requiring special installation such as a hardware configuration-dependent file. Its associated special installation scripts are GROUP.opt and GROUP.ins, where *GROUP* represents the group name.

< id designates a file of the above category which has special installation scripts named GROUPid.opt,

GROUPid.ins, where *GROUP* represents the group name. *Id* can be 5 or fewer characters. The total number of characters in *GROUP and id must* be 10 or fewer.

The following options are recognized by *qlist*:

-V output additional, verbose messages to *stderr*.

-m make file lists from *proto* file. This option is used in packaging software.

-o print files in no group

-d use *dir* as location for file lists

-p use prefix when printing (default = ./)

File lists output with the **-m** option for group "WP" are named as follows:

+	WP.cust
-	WP.noup
@	WP.noqu
<	WP.fopt, WP.flst
< id	WP.fopt, WPid.lst
the rest	WP

-l list files in directory *dir* from *proto* file to *stdout*.

-p use prefix when printing (default = ./)

-s set links in *root* directory which are indicated by **:L** lines in *proto* file. *Root* must be a rooted path name or ".". This option is used in software installations.

EXAMPLE

A sample extended proto file follows. Note that the files **Document** and **Gloss** are really links to the file **Admin**, as indicated by **:L** at the beginning of these lines. Also note that the lines ending in < designate optionally installed, or specially installed files.

```
/mkboot
0 0
d--777 2 2
install             d--775 0 0
    WPRel                       ---444 0 0 /install/WPRel   WP
    $
oa                  d--775 0 0
    .Key d--755 0 0
         Admin      ---444 2 2 /oa/.Key/Admin               CTIXOA
:L       Document              /oa/.Key/Admin               CTIXOA
```

```
:L      Gloss                   /oa/.Key/Admin                  CTIXOA
        $
    .Document   d--775 0 0
        Recruit ---666 2 2 /oa/.Document/Recruit    WP
        $
    .Gloss      d--775 0 0
        Sample  ---666 2 2 /oa/.Gloss/Sample        WP
        $
    Centronix       ---555 2 2 /oa/Centronix    WP<     sys
    ImagenDriver    ---555 2 2 /oa/ImagenDriver WP<     sys
    SerialDriver    ---555 2 2 /oa/SerialDriver WP<     sys
    abs_rel         ---555 2 2 /oa/abs_rel      WP<     propt
    ctospool        ---555 2 2 /oa/ctospool     WP
    def_wp          ---555 2 2 /oa/def_wp       WP<     propt
    spoolstat       ---555 2 2 /oa/spoolstat    WP
    wp_def          ---555 2 2 /oa/wp_def       WP<     propt
    wp_edit         ---555 2 2 /oa/wp_edit      WP
    wp_merge        ---555 2 2 /oa/wp_merge     WP
    wp_print        ---555 2 2 /oa/wp_print     WP
    wp_review       ---555 2 2 /oa/wp_review    WP
    wpp_band        ---555 2 2 /oa/wpp_band     WP<     propt
    wpp_canprt      ---555 2 2 /oa/wpp_canprt   WP
    wpp_diablo      ---555 2 2 /oa/wpp_diablo   WP<     propt
    wpp_imagen      ---555 2 2 /oa/wpp_imagen   WP<     propt
    wpp_laser       ---555 2 2 /oa/wpp_laser    WP<     propt
    wpp_necspin     ---555 2 2 /oa/wpp_necspin  WP<     propt
    wpp_prtsh       ---555 2 2 /oa/wpp_prtsh    WP
    $
$
```

SEE ALSO

qinstall(1), ctinstall(1).

NAME

rand, srand – simple random-number generator

SYNOPSIS

int rand ( )

void srand (seed)
unsigned seed;

DESCRIPTION

Rand uses a multiplicative congruential random-number generator with period 2^{32} that returns successive pseudo-random numbers in the range from 0 to $2^{15}-1$.

Srand can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NOTE

The spectral properties of *rand* leave a great deal to be desired. *Drand48*(3C) provides a much better, though more elaborate, random-number generator.

SEE ALSO

drand48(3C).

NAME

rcmd, rresvport, ruserok – routines for returning a stream to a remote command

SYNOPSIS

```
rcmd (ahost, inport, locuser, remuser, cmd, fd2p);
char **ahost;
unsigned short inport;
char *locuser, *remuser, *cmd;
int *fd2p;

rresvport (port);
int *port;

ruserok (rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;
```

DESCRIPTION

Rcmd is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. *Rresvport* is a routine which returns a descriptor to a socket with an address in the privileged port space. *Ruserok* is a routine used by servers to authenticate clients requesting service with *rcmd*. All three functions are present in the same file and are used by the *rshd*(1NM) server (among others).

Rcmd looks up the host **ahost* using *getnamehost*(3N), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port *inport*.

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller and given to the remote command as *stdin* and *stdout*. If *fd2p* is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel and will also accept bytes on this channel as being CTIX signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the *stderr* (unit 2 of the remote command) will be made the same as the *stdout* and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in *rshd*(1NM).

The *rresvport* routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by *rcmd* and several other routines. Privileged addresses consist of a port in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to a socket.

Ruserok takes a remote host's name, as returned by a *gethostent*(3N) routine, two user names and a flag indicating if the local user's name is the super-user. It then checks the files **/etc/hosts.equiv** and, possibly, **.rhosts** in the current working directory (normally the local user's home directory) to see if the request for service is allowed. A 1 is returned if the machine name is listed in the **hosts.equiv** file or if the host and remote user name are found in the **.rhosts** file; otherwise *ruserok* returns 0. If the *superuser* flag is 1, the checking of the **host.equiv** file is bypassed.

SEE ALSO

rlogin(1C), rcmd(1C), rexec(3N), rexecd(1NM), rlogind(1NM), rshd(1NM)

BUGS

There is no way to specify options to the *socket* call which *rcmd* makes.

NAME

regcmp, regex – compile and execute regular expression

SYNOPSIS

char *regcmp (string1 [, string2, . . .], (char *)0)
char *string1, *string2, . . .;

char *regex (re, subject[, ret0, . . .])
char *re, *subject, *ret0, . . .;

extern char *__loc1;

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 NULL 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 NULL on failure or a pointer to the next unmatched character on success. A global character pointer *__loc1* 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 a 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 first or last 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. The value *m* is the minimum number and *u* is a number, less than 256, which is the maximum. If only *m* is present (e.g., {m}), it indicates the exact number of times the regular expression is to be

NAME

rcp – remote file copy

SYNOPSIS

/usr/local/bin/rcp [**–r**] file1 [file2 ...] target

DESCRIPTION

Rcp copies files between two nodes. *Rcp* works like the *cp* command (see *cp*(1)), with some extensions but without an option to specify PILF cluster size.

File1 is copied to *target*. If *target* is a directory, one or more files are copied into that directory; the copies have the same names as the originals.

File and directory names follow a convention which is an extension of the normal CTIX convention. Names take one of three forms:

host.user:path
host:path
path

where

host is the name of the system which contains or will contain the file. If no host is specified (the simple *path* form of the name), the system on which the command is executed is assumed.

user is the name of a user on the specified system. If no user is specified (the *host.path* and *path* forms of the name), the user on the remote system whose name is the same as the user who executed the *rcp* command is used.

Access to the file system is as if by the specified user who has just logged in. Created files belong to the specified user and the specified user's group (taken from the password file). File and directory modifications can only occur if the specified user has permission to do them. If *path* does not begin with a slant (/), it is assumed to be relative to the specified user's home directory.

To use a user name on a remote system, the remote system must have declared it "equivalent" to your user name. See *rhosts*(4N).

path is a conventional CTIX/UNIX path name. *Path* can include file name generation sequences (*, ?, [...]); it may be necessary to quote these to prevent their expansion on the local system.

An exclamation point (!) is allowed in place of the colon.

The **-r** (recursive) option copies directory hierarchies. If a file specified for copying is a directory and **-r** is specified, the entire hierarchy under it is copied. When **-r** is specified, *target* must be a directory.

When **-r** is not specified, copying directories is an error.

Note that a third system (not the source or target system of the copy) can execute *rcp*.

EXAMPLES

The following examples are executed on system alpha, by user fred. Alpha is networked to beta and gamma.

The first example copies *list* from fred's home directory on alpha to fred's home directory on beta.

```
rcp list beta:list
```

The next example copies a directory hierarchy. The original is rooted at *src* in fred's home directory on beta. The copy is to be rooted in *src* in the working directory.

```
rcp -r beta:src .
```

Finally, fred copies a file from diane's home directory on beta to */usr/tmp* on gamma; the copy on gamma is to belong to karl. Both diane and karl must have previously declared fred on alpha equivalent to their own user names; see *rhosts*(4N).

```
rcp beta.diane:junk gamma.karl:/usr/tmp
```

Note that *junk* is not placed in karl's home directory because the *path* part of the name begins with a slash.

FILES

/etc/hosts.equiv
$HOME/.rhosts

REQUIREMENTS

Both nodes involved in the copy must be running the *rshd*(1NM) server.

DIAGNOSTICS

Most diagnostics are self-explanatory. "Permission denied" means either that the remote user does not have permission to do what you want or that the remote user is not equivalent to you.

WARNINGS

If a remote shell invoked by *rcp* has output on startup, *rcp* will get confused. This is never a problem with *sh*(1), because it is not called as a login shell.

The **-r** option doesn't work correctly if the copy is purely local. Use *cpio*(1), instead.

NAME

reboot – reboot the system

SYNOPSIS

/etc/reboot

DESCRIPTION

Reboot issues a *syslocal*(2) call to ask the system to wait for the disks to become quiescent and then to reboot the system. The reboot procedure is identical to power-on reset except that the system will not try to take a crash dump.

Only super-user is allowed to execute *reboot*.

NAME

regcmp – regular expression compile

SYNOPSIS

regcmp [–] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program 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) followed 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   "\({0,1}([2-9][01][1-9])$0\){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

regcmp(3X).

NAME

renice – alter priority of running process by changing nice

SYNOPSIS

/etc/renice pid [priority]

DESCRIPTION

Renice can be used by the super-user to alter the priority of a running process. By default, the nice of the process is made 19, which means that it will run only when nothing else in the system wants to. This can be used to nail long running processes that are interfering with interactive work.

Renice can be given a second argument to choose a nice other than the default. Negative nices can be used to make things go very fast.

FILES

/unix
/dev/kmem

SEE ALSO

nice(1).

BUGS

If you make the nice very negative, then the process cannot be interrupted. To regain control you must put the nice back (e.g., to 0).

NAME

rexec – return stream to a remote command

SYNOPSIS

```
rexec (ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
unsigned short inport;
char *user, *passwd, *cmd;
int *fd2p;
```

DESCRIPTION

Rexec looks up the host **ahost* using *getnamehost*(3N), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a user name and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's **.netrc** file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well-known DARPA Internet port to use for the connection; it will normally be the value returned from the call "getnameserv("exec", "tcp")" (see *getservent*(3N)). The protocol for connection is described in *rexecd*(1NM).

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as *stdin* and *stdout*. If *fd2p* is non-zero, then a auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel and will also accept bytes on this channel as being CTIX signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the *stderr* (unit 2 of the remote command) will be made the same as the *stdout* and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO

rcmd(3N), rexecd(1NM).

BUGS

There is no way to specify options to the *socket* call which *rexec* makes.

NAME

scanf, fscanf, sscanf – convert formatted input

SYNOPSIS

```
#include <stdio.h>

int scanf (format [ , pointer ] ... )
char *format;

int fscanf (stream, format [ , pointer ] ... )
FILE *stream;
char *format;

int 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 function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* 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. White-space characters (blanks, tabs, new-lines, or form-feeds) which, except in two cases described below, 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, an optional **l** (ell) or **h** indicating the size of the receiving variable, and a conversion code.

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 *. The suppression of assignment provides a way of describing an input field which is to be skipped. 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. For all descriptors except [and **c**, white space leading an input field is ignored.

NAME

rlogin – remote login

SYNOPSIS

/usr/local/bin/rlogin node [**-ec**] [**-l** name]
/usr/hosts/node [**-ec**] [**-l** name]

DESCRIPTION

Rlogin connects you to a login shell executing on *node*. The second, simplified form of the command is equivalent to the first, but is only available if *mkhosts*(1NM) was previously run by the system administrator. By default, *rlogin* uses the same user name on the remote node that the user is using on the local node. The remote login program will not require a password if the remote node has declared the two users equivalent (see *rhosts*(4N)).

Rlogin attempts to configure the remote "terminal" in a convenient way. The TERM environment variable on the remote shell is automatically set to match its value on the local shell which ran *rlogin*. Echoing takes place at the remote node. Flow control on XON/XOFF and flushing of input and output on interrupts are handled properly.

Close the connection by hanging up on *rlogin*, by logging out of the remote node, or by typing "~." (tilde-period) at the beginning of a line. The hangup and the tilde-period command both cause a hangup on the remote "terminal." To send an input line beginning with tilde to the remote node, begin the line with two tildes.

Rlogin understands the following options.

-ec Use the character *c* instead of tilde as the escape character. There must not be a space between **e** and *c* on the command line. A *c*-period at the beginning of an input line closes the connection, and *cc* at the beginning of an input line sends a single *c*.

-l *user* Login as *user* on the remote system. *User's* password is not required provided that the local user name is on *user's* list of "equivalent" user names. See *rhosts*(4N).

SEE ALSO

rcmd(1N).

NAME

rlogind – remote login server

SYNOPSIS

/etc/rlogind

DESCRIPTION

Rlogind is a network server which supports remote logins by programs such as *rlogin*(1N). It is normally executed by the startup file, **/etc/rc**.

Rlogind enforces an authentication procedure based on equivalence of user names (see *rhosts*(4N)). This procedure assumes all nodes on the network are equally secure.

SEE ALSO

rlogin(1N), rhosts(4N).

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [**–fri**] file ...

rmdir 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 –r*SID* 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 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(4).
CTIX Programmer's Guide, Section 9.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/etc/route [**–f**] [command destination gateway [metric]]

DESCRIPTION

Route is a program used to manually manipulate the network routing tables. It accepts two commands: ***add***, to add a route; and ***delete***, to delete a route.

All commands have the following syntax:

/etc/route *command destination gateway* [*metric*]

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a "local address part" of INADDR_ANY, the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database; see *hosts*(4N). If this lookup fails, the name is then looked for in the network name database; see *networks*(4N).

Route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **–f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

DIAGNOSTICS

"add *node*: **gateway** *node* **flags** *hex-flags*"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

"delete *node*: **gateway** *node* **flags** *hex-flags*"
As above, but when deleting an entry.

"*node node* **done**"
When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

"not in table"
A delete operation was attempted for an entry which wasn't present in the tables.

"routing table overflow"
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO
intro(4), netman(1NM).

NAME

rshd – remote shell server

SYNOPSIS

/etc/rshd

DESCRIPTION

Rshd is the network server for programs such as *rcmd*(1N) and *rcp*(1N) which need to execute a noninteractive shell on remote machines. It is normally executed by the startup file, **/etc/rc**.

Rshd enforces an authentication procedure based on equivalence of user names (see *rhosts*(4n)). This procedure assumes all nodes on the network are equally secure.

SEE ALSO

rcmd(1N), rcp(1N).

NAME

rsterm – manually start and stop terminal input and output

SYNOPSIS

/etc/rsterm number device

DESCRIPTION

Rsterm manually exercises the start/stop features of the terminal driver. (For a discussion of start/stop features, see the STOP and START characters and IXON, IXANY, and IXOFF flags under *termio*(7).) *Rsterm* requires two parameters:

number A number specifying the action:

- *0* Suspend output, as if the terminal had sent a STOP character to the system.
- *1* Resume output, as if the terminal had sent a START character to the system.
- *2* Block input by sending the terminal a STOP character, as if the terminal had nearly filled the terminal's input queue.
- *3* Unblock input by sending the terminal a START character, as if the system had nearly emptied the terminal's input queue.

device The special file for the terminal.

Normally, STOP is the ASCII XOFF character, Control–S, and START is the ASCII XON character, Control–Q.

Operation 2 (resume output) is the most used. Use it when a terminal (a printer for example) has sent a STOP character and cannot be made to send a START character.

Rsterm is one way to clear up a terminal. Another way is to kill all processes associated with the terminal: this momentarily closes the special file, returning all terminal modes to their initial state. See *kill*(1).

You must be superuser to run *rsterm*.

FILES

/dev/tty??? – terminal devices

SEE ALSO

kill(1), termio(7).

NAME

runacct – run daily accounting

SYNOPSIS

/usr/lib/acct/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. Disk block counts are reported for 512-byte blocks.

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

/etc/wtmp
/usr/adm/pacct*
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), mail(1), acct(2), acct(4), utmp(4).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

NAME

ruptime – display status of nodes on local network

SYNOPSIS

/usr/local/bin/ruptime [**–a**] [**–l**] [**–t**] [**–u**]

DESCRIPTION

Ruptime displays status information for nodes on the local network. For each node, a line is printed displaying: the node name; whether the node is up (a node is considered "down" if its *rwhod*(1NM) server has not broadcast in five minutes); the time the node has been up in days, hours, and minutes; the number of logged-in users logged who have used their keyboards in the last hour; and the load (average number of jobs in the run queue) for the last one minute, five minutes, and 15 minutes.

When no options are specified, the status lines are sorted by node name.

Here are the options:

–a Count all logged-in users, including idle ones.

–l Sort status lines by load average.

–t Sort status lines by time node has been up.

–u Sort status lines by number of users.

REQUIRMENTS

Each node to be listed must be running the *rwhod*(1MN) server, which broadcasts a status packet once a minute. The local node must also be running this server to maintain data files.

FILES

/usr/spool/rwho/whod.* data files

SEE ALSO

rwho(1N).

NAME

rwho – who is logged in on local network

SYNOPSIS

/usr/local/bin/rwho [**–a**]

DESCRIPTION

Rwho lists users logged in on machines on the local network in a format similar to that of *who*(1). Without options, only users who have typed in the last hour are listed. For each user listed, *rwho* displays the user name; the node name; and the date and time the user logged in. If the user has not typed in the last minute, *rwho* also displays the user's idle time in hours and minutes.

Rwho understands the following option:

–a List all users on active nodes (users idle for more than an hour are listed).

If information from a node is more than five minutes old, the node is assumed to be down and its users are not listed.

REQUIREMENTS

Each node to be listed must be running the *rwhod* (1MN) server, which broadcasts a status packet once a minute. The local node must also be running this server to maintain the data files.

FILES

/usr/spool/rwho/whod.* information about other nodes

SEE ALSO

ruptime(1N), rwhod(1NM).

NAME

rwhod – node status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

Rwhod collects and distributes information about nodes on the local network, including the local node. It is normally executed by the startup file, **/etc/rc**. It performs four chores once a minute:

- Gathers information about the local node.
- Broadcasts information about the local node for the benefit of *rwhod* servers running on other nodes.
- Collects information broadcast by *rwhod* servers on other nodes.
- Maintains network status files, using information gathered by this and the other *rwhod* servers.

The files maintained by *rwho* have names of the form **/usr/spool/rwho/whod.***name*, where *name* is the name of the host whose status is in the file. Each status file begins with the header of the following form:

```
struct whod {
        char wd_vers;       /* version number */
        char wd_type;       /* type number */
        char wd_fill;       /* ignored */
        int  wd_sendtime;/* time this packet sent */
        int  wd_recvtime;/* time this packet received */
        char wd_hostname[32];
                            /* name of originating node */
        int  wd_loadav[3];
                            /* load averages
                            (see rwho(1N)) */
        int  wd_boottime;
                            /* boot time of originating
                            node */
};
```

The node name of a system is printed by the *uname*(1) command. The remainder of the file consists of user records:

```
struct outmp {
        char out_line[8];   /* terminal name */
```

```
        char out_name[8]; /* user name */
        int  out_ltime;   /* login time */
        int  out_itime;   /* idle time */
};
```

Rwho performs an *nlist*(3) on **/unix** every 10 minutes in case that file is not the current system image.

Rwho transmits and receives messages at the port indicated in the "rwho" service specification. See *services*(4N).

FILES

/usr/spool/rwho/whod.*

WARNINGS

Death of this server makes other nodes think that this node is down.

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

sadp – disk access profiler

SYNOPSIS

sadp [**–th**] [**–d** device[–drive]] s [n]

DESCRIPTION

Sadp reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is done repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of 8 cylinders.

The valid value of *device* is **disk.** *Drive* specifies the disk drives and it may be:

a drive number in the range supported by *device*,
two numbers separated by a minus (indicating an inclusive range),

or

a list of drive numbers separated by commas.

Up to 8 disk drives may be reported. The **–d** option may be omitted, if only one *device* is present.

The **–t** flag causes the data to be reported in tabular form. The **–h** flag produces a histogram on the printer of the data. Default is **–t**.

EXAMPLE

The command:

sadp –d disk –0 900 4

will generate 4 tabular reports, each describing cylinder usage and seek distance of disk drive 0 during a 15-minute interval.

FILES

/dev/kmem

SEE ALSO

MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

sag – system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

Sag graphically displays the system activity data stored in a binary data file by a previous *sar*(1) run. Any of the *sar* data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. *Sag* invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what is available). These *options* are passed through to *sar*:

–s *time* Select data later than *time* in the form hh [:mm]. Default is 08:00.

–e *time* Select data up to *time*. Default is 18:00.

–i *sec* Select data at intervals as close as possible to *sec* seconds.

–f *file* Use *file* as the data source for *sar*. Default is the current daily data file **/usr/adm/sa/sa***dd*.

Other *options*:

–T *term* Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. If *term* is **vpr**, output is processed by **vpr –p** and queued to a Versatec printer. Default for *term* is $TERM.

–x *spec* x axis specification with *spec* in the form: "name [op name] . . . [lo hi]"

–y *spec* y axis specification with *spec* in the same form as above.

Name is either a string that will match a column header in the *sar* report, with an optional device name in square brackets, e.g., **r+w/s[dsk–1]**, or an integer value. *Op* is + – * or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and – have precedence over * and /. Evaluation is left to right. Thus A / A + B * 100 is evaluated (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be

given for **-y**. Enclose the **-x** and **-y** arguments in " " if blanks or \<CR> are included. The **-y** default is:

-y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLES

To see today's CPU utilization:

```
sag
```

To see activity over 15 minutes of all disk drives:

```
TS=`date +%H:%M`
sar -o tempfile 60 15
TE=`date +%H:%M`
sag -f tempfile -s $TS -e $TE -y "r+w/s[dsk]"
```

FILES

/usr/adm/sa/sa*dd* daily data file for day *dd*.

SEE ALSO

sar(1), tplot(1G).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

sar – system activity reporter

SYNOPSIS

sar [**–ubdycwaqvmprAC**] [**–o** file] t [n]

sar [**–ubdycwaqvmprAC**] [**–s** time] [**–e** time] [**–i** sec] [**–f** file]

DESCRIPTION

Sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the **–o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded *file,* either the one specified by **–f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd.* The starting and ending times of the report can be bounded via the **–s** and **–e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **–i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

–u Report CPU utilization (the default):
%usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

–b Report buffer activity:
bread/s, bwrit/s – transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s – accesses of system buffers;
%rcache, %wcache – cache hit ratios, e. g., 1 – bread/lread;
pread/s, pwrit/s – transfers via raw (physical) device mechanism.

–d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk*– is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or

to device, number of bytes transferred in 512-byte units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

-y Report TTY device activity:
rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.

-c Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls.

-w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);
pswch/s – process switches.

-a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

-q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc – run queue of processes in memory and runnable;
swpq-sz, %swpocc – swap queue of processes swapped out but ready to run.

-v Report status of process, i-node, file, record lock and file header tables:
proc-sz, inod-sz, file-sz, lock-sz, fhdr-sz – entries/size for each table, evaluated once at sampling point;
ov – overflows that occur between sampling points for each table.

-m Report message and semaphore activities:
msg/s, sema/s – primitives per second.

-p Report paging activity:
vflt/s, pflt/s, pgfil/s, rclm/s –number of address translation faults, protection faults, page ins from file system and pages reclaimed occurring per second.

-r Report free swap and memory space:
freemem – number of free pages of memory;
freeswp – number of free blocks of swap space; the

free space reported is not contiguous.

-A Report all data. Equivalent to **-udqbwcayvmpC**.

-C

In addition to rates, print the actual counts. This is useful if the rates are 0.

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar -o temp 60 10
```

To later review disk and tape activity from that period:

```
sar -d -f temp
```

FILES

/usr/adm/sa/sa*dd* daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G), sar(1M).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

sa1, sa2, sadc – system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]

/usr/lib/sa/sa1 [t n]

/usr/lib/sa/sa2 [**–ubdycwaqvmprAC**] [**–s** time] [**–e** time] [**–i** sec]

DESCRIPTION

System activity data can be accessed at the special request of a user (see *sar*(1)) and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include CPU utilization counters, buffer usage counters, disk and tape I/O activity counters, TTY device activity counters, switching and system-call counters, file-access counters, queue activity counters, and counters for interprocess communications.

Sadc and shell procedures, *sa1* and *sa2* are used to sample, save, and process this data.

Sadc, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time to mark the time at which the counters restart from zero. The */etc/rc* entry:

```
su sys -c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`"
```

writes the special record to the daily data file to mark the system restart.

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in **/usr/spool/cron/crontabs/sys** (see *cron*(1M)):

```
0 * * * 0,6 /usr/lib/sa/sa1
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3
0 18-7 * * 1-5 /usr/lib/sa/sa1
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar*(1), writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The entry in

/usr/spool/cron/crontabs/sys:

```
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i
3600 -A
```

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si;
        /* see /usr/include/sys/sysinfo.h */
        struct minfo mi;
        /* defined in /usr/include/sys/sysinfo.h */
        int  szinode;
        /* current size of inode table  */
        int  szfile;
        /* current size of file table  */
        int  szproc;
        /* current size of proc table  */
        int   szlckf;
        /* current size of file record header table */
        int   szlckr;
        /* current size of file record lock table */
        int  mszinode;
        /* size of inode table  */
        int  mszfile;
        /* size of file table  */
        int  mszproc;
        /* size of proc table  */
        int   mszlckf;
       /* maximum size of file record header table */
        int   mszlckr;
        /* maximum size of file record lock table */
        long  inodeovf;
        /* cumulative overflows of inode table  */
        long  fileovf;
        /* cumulative overflows of file table  */
        long  procovf;
        /* cumulative overflows of proc table  */
        time_t  ts;
        /* time stamp  */
        int   apstate;
        long  devio[NDEVS][4];
        /* device unit information */
#define IO_OPS0
        /* cumulative I/O requests  */
#define IO_BCNT1
        /* cumulative blocks transferred */
#define IO_ACT2
```

```
                /* cumulative drive busy time in ticks */
        #define IO_RESP3
                /* cumul. I/O resp time in ticks since boot */
        };
```

FILES

/usr/adm/sa/sa*dd* daily data file
/usr/adm/sa/sar*dd* daily report file
/tmp/sa.adrfl address file

SEE ALSO

cron(1M), sag(1G), sar(1), timex(1).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

sccsdiff –r*SID1* –r*SID2* [–**p**] [–**s***n*] 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.

–r*SID?*	*SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.
–**p**	pipe output for each file through *pr*(1).
–**s***n*	*n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).
CTIX Programmer's Guide, Section 9.

DIAGNOSTICS

"*file*: No differences"	If the two versions are the same.

Use *help*(1) for explanations.

NAME

script – make typescript of terminal session

SYNOPSIS

script [**-a**] [**-q**] [**-S** shell] [file]

DESCRIPTION

Script makes a typescript of your interaction with the system. *Script* forks a shell with standard input and output diverted to pipes. Input to *script* is written to the shell's input pipe; *script* writes the shell's output pipe; and a typescript of both is written to *file*. The default for *file* is *typescript*. *File* begins and ends with time stamps for the session. *Script* terminates with an error if *file* already exists.

To terminate *script*, terminate the shell or type Control-D (FINISH on Convergent Technologies Terminals). A Control-D to *script* terminates the shell and all programs run from the shell by closing the pipes. Control-D behaves as though you had typed an infinite number of Control-Ds.

The run file for the shell is taken from the SHELL environment variable, set by *login*(1M). If **SHELL** is not set, **/bin/sh** is used.

Here are the options:

-q Quiet operation. *Script's* opening and closing messages are suppressed, as are the time stamps at the beginning and end of *file*.

-S *shell*
Use *shell* as the name of the shell run file.

-a If *file* already exists, append typescript to it.

WARNINGS

Script's limitations result from its use of pipes:

There is no way to send an end-of-file to the shell without terminating *script*.

Programs that use the standard input to examine and control the user's terminal will have problems or not work at all. Examples are *stty*(1), *tset*(1), *tty*(1), *ex*(1), and *vi*(1).

When the user interrupts a printing process, *script* attempts to flush the output backed up in the pipe for better response. Usually the next prompt also gets flushed.

NAME

sdb – symbolic debugger

SYNOPSIS

sdb [**–w**] [**–W**] [objfil [corfil [directory-list]]]

DESCRIPTION

Sdb is a symbolic debugger that can be used with C programs. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

Objfil is normally an executable program file which has been compiled with the **–g** (debug) option; if it has not been compiled with the **–g** option, or if it is not an executable file, the symbolic capabilities of *sdb* will be limited, but the file can still be examined and the program debugged. 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. A – in place of *corfil* will force *sdb* to ignore any core image file. The colon separated list of directories (*directory-list*) is used to locate the source files used to build *objfil*.

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. Otherwise, they are set to the first line in *main()*. The current line and file may be changed with the source file examination commands.

Initially *sdb* has a greater-than character (>) prompt, which indicates that *sdb* is ready for the user to enter the first command. After *sdb* has begun, the prompt is <*x*>, where *x* is the name of the last command given.

By default, warnings are provided if the source files used in producing *objfil* cannot be found, or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the **–W** flag.

Names of variables are written just as they are in C. Note that names in C are now of arbitrary length, *sdb* will no longer truncate names. 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*]. Pointers may be dereferenced by using the form *pointer*[0]. Combinations of these forms may also be used. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands. Generally, *sdb* will interpret a structure as a set of variables. Thus, *sdb* will display the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable*[*number*][*number*]..., or as *variable*[*number,number,...*]. In place of *number*, the form *number;number* may be used to indicate a range of values, * may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. All the variations mentioned in naming variables may be used. *Number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If no procedure is specified, the procedure currently executing is used by default.

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.

While a process is running under *sdb*, all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of **-w** permits overwriting

locations in *objfil*.

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 (*b1, e1, f1*) and (*b2, e2, f2*) and the *file address* corresponding to a written *address* is calculated as follows:

b1 address < e1

file address = *address* + *f1* - *b1*

otherwise

b2 address < e2

file address = *address* + *f2* - *b2*,

otherwise, the requested *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.

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

In order for *sdb* to be used on large files, all appropriate values are kept as signed 32-bit integers.

Commands

The commands for examining data in the program are:

t Print a stack trace of the terminated or halted program.

T Print the top line of the stack trace.

variable[/*clm*]

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

b	one byte
h	two bytes (half word)
l	four bytes (long word)

Legal values for *m* are:

c	character
d	decimal
u	decimal, unsigned

o	octal
x	hexadecimal
f	32-bit single precision floating point
g	64-bit double precision floating point
s	Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.
a	Print characters starting at the variable's address. This format may not be used with register variables.
p	pointer to procedure
i	disassemble machine-language instruction with addresses printed numerically and symbolically.
I	disassemble machine-language instruction with addresses just printed numerically.

The length specifiers are only effective with the formats **c**, **d**, **u**, **o** and **x**. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, *sdb* choses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, then this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells *sdb* to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count 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, variables local to the current procedure and global variables are matched; if a procedure name is specified then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

linenumber?*lm*
variable:?*lm*
Print the value at the address from **a.out** or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

variable=*lm*
linenumber=*lm*
number=*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, a character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted '*character*. Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression which indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

f Print the 68881 floating-point registers.

x Print the machine registers and the current machine-language instruction.

X Print the current machine-language instruction.

The commands for examining source files are:

e *procedure*
e *file-name*
e *directory/*
e *directory file-name*
The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, file name, or directory is given, the

current procedure name and file name 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 omitted.

?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 deleted.

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.

w Window. Print the 10 lines around the current line.

number
Set the current line to the given line number. Print the new current line.

count+
Advance the current line by *count* lines. Print the new current line.

count–
Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

count **r** *args*
count **R**
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.

linenumber **c** *count*
linenumber **C** *count*
Continue after a breakpoint or interrupt. If *count* is given, it specifies the breakpoint at which to stop after ignoring *count* - 1 breakpoints. **C** continues with the signal which caused the program to stop reactivated and **c** ignores it. If a line number is specified, a temporary breakpoint is placed at the line and execution is continued. This

temporary breakpoint is deleted when the command finishes.

linenumber **g** *count*
Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

s *count*
S *count*
Single step the program through *count* lines. If no count is given then the program is run for one line. **S** is equivalent to **s** except it steps through procedure calls.

i
I Single step by one machine-language instruction. **I** steps with the signal which caused the program to stop reactivated and **i** ignores it.

variable\$**m** *count*
address:**m** *count*
Single step (as with **s**) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

level **v**
Toggle verbose mode, for use when single stepping with **S**, **s** or **m**. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

k Kill the program being debugged.

procedure(arg1,arg2,...)
procedure(arg1,arg2,...)/*m*
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**.

linenumber **b** *commands*
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 **-g** option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, 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. If **k** is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

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.

l 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
Perform the previous command again.

control-D
Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last.

< *filename*
Read commands from *filename* until the end of file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; < may not appear as a command in a file.

M Print the address maps.

M [?/][*] *b e f*
Record new values for the address map. The arguments **?** and **/** specify the text and data maps, respectively. The first segment, (*b1, e1, f1*), is changed unless * is specified, in which case the second segment (*b1, e1, f1*), of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

" *string*
Print the given string. The C escape sequences of the form *\character* are recognized, where *character* is a nonnumeric character.

q Exit the debugger.

The following commands also exist and are intended only for debugging the debugger:

V Print the version number.
Q Print a list of procedures and files being debugged.
Y Toggle debug output.

Sdb may be instructed to monitor a given memory location and stop the program when the value at that location changes in any given way. For example:

> if x <= 123 The above example instructs *sdb* to monitor the value at location x. When the user gives the command to continue (c), *sdb* checks the value of x at every source line executed and stops the program if the given condition becomes true. Note that use of this construct slows the real-time execution of a program.

The syntax of the *if* command is as follows:

if Shows a list of the current data breakpoints; assigns a number to each.

if *var* Monitors the value of *var* and stops the program if the value changes. A variable name may be used for *var*, as well as a constant address. Comparisons are done as either 4-byte signed or 4-byte unsigned, depending on the data type. To perform a 1-byte or 2-byte comparison, an optional length value may accompany *var*. An example of a 2-byte comparison is

if x,2 = 0xff

if *var rel value*
Compares the value of *var* to the constant given

and stops the program if the condition is true. The values of *rel* may be =, ==, <, <=, >, >=, or !=.

off *n* Disables or turns off a data breakpoint without removing it from the list.

on *n* Enables a breakpoint that was turned off.

out *n* Removes a breakpoint from the list.

Conditional breakpoints are used in a manner similar to data breakpoints, except that the user specifies a place in the program at which *sdb* should stop to check the data values. For example,

mysub:99 b if xyz = 123

The above example instructs *sdb* to check the value of xyz every time the program arrives at line 99 of subroutine *mysub*. If the condition is true, then execution stops there, as with a normal breakpoint. This type of breakpoint does not monitor the value xyz at every line of code, as the data breakpoint does.

FILES

a.out
core

SEE ALSO

cc(1), sh(1).
a.out(4), core(4).

WARNINGS

When *sdb* prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The value is assumed to be **int** (integer).

Data which are stored in text sections are indistinguishable from functions.

Line number information in optimized functions is unreliable, and some information may be missing.

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.

When setting a breakpoint at a procedure, *sdb* will inconsistently produce the incorrect line number. This seems to occur when the object file is newer than the source file. Recompiling the source program will correct this problem.

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

Sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

The following options exist:

–w *n* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

–l Only print the left side of any lines that are identical.

–s Do not print identical lines.

–o *output* Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l append the left column to the output file

r append the right column to the output file

s turn on silent mode; do not print identical lines

v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the ***output*** file.

SEE ALSO
diff(1), ed(1).

NAME

sed – stream 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.

(1) **a**
text Append. Place *text* on the output before reading the next input line.

(2) **b** *label* Branch to the **:** command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) **c**
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** Delete the pattern space. Start the next cycle.

(2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** Replace the contents of the pattern space by the contents of the hold space.

(2) **G** Append the contents of the hold space to the pattern space.

(2) **h** Replace the contents of the hold space by the contents of the pattern space.

(2) **H** Append the contents of the pattern space to the hold space.

(1) **i**
text Insert. Place *text* on the standard output.

(2) **l** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) **p** Print. Copy the pattern space to the standard output.

(2) **P** Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) **q** Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

- **n** n= 1 - 512. Substitute for just the nth occurrence of the *regular expression*.
- **g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
- **p** Print the pattern space if a replacement was made.
- **w** *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) **t** *label* Test. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) **w** *wfile*
Write. Append the pattern space to *wfile*.

(2) **x** Exchange the contents of the pattern and hold spaces.

(2) **y**/*string1*/*string2*/
Transform. Replace all occurrences of characters in *string1* with the corresponding

character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*
: Don't. Apply the *function* (or group, if *function* is { } only to lines *not* selected by the address(es).

(0) : *label*
: This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) =
: Place the current line number on the standard output as a line.

(2) {
: Execute the following commands through a matching } only when the pattern space is selected.

(0) #
: If a # appears as the first character on the first line of a script file, that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).
CTIX Programmer's Guide, Section 15.

NAME

setaddr – set DARPA Internet address from node name

SYNOPSIS

/etc/setaddr

DESCRIPTION

Setaddr sets the DARPA Internet address of the system. It gets the address by using the node name to find the local system's entry in **/etc/hosts** (see hosts (4)). The node name must have been previously set by the *setuname*(1M) command or *setuname*(2) system call. *Setaddr* is executed at boot time. It should appear in **/etc/rc** (see *brc*(1M)) after *setuname* and before the networking servers.

FILES

/etc/hosts names and addresses of hosts

NAME

setenet – write Ethernet address on disk

SYNOPSIS

/etc/setenet address

DESCRIPTION

Setenet writes an Ethernet Address, specified by its parameter, into the volume home block of disk 0. It is run once only before any other Ethernet software. The Ethernet Address is a 32-bit integer that uniquely identifies each computer system running Ethernet. *Setenet* requires it to be in hexadecimal.

Setenet employs *iv*(1) to actually modify the volume home block; this produces some irrelevant information about the bad block table.

FILES

/dev/fp000 reserved area of disk 0

SEE ALSO

iv(1), fp(7).

NAME

setmnt – establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

Setmnt creates the **/etc/mnttab** table (see *mnttab*(4)), which is needed for both the *mount*(1M) and *umount* 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., **dsk/c?d?s?**) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the *mnttab*(4) entry.

FILES

/etc/mnttab

SEE ALSO

devnm(1M), mount(1M), mnttab(4).

BUGS

Evil things will happen if *filesys* or *node* are longer than 32 characters.

Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

setuname – set name of system

SYNOPSIS

/etc/setuname [**-s** sysname] [**-n** nodename] [**-r** release] [**-v** *version*]

DESCRIPTION

Setuname sets the values reported by *uname*. Options set the same things that they report in *uname*.

Only the superuser can execute *setuname* successfully.

SEE ALSO

uname(1), uname(2).

NAME

sh, rsh – shell, the standard/restricted command programming language

SYNOPSIS

sh [**−acefhiknrstuvx**] [args]
rsh [**−acefhiknrstuvx**] [args]

DESCRIPTION

Sh is a new version of the shell. It replaces the shell in a previous version of CTIX. Its new features include:

- Functions: user-defined built-in commands. See *Commands* below.
- Remembering both previously used and user-specified executable files in a hash table. See *Execution* and *Special Commands* (for the **hash** command) below.
- New parameters for mail checking and accounting. See *Parameter Substitution* below.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, −, $, and !.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by | (or, for historical compatibility, by ^). The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or | |, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and | |. The symbols && and | | also have equal precedence. A semicolon (;)

causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (**| |**) 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*) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] . . . [**else** *list*] **fi**
: The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
: A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)
Execute *list* in a sub-shell.

{*list*;}
list is simply executed.

name () {*list*;}
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (` `) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution

The character $ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter*:-*word*}
If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:=*word*}
If *parameter* is not set or is null set it to *word*;

the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:?*word*}
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:+*word*}
If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:- `pwd` }
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the **set** command.
? The decimal value returned by the last synchronously executed command.
$ The process number of this shell.
! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.
PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.
CDPATH The search path for the *cd* command.
MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.
MAILCHECK This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH	A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is ***you have mail***.
PS1	Primary prompt string, by default **$**.
PS2	Secondary prompt string, by default >.
IFS	Internal field separators, normally **space**, **tab**, and **new-line**.
SHACCT	If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed. Accounting routines such as *acctcom*(1) and *acctcms*(1M) can be used to analyze the data collected.
SHELL	When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (" " or ' ') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears, the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, 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. If the first character following the opening [is a !, any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (' '), except a single quote, are quoted. Inside double quote marks (" "), parameter and command substitution occurs and \ quotes the characters \, `, ", and $. **"$*"** is equivalent to **"$1 $2 ..."**, whereas **"$@"** is equivalent to **"$1" "$2" ...**.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, 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, it is created; otherwise, it is truncated to zero length.
>>word	Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) **\newline** is ignored, and \ must be used to quote the characters \, $, `, and the first character of *word*. If - is appended to <<, all leading tabs are stripped from *word* and from the document.

<&digit The standard input is duplicated from file descriptor *digit* (see *dup*(1)). Similarly for the standard output using >.

<&- The standard input is closed. Similarly for the standard output using >.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by **&**, the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see *environ*(5)) is a list of name-value

pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set** **-a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd
and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by *&*; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command

name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash –r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by PATH is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified, resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter CDPATH defines the search path for

the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a /, the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

echo [*arg* . . .]
Echo arguments. See *echo*(1) for usage and description.

eval [*arg* . . .]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* . . .]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted, the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* . . .]
The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

hash [**−r**] [*name* . . .]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **−r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are

indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *name*s are marked *readonly* and the values of the these *name*s may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**--aefhkntuvx** [*arg* ...]]

- **-a** Mark variables which are modified or created for export.
- **-e** Exit immediately if a command exits with a non-zero exit status.
- **-f** Disable file name generation
- **-h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- **-k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **-n** Read commands but do not execute them.
- **-t** Exit after reading and executing one command.
- **-u** Treat unset variables as an error when substituting.
- **-v** Print shell input lines as they are read.
- **-x** Print commands and their arguments as they are executed.
- **--** Do not change any of the flags; useful in setting **$1** to -.

Using + rather than − causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given, the values of all names are printed.

shift [*n*]
The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test
Evaluate conditional expressions. See *test*(1) for usage and description.

times
Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
The command *arg* is 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, all trap(s) *n* are reset to their original values. If *arg* is the null string, this signal is ignored by the shell and by the commands it invokes. If *n* is 0, the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [**−f**] [*n*]
imposes a size limit of *n*
−f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.
If no option is given, **−f** is assumed.

umask [*nnn*]
The user file-creation mask is set to *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* . . .]
For each *name*, remove the corresponding variable or function. The variables PATH, PS1, PS2, MAILCHECK and IFS cannot be unset.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given, all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is –, commands are initially read from **/etc/profile** and 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, commands are read from *string*.

–s If the **–s** flag is present or if no arguments remain, commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

–i If the **–i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

–r If the **–r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **$PATH**,
specifying path or command names containing /,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

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 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

acctcom(1), acctcms(1M), cd(1), csh(1), echo(1), env(1), expr(1), login(1), newgrp(1), pwd(1), test(1), umask(1), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), a.out(4), profile(4), environ(5).

NOTE

If the first character in an executable file is #, *csh* assumes that the file is a *csh* script. For compatibility with *csh*, it is recommended that *sh* scripts begin with a blank line.

WARNINGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

NAME

shl – shell layer manager

SYNOPSIS

shl

DESCRIPTION

Shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty* option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *Shl* has its own prompt, >>>, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (**/dev/sxt???**). The virtual device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* **(1)** through **(7)** cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create [*name*]
Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where # is the last digit of the virtual device bound to the layer. The shell prompt variable **PS1** is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block *name* [*name* ...]
For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option –**loblk** within the layer.

delete *name* [*name* ...]
For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)).

help (or **?**)
Print the syntax of the *shl* commands.

layers [**-l**] [*name* ...]
For each *name*, list the layer name and its process group. The **-l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **-loblk** within the layer.

quit Exit *shl*. All layers are sent the SIGHUP signal.

name Make the layer referenced by *name* the current layer.

FILES

/dev/sxt/???	Virtual tty devices
$SHELL	Variable containing path name of the shell to use (default is **/bin/sh**).
/etc/drvload	

SEE ALSO
sh(1), stty(1), ioctl(2), signal(2), sxt(7).
MightyFrame Administrator's Reference Manual.

NOTE
The *sxt* driver must be loaded before *shl* can be used.

NAME

shutdown, halt – terminate all processing

SYNOPSIS

/etc/shutdown [*grace*]
/etc/halt

DESCRIPTION

Shutdown shuts down CTIX in an orderly manner. It cautiously terminates all currently running processes. 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. User's are given *grace* seconds (default 60). to log out on their own.

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*(1)). This must be done before re-booting the system, to insure file system integrity.

Halt shuts down CTIX in a safe but abrupt way. It is meant for small installations where verbal warnings are faster than terminal messages.

DIAGNOSTICS

The most common error diagnostic that will occur is *device busy*. This diagnostic happens when a particular file system could not be unmounted.

SEE ALSO

mount(1M), sync(1), update(1M).
MightyFrame Administrator's Reference Manual.
MiniFrame Administrator's Manual.

NAME

size – print section sizes of common object files

SYNOPSIS

size [**–o**] [**–x**] [**–V**] files

DESCRIPTION

The *size* command produces section size information for each section in the common object files. The size of the text, data and bss (uninitialized data) sections are printed along with the total size of the object file. If an archive file is input to the *size* command the information for all archive members is displayed.

Numbers will be printed in decimal unless either the **–o** or the **–x** option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The **–V** flag will supply the version information on the *size* command.

SEE ALSO

as(1), cc(1), ld(1), a.out(4), ar(4).

DIAGNOSTICS

size: name: cannot open
if *name* cannot be read.

size: name: bad magic
if *name* is not an appropriate common object file.

NAME

slattach, sldetach – attach and detach serial lines as network interfaces

SYNOPSIS

/etc/slattach devname source destination [baudrate]

/etc/sldetach interface-name

DESCRIPTION

Slattach is used to assign a serial (tty) line to a network interface using the DARPA Internet Protocol, and to define the source and destination network addresses. The *devname* parameter is the name of the device the serial line is attached to, *e.g.*, /dev/tty001. The source and destination are either host names present in the host name data base (see ***hosts***(4)), or DARPA Internet addresses expressed in the Internet standard "dot notation". The optional *baudrate* parameter is used to set the speed of the connection; if not specified, the default of 9600 is used.

Only the superuser may attach or detach a network interface.

Sldetach is used to remove the serial line that is being used for IP from the network tables and allow it to be used as a normal terminal again. *Interface-name* is the name that is shown by ***netstat***(1N); in general, an interface on tty0*xx* will be referred to as sl*xx*.

EXAMPLES

```
/etc/slattach tty001 tom-src genstar
/etc/slattach /dev/tty001 hugo dahl 4800
/etc/sldetach sl01
```

FILES

/etc/hosts, /dev/*

DIAGNOSTICS

Various messages indicating:

- the specified interface does not exist
- the requested address is unknown
- the user is not the superuser

SEE ALSO

hosts(4), netstat(1N), ifconfig(1NM).

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 2,147,483,647 seconds.

NAME

sort – sort and/or merge files

SYNOPSIS

sort [**–cmu**] [**–o**output] [**–y**kmem] [**–z**recsz] [**–dfiMnr**] [**–b**tx] [+pos1 [–pos2]] [files]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if – is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

–c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

–m Merge only, the input files are already sorted.

–u Unique: suppress all but one in each set of lines having equal keys.

–o*output*

The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **–o** and *output*.

–y*kmem*

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **–y0** is guaranteed to start with minimum memory. By convention, **–y** (with no argument) starts with maximum memory.

–z*recsz*

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **–c** or **–m** options, a popular system default size

will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

-d "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

-f Fold lower case letters into upper case.

-i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

-M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **-M** option implies the **-b** option (see below).

-n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

-r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 -pos2* restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *-pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

−t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

−b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **−b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *−pos2* argument (see below).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; *+m.***0b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *−m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *−m.*1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 −2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort −r −o outfile +1.0 −1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort −r +1.0b −1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

sort -t: +2n -3 /etc/passwd

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

sort -um +2 -3 infile

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the **-c** option. When the last line of an input file is missing a **new-line** character, *sort* appends one, prints a warning message, and continues.

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [**–v**] [**–b**] [**–x**] [**–l**] [**–i**] [+local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

Spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

Spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the **–v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **–b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the **–x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(1)) follows chains of included files (**.so** and **.nx** *troff*(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **–l** option, *spell* will follow the chains of *all* included files. Under the **–i** option, *spell* will ignore all chains of included files.

Under the *+local_file* option, words found in *local_file* are removed from *spell*'s output. *Local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy-y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin *n* Reads *n* sorted hash codes from the standard input and writes a compressed spelling list on the standard output. Information about the hash coding is printed on standard error.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

EXAMPLES

The following example creates the hashed spell list **hlist** and checks the result by comparing the two temporary files; they should be equal.

```
cat goodwds | /usr/lib/spell/hashmake | sort -u >tmp1
cat tmp1 | /usr/lib/spell/spellin `cat tmp1 | wc -l` >hlist
cat hlist | /usr/lib/spell/hashcheck >tmp2
diff tmp1 tmp2
```

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(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 file that is added to the hashed *spelling_list* via *spellin*.

The British spelling feature 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*(1G).

The following *options* are recognized, each as a separate argument:

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, 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(1G).

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 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if – is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

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 **c5**. 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 [**-c***i*] – absolute value
columns (similarly for **-c** options that follow)

af [**-c***i* **t v**] – arithmetic function
titled output, **v**erbose

ceil [**-c***i*] – round up to next integer

cusum [**-c***i*] – cumulative sum

exp [**-c***i*] – exponential

floor [**-c***i*] – round down to next integer

gamma [**-c***i*] – gamma

list [**-c***i* **d***string*] – list vector elements
delimiter(s)

log [**-c***i* **b***f*] – logarithm
base

mod [**-c***i* **m***f*] – modulus
modulus

pair [**-c***i* **F***file* **x***i*] – pair elements
File containing base vector, **x** group size

power [**-c***i* **p***f*] – raise to a power
power

root [**-c***i* **r***f*] – take a root
root

round [**-c***i* **p***i* **s***i*] – round to nearest integer, .5 rounds to 1
places after decimal point,
significant digits

siline [**-c***i* **i***f* **n***i***s***f*] – generate a line given slope and intercept
intercept, **n**umber of positive integers, **s**lope

sin [**-c***i*] – sine

subset [**-a***f* **b***f* **c***i* **F***file* **i***i* **l***f* **nl np p***f* **s***i* **t***i*] – generate a subset
above, **b**elow, **F**ile with master vector, **i**nterval, **l**eave, master contains element **n**umbers to **l**eave, master contains element **n**umbers to **p**ick, **p**ick, **s**tart, **t**erminate

Summarizers:

bucket [**-a***i* **c***i* **F***file* **h***f* **i***i* **l***f* **n***i*] – break into buckets
average size, **F**ile containing bucket boundaries, **h**igh, **i**nterval, **l**ow, **n**umber
Input data should be sorted

cor [**-F***file*] – correlation coefficient
File containing base vector

hilo [**– h l o ox oy**]– find high and low values
high only, **l**ow only, **o**ption form, **o**ption form with **x** prepended, **o**ption form with **y** prepended

lreg [**-F***file* **i o s**] – linear regression
File containing base vector, **i**ntercept only, **o**ption form for ***siline***, **s**lope only

mean [**-f***f* **n***i* **p***f*] – (trimmed) arithmetic mean
fraction, **n**umber, **p**ercent

point [**-f***f* **n***i* **p***f* **s**] – point from empirical cumulative density function
fraction, **n**umber, **p**ercent, **s**orted input

prod – internal product

qsort [**-c***i*] – quick sort

rank – vector rank

total – sum total

var – variance

Translators:

bar [**-a b f g r***i* **w***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] – build a bar chart
suppress **a**xes, **b**old, suppress **f**rame, suppress **g**rid, **r**egion, **w**idth in percent, **x** origin, suppress **x**-axis label, **y** origin, suppress **y**-axis label, **y**-axis lower bound, **y**-axis **h**igh bound
Data is rounded off to integers.

hist [**-a b f g r***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] – build a histogram
suppress **a**xes, **b**old, suppress **f**rame, suppress **g**rid, **r**egion, **x** origin, suppress **x**-axis label, **y** origin, suppress **y**-axis label, **y**-axis lower bound, **y**-axis **h**igh bound

label [**-b c F***file* **h p r***i* **x xu y yr**] – label the axis of a GPS file
bar chart input, retain **c**ase, label **F**ile, **h**istogram input, **p**lot input, **r**otation, **x**-axis, **u**pper **x**-axis, **y**-axis, **r**ight **y**-axis

pie [**-b o p pn***i* **pp***i* **r***i* **v x***i* **y***i*] – build a pie chart
bold, values **o**utside pie, **v**alue as **p**ercentage(:=100), value as **p**ercentage(:=i), draw **p**ercent of **p**ie, **r**egion, no **v**alues, **x** origin, **y** origin
Unlike other nodes, input is lines of the form

[< **i e f c***c* >] value [label]
ignore (do not draw) slice, **e**xplode slice, **f**ill slice, **c**olor slice *c* = (**b**lack, **r**ed, **g**reen, **b**lue)

plot [**-a b c***string* **d f F***file* **g m r***i* **x***f* **xa xi***f* **xh***f* **xl***f* **xn***i* **xt y***f* **ya yi***f* **yh***f* **yl***f* **yn***i* **yt**] – plot a graph
suppress **a**xes, **b**old, plotting **c**haracters, **d**isconnected, suppress **f**rame, **F**ile containing **x** vector, suppress **g**rid, **m**ark points, **r**egion, **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 l***string* **v***string* **u***string*] – title a vector or a GPS
title **b**old, retain **c**ase, **l**ower title, **u**pper title, **v**ector title

Generators:

gas [**-c***i* **i***f* **n***i* **s***f* **t***f*] – generate additive sequence
interval, **n**umber, **s**tart, **t**erminate

prime [**-c***i* **h***i* **l***i* **n***i*] – generate prime numbers
high, **l**ow, **n**umber

rand [**-c***i* **h***f* **l***f* **m***f* **n***i* **s***i*] – generate random sequence
high, **l**ow, **m**ultiplier, **n**umber, **s**eed

RESTRICTIONS
Some nodes have a limit on the size of the input vector.

SEE ALSO

graphics(1G), strtod(3C), gps(4).

NAME

strings – extract the ASCII text strings in a file

SYNOPSIS

strings [**–a**] [**–o**] [–#] file ...

DESCRIPTION

Strings looks for ASCII text strings in a file. It is useful for examining and identifying object and other binary files. A string is any sequence of 4 or more printing characters ending with a newline or a null. If the file is an object file, the search is restricted to the initialized data space.

Here are the options:

–a Don't restrict object file searches.

–o Precede each string with its octal offset.

–# Make # the minimum string length instead of 4.

SEE ALSO

od(1).

WARNING

The algorithm for identifying strings is rather primitive.

NAME

strip – strip symbol and line number information from a common object file

SYNOPSIS

strip [**–l**] [**–x**] [**–r**] [**–s**] [**–V**] filename

DESCRIPTION

The ***strip*** command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

–l Strip line number information only; do not strip any symbol table information.

–x Do not strip static or external symbol information.

–r Reset the relocation indexes into the symbol table.

–s Reset the line number indexes into the symbol table (do not remove). reset the relocation indexes into the symbol table.

–V Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, ***strip*** will complain and terminate without stripping *file-name* unless the **–r** flag is used.

If the ***strip*** command is executed on a common archive file (see *ar*(4)) the archive symbol table will be removed. The archive symbol table must be restored by executing the *ar*(1) command with the **s** option before the archive can be link-edited by the *ld*(1) command. *Strip* will instruct the user with appropriate warning messages when this situation arises.

The purpose of this command is to reduce the file storage overhead taken by the object file.

FILES

/usr/tmp/strp??????

SEE ALSO

ar(1), as(1), cc(1), ld(1), a.out(4), ar(4).

DIAGNOSTICS

strip: name: cannot open
if *name* cannot be read.

strip: name: bad magic
if *name* is not an appropriate common object file.

strip: name: relocation entries present; cannot strip
if *name* contains relocation entries and the **-r** flag is not used, the symbol table information cannot be stripped.

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 *termio*(7). Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (**–parenb**) enable (disable) parity generation and detection.

parodd (**–parodd**) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *termio*(7)).

0 hang up phone line immediately.

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 19200 Set terminal baud rate to the number given, if possible; (all speeds are not supported by all hardware interfaces); **extb** stands for 38400 or for external clock synchronization, depending on the device.

hupcl (**–hupcl**) hang up (do not hang up) a 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.

loblk (**–loblk**) block (do not block) output from a non-current layer.

Input Modes

ignbrk (**–ignbrk**) ignore (do not ignore) break on input.

brkint (–brkint)	signal (do not signal) INTR on break.
ignpar (–ignpar)	ignore (do not ignore) parity errors.
parmrk (–parmrk)	mark (do not mark) parity errors (see *termio*(7)).
inpck (–inpck)	enable (disable) input parity checking.
istrip (–istrip)	strip (do not strip) input characters to seven bits.
inlcr (–inlcr)	map (do not map) NL to CR on input.
igncr (–igncr)	ignore (do not ignore) CR on input.
icrnl (–icrnl)	map (do not map) CR to NL on input.
iuclc (–iuclc)	map (do not map) upper-case alphabetics to lower case on input.
ixon (–ixon)	enable (disable) START/STOP output control. Output is stopped by sending 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 *termio*(7)).

nl0 nl1 select style of delay for line-feeds (see *termio*(7)).

tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *termio*(7) or *stermio*(7)).

bs0 bs1 select style of delay for backspaces (see *termio*(7)).

ff0 ff1 select style of delay for form-feeds (see *termio*(7)).

vt0 vt1 select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (–isig) enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.

icanon (–icanon) enable (disable) canonical input (ERASE and KILL processing).

xcase (–xcase) canonical (unprocessed) upper/lower-case presentation.

echo (–echo) echo back (do not echo back) every character typed.

echoe (–echoe) echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (–echok) echo (do not echo) NL after KILL character.

lfkc (–lfkc) the same as **echok (–echok)**; obsolete.

echonl (–echonl) echo (do not echo) NL.

noflsh (–noflsh) disable (enable) flush after INTR, QUIT, or SWTCH.

Control Assignments

control-character c set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **eof**, **ctab**, **min**, or **time** (**ctab** is used with **–stappl**; see *stermio*(7)), (**min** and **time** are used with **–icanon**; see *termio*(7)). 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.

line *i* — set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity** — enable **parenb** and **cs7**.

oddp — enable **parenb**, **cs7**, and **parodd**.

-parity, **-evenp**, or **-oddp** — disable **parenb**, and set **cs8**.

raw (**-raw** or **cooked**) — enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

nl (**-nl**) — unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**-lcase**) — set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (-LCASE) — same as **lcase** (**-lcase**).

tabs (**-tabs** or **tab3**) — preserve (expand to spaces) tabs when printing.

ek — reset ERASE and KILL characters back to normal # and @.

sane — resets all modes to some reasonable values.

term — set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

Cluster Terminals

Options which are meaningless to the RS-422 interface are ignored by cluster terminals. See *termio*(7) for specifics.

SEE ALSO

tabs(1), ioctl(2), termio(7).

NAME

su – become super-user or another user

SYNOPSIS

su [–] [name [arg . . .]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (i.e., super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or **/bin/sh** if none is specified (see *sh*(1)). To restore normal user ID privileges, type an **EOF** (*cntrl-d*) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **–c** *string* executes *string* via the shell and an arg of **–r** will give the user a restricted shell. 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.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a –, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the the program used as the shell with an *arg0* value whose first character is –, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of $PATH, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **profile** can check *arg0* for **–sh** or **–su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(1).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

EXAMPLES

To become user **bin** while retaining your previously

exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

```
su - bin -c "command args"
```

FILES

/etc/passwd	system's password file
/etc/profile	system's profile
$HOME/.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1), passwd(4), profile(4), environ(5).

NAME

sum – print checksum and block count of a file

SYNOPSIS

sum [**–r**] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **–r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

swap – swap administrative interface

SYNOPSIS

/etc/swap –a swapdev [swaplow [swaplen]]
/etc/swap –d swapdev [swaplow]
/etc/swap –l

DESCRIPTION

Swap provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

–a Add the specified swap area. *Swapdev* is the name of block special device, e.g., **/dev/dsk/c0d0s2**. *Swaplow* is the offset in 1 Kbyte blocks into the device where the swap area should begin. *Swaplen* is the length of the swap area in 1 Kbyte blocks up to the size of the specified partition. This option can only be used by the super-user. Swap areas are normally added by the system start up routine **/etc/drvload**.

–d Delete the specified swap area. *Swapdev* is the name of block special device, e.g., **/dev/dsk/c0d0s2**. *Swaplow* is the offset in 1 Kbyte blocks into the device where the swap area should begin. Using this option marks the swap area as "being deleted." The system will not allocate any new blocks from the area, and will try to free swap blocks from it. The area will remain in use until all blocks from it are freed. This option can only be used by the super-user.

–l List the status of all the swap areas. The output has four columns:

DEV The *swapdev* special file for the swap area if one can be found in the **/dev/dsk** or **/dev** directories, and its major/minor device number in decimal.

LOW The *swaplow* value for the area in 1 Kbyte blocks.

LEN The *swaplen* value for the area in 1 Kbyte blocks.

FREE The number of free 1 Kbyte blocks in the area. If the swap area is being deleted, this column will be marked **(indel)**.

NAME

system – system description file

DESCRIPTION

The system description describes tunable variables and hardware configuration to the CTIX system.

The file is formatted in sections. Each section begins with a section header (a ! followed by a single word). Each section varies in format, depending upon the format required by the program that uses the data provided by that section.

In the example file the **!VMESLOTS** section describes the VME boards for the EEPROM. The slot field is the slot position in the VME bus. The type field is the board type; board types may be:

1	CMC Ethernet board
2	Interphase SMD disk controller board
3	Xylogics 1/2-inch tape controller board

The address field is the location of the board. The length field is the address space size of the board. The optional initialization function name is an initialization function that is called by the PROM at boot time.

The **!VMECODE** section consists of a list of files that describe the executable code to be loaded into the EEPROM. This section is required only if a bootable initialization function was specified.

EXAMPLE

```
!FILENAMES
PROM_IFILE=/etc/lddrv/EEPROM.ifile
EEPROM_FILE=/dev/vme/eeprom
INIT_CFILE=tunevar.c
!VMESLOTS
* The following section describes the VME boards
*
*slot    type    address          length   [Initialization
*                                          function name ]
*
0        2       C1000000         512      initVs32
1        2       C1000200         512
*one CMC Ethernet controller)
2        1       C0DE0200         131072
*
!VMECODE
diskvs32.o
```

SEE ALSO

lddrv(1M), ldeeprom(1M), mktunedrv(1M), vme(7).
MightyFrame Administrator's Reference Manual.

FILES

/etc/system
/dev/vme/eeprom

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. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

SEE ALSO

sync(2).

NAME

tabs – set tabs on a terminal

SYNOPSIS

tabs [tabspec] [**+mn**] [**–T**type]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

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., CTIX "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+2**n*, 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 CTIX "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff* **–h** option for high-speed output. Another special case is the value **–0**, implying no tabs at all.

n1,n2,...
The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

––file If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **–8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

tabs –– file; pr file

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

–T*type* *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(5). If no **–T** flag is supplied, *tabs* searches for the **$TERM** value in

the *environment* (see *environ*(5)). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+m*n* The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. 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

illegal tabs	when arbitrary tabs are ordered incorrectly.
illegal increment	when a zero or missing increment is found in an arbitrary specification.
unknown tab code	when a "canned" code cannot be found.
can't open	if *--file* option used, and file can't be opened.
file indirection	if *--file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

nroff(1), environ(5), term(5).

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

NAME

tail - deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *-number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **-f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. 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. As another example, the command:

```
tail -15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1), head(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. The ***key*** may not include spaces. 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. Blocked tapes cannot be appended.

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. If a named file on tape does not exist on the system, the file is created with the same mode as the one on tape except that the set-user-ID and set-group-ID bits are not set unless you are super-user. If the files exist, their modes are not changed except for the bits described above. The owner, group, and modification time 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.

t The names of all the files 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. Blocked tapes (including QIC tapes) cannot be overwritten.

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:

#s Where **#** is a tape drive number (**0,...,7**), and **s** is the density (**l** - low (800 bpi), **m** - medium

(1600 bpi), or **h** - high (6250 bpi)). This modifier selects the drive on which the tape is mounted. The default is **0**. The density option is ignored on some tapes, such as QIC tapes.

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. By default, **/dev/mt/c0d0??** and **/dev/rqic/c0d?**, respectively, are tried. If the name of the file is **–**, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *Tar* can also be used to move hierarchies with the command:

```
cd fromdir; tar cf - . | (cd todir;
tar xf -)
```

b Causes *tar* to use the next argument as the blocking factor for tape records (512 bytes). The default is 1 for most tape drives, 128 for QIC tape, and the maximum is 128. This option should only be used with raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes (key letters **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* not to restore the modification times. The modification time of the file will be the time of extraction.

o Causes extracted files to take on the user and group identifier of the user running the program rather than those on the tape.

The following command may be used to archive onto a QIC tape:

```
cd dir; tar c
```

FILES

/dev/mt/c0d?
/dev/rqic/c0d0??
/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.
Note that **tar c0m** is not the same as **tar cm0**.

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [–TX] [files]

DESCRIPTION

Tbl is a preprocessor that formats tables for *nroff*. 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 semicolon (;).

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 (or if - is specified as the last argument), *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn*(1) or *neqn*, *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:

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

SEE ALSO

neqn(1), mm(1), nroff(1), mm(5), mv(5).

BUGS

See *BUGS* under *nroff*(1).

NAME

tdl, gtdl, ptdl – RS-232 terminal download

SYNOPSIS

/usr/local/bin/tdl [type]
/usr/local/bin/gtdl [runfile]
/usr/local/bin/ptdl [runfile]

DESCRIPTION

Tdl, gtdl, and *ptdl* download a terminal system image over a an RS-232 line. The program is run from the terminal that is to receive the system image, which must be a Convergent Technologies terminal running in boot ROM emulation mode.

Type is a number that specifies one of the standard terminal types. If *type* is omitted, *tdl* sends an escape sequence to the terminal to discover its type. (If the user has not used the boot ROM **T** command, the escape sequence produces a "101" on a Programmable Terminal and a "201" on a Graphics Terminal. These cause *tdl* to download **/usr/lib/iv/ws101.232** or **/usr/lib/iv/ws201.232**, respectively.)

Runfile is the name of a download image file.

Ptdl and *gtdl* require a terminal with a Release 1.0 boot ROM. *Tdl* requires a terminal with a Release 2.0 boot ROM.

To use *tdl*, follow this procedure:

1. Turn the terminal on while holding down the space bar. Be sure to keep the space bar down until the boot ROM prompt appears on the screen.
2. Use the boot ROM commands to set whatever communication options you need. Do not use the **T** (system image type) command unless you need a nonstandard type.
3 Enter the boot ROM **E** (emulate serial terminal) command.
4. If necessary, establish a connection with the host system and log in as *tdl, ptdl,* or *gtdl.*
5. Run *tdl, ptdl,* or *gtdl* with no parameters.

To allow users to download their terminals by logging in, for example, as **tdl,** add the appropriate login entries to **/etc/passwd**:

```
tdl::50:1:Terminal Down Load:/:/usr/local/bin/tdl
ptdl::51:1:PT232 Download (1.0 boot ROM) :/:/usr/local/bin/ptdl
```

gtdl::52:1:GT232 Download (1.0 boot ROM) :/:/usr/local/bin/gtdl

The download area must be specified on the disk; see *iv*(1).

FILES

/usr/lib/iv/ws*.232	CTIX copies of the system images
/usr/local/bin/ws*.232	checked if system image not in /usr/lib/iv

When acting on a type sent from the terminal, *tdl* downloads **/usr/liv/iv/ws*xxx*.232**, where *xxx* is the three-digit terminal type. If that file is missing, *tdl* looks for **/usr/local/bin/ws*xxx*.232.**

SEE ALSO

iv(1).
Programmable Terminal Programmer's Guide.
Graphic Terminal Programmer's Guide.

DIAGNOSTICS

The terminal displays dashs (–) to indicate successfully transmitted blocks, questions marks (?) to indicate nonfatal tranmission errors. A fatal transmission error produces an appropriate message from the terminal and a return to the boot ROM emulate code; you may need to press the RETURN key to get a shell prompt.

WARNINGS

Tdl, tdtl, and *ptdl* do not verify that the download file is a valid terminal system image.

The 2.0 GT boot ROM does not support downloading run images greater than 65,536 bytes. Attempting to download images greater than 65,536 bytes may cause the terminal to fail.

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

term – format of compiled term file.

SYNOPSIS

term

DESCRIPTION

Compiled terminfo descriptions are placed under the directory **/usr/lib/terminfo**. In order to avoid a linear search of a huge CTIX system directory a two-level scheme is used: **/usr/lib/terminfo/c/name** where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, *act4* can be found in the file **/usr/lib/terminfo/a/act4**. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the *tic*(1M) program, and read by the routine *setupterm*. Both of these pieces of software are part of *curses*(3X). The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal 0432); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value −1 is represented by 0377, 0377; other negative values are illegal. The −1 generally means that a capability is missing from this terminal. Note that this format corresponds to the hardware of the VAX and PDP-11. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the '|' character. The section is terminated with an ASCII NUL character.

database, and always corresponds to the old **termcap** capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short and to allow the tabs in the source file **caps** to line up nicely. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

(P) indicates that padding may be specified

(G) indicates that the string is passed through tparm with parameters as given (#*i*).

(*) indicates that padding may be based on the number of lines affected

($\#_i$) indicates the i^{th} parameter.

Variable Booleans	Cap-name	I. Code	Description
auto_left_margin,	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin,	am	am	Terminal has automatic margins
beehive_glitch,	xsb	xb	Beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch,	xhp	xs	Standout not erased by overwriting (hp)
eat_newline_glitch,	xenl	xn	newline ignored after 80 cols (Concept)
erase_overstrike,	eo	eo	Can erase overstrikes with a blank
generic_type,	gn	gn	Generic line type (e.g.,, dialup, switch).
hard_copy,	hc	hc	Hardcopy terminal
has_function_line	hfl	hf	Terminal has a function key label line
has_meta_key,	km	km	Has a meta key (shift, sets parity bit)
has_status_line,	hs	hs	Has extra "status line"
insert_null_glitch,	in	in	Insert mode distinguishes nulls
memory_above,	da	da	Display may be retained above the screen
memory_below,	db	db	Display may be retained below the screen
move_insert_mode,	mir	mi	Safe to move while in insert mode
move_standout_mode,	msgr	ms	Safe to move in standout modes
over_strike,	os	os	Terminal overstrikes
status_line_esc_ok,	eslok	es	Escape can be used on the status line

teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch,	hz	hz	Hazeltine; can not print ~'s
transparent_underline,	ul	ul	underline character overstrikes
xon_xoff,	xon	xo	Terminal uses xon/xoff handshaking
Numbers:			
columns,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
line_attribute	ldaat	LA	Line drawing character attribute
lines,	lines	li	Number of lines on screen or page
lines_of_memory,	lm	lm	Lines of memory if > lines. 0 means varies
magic_cookie_glitch,	xmc	sg	Number of blank chars left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed
virtual_terminal,	vt	vt	Virtual terminal number (UNIX system)
width_status_line,	wsl	ws	No. columns in status line
Strings:			
back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	change to lines #1 through #2 (vt100) (PG)
clear_all_tabs,	tbc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Term. settable cmd char in prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
cursor_down,	cud1	do	Down one line
cursor_home,	home	ho	Home cursor (if no cup)
cursor_invisible,	civis	vi	Make cursor invisible
cursor_left,	cub1	le	Move cursor left one space
cursor_mem_address,	mrcup	CM	Memory relative cursor addressing
cursor_normal,	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right,	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll,	ll	ll	Last line, first column (if no cup)
cursor_up,	cuu1	up	Upline (cursor up)
cursor_visible,	cvvis	vs	Make cursor very visible
delete_character,	dch1	dc	Delete character (P*)

delete_line,	dl1	dl	Delete line (P*)
dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
enter_blink_mode,	blink	mb	Turn on blinking
enter_bold_mode,	bold	md	Turn on bold (extra bright) mode
enter_ca_mode,	smcup	ti	String to begin programs that use cup expand center; lw(1.4i) lw(.4i) lw(.4i) lw(1.8i).
enter_delete_mode,	smdc	dm	Delete mode (enter)
enter_dim_mode,	dim	mh	Turn on half-bright mode
enter_insert_mode,	smir	im	Insert mode (enter);
enter_protected_mode,	prot	mp	Turn on protected mode
enter_reverse_mode,	rev	mr	Turn on reverse video mode
enter_secure_mode,	invis	mk	Turn on blank mode (chars invisible)
enter_standout_mode,	smso	so	Begin stand out mode
enter_underline_mode,	smul	us	Start underscore mode
erase_chars	ech	ec	Erase #1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
exit_attribute_mode,	sgr0	me	Turn off all attributes
exit_ca_mode,	rmcup	te	String to end programs that use cup
exit_delete_mode,	rmdc	ed	End delete mode
exit_insert_mode,	rmir	ei	End insert mode
exit_standout_mode,	rmso	se	End stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ich1	ic	Insert character (P)
insert_line,	il1	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (p*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC	Sent by clear screen or erase key
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdch1	kD	Sent by delete character key
key_dl,	kdl1	kL	Sent by delete line key
key_down,	kcud1	kd	Sent by terminal down arrow key
key_eic,	krmir	kM	Sent by rmir or smir in insert mode
key_eol,	kel	kE	Sent by clear-to-end-of-line key
key_eos,	ked	kS	Sent by clear-to-end-of-screen key
key_f0,	kf0	k0	Sent by function key f0

key_f1,	kf1	k1	Sent by function key f1
key_f10,	kf10	ka	Sent by function key f10
key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9
key_home,	khome	kh	Sent by home key
key_ic,	kich1	kI	Sent by ins char/enter ins mode key
key_il,	kil1	kA	Sent by insert line
key_left,	kcub1	kl	Sent by terminal left arrow key
key_ll,	kll	kH	Sent by home-down key
key_npage,	knp	kN	Sent by next-page key
key_ppage,	kpp	kP	Sent by previous-page key
key_right,	kcuf1	kr	Sent by terminal right arrow key
key_sf,	kind	kF	Sent by scroll-forward/down key
key_sr,	kri	kR	Sent by scroll-backward/up key
key_stab,	khts	kT	Sent by set-tab key
key_up,	kcuu1	ku	Sent by terminal up arrow key
keypad_local,	rmkx	ke	Out of "keypad transmit" mode
keypad_xmit,	smkx	ks	Put terminal in "keypad transmit" mode
lab_f0,	lf0	l0	Labels on function key f0 if not f0
lab_f1,	lf1	l1	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10
lab_f2,	lf2	l2	Labels on function key f2 if not f2
lab_f3,	lf3	l3	Labels on function key f3 if not f3
lab_f4,	lf4	l4	Labels on function key f4 if not f4
lab_f5,	lf5	l5	Labels on function key f5 if not f5
lab_f6,	lf6	l6	Labels on function key f6 if not f6
lab_f7,	lf7	l7	Labels on function key f7 if not f7
lab_f8,	lf8	l8	Labels on function key f8 if not f8
lab_f9,	lf9	l9	Labels on function key f9 if not f9
ld_upleft	ldul	TL	Upper left corner box character
ld_upright	ldur	TR	Upper right corner box character
ld_botleft	ldul	BL	Bottom left corner box character
ld_botright	ldbl	BR	Bottom right corner box character
ld_vertleft	ldvl	VL	Left-hand side box character
ld_vertright	ldvr	VR	Right-hand side box character
ld_hortop	ldht	TH	Top side box character
ld_horbot	ldhb	BH	Bottom horizontal box character
ld_upleft	ldul	TL	Upper left corner box character
ld_upleft	ldul	TL	Upper left corner box character
ld_upleft	ldul	TL	Upper left corner box character
meta_on,	smm	mm	Turn on "meta mode" (8th bit)
meta_off,	rmm	mo	Turn off "meta mode"

newline,	nel	nw	Newline (behaves like cr followed by lf)
pad_char,	pad	pc	Pad character (rather than null)
parm_dch,	dch	DC	Delete #1 chars (PG*)
parm_delete_line,	dl	DL	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC	Insert #1 blank chars (PG*)
parm_index,	indn	SF	Scroll forward #1 lines (PG)
parm_insert_line,	il	AL	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP	Move cursor up #1 lines (PG*)
pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit,	pfx	px	Prog funct key #1 to xmit string #2
print_screen,	mc0	ps	Print contents of the screen
prtr_off,	mc4	pf	Turn off the printer
prtr_on,	mc5	po	Turn on the printer
repeat_char,	rep	rp	Repeat char #1 #2 times. (PG*)
reset_1string,	rs1	r1	Reset terminal completely to sane modes.
reset_2string,	rs2	r2	Reset terminal completely to sane modes.
reset_3string,	rs3	r3	Reset terminal completely to sane modes.
reset_file,	rf	rf	Name of file containing reset string
restore_cursor,	rc	rc	Restore cursor to position of last sc
row_address,	vpa	cv	Vertical position absolute set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
set_tab,	hts	st	Set a tab in all rows, current column
set_window,	wind	wi	Current window is lines #1-#2 cols #3-#4
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column #1
underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1	Upper left of keypad
key_a3,	ka3	K3	Upper right of keypad
key_b2,	kb2	K2	Center of keypad
key_c1,	kc1	K4	Lower left of keypad
key_c3,	kc3	K5	Lower right of keypad
prtr_non,	mc5p	pO	Turn on the printer for #1 bytes

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the *terminfo* file as of this writing.

```
concept100 | c100| concept | c104 | c100-4p | concept 100,
  am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
cols#80, cr=^M$<9>, cub1=^H, cud1=^J, cuf1=\E=,
cup=\Ea%p1%' '%+%c%p2%' '%+%c,
cuu1=\E;, cvvis=\EW, db, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\Ek$<20>\EK, ht=\t$<8>,
il1=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
lines#24, mir, pb#9600, prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
rev=\ED, rmcup=\Ev  $<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\200,
smcup=\EU\Ev  8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
smul=\EG, tabs, ul, vt#8, xenl,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments may be included on lines beginning with "#". Capabilities in *terminfo* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has ***automatic margins*** (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as **el** (clear to end of line sequence) are given by the two-character code, an '=', and then a string ending at the next following ','. A delay in milliseconds may appear anywhere in such a capability, enclosed in $<..> brackets, as in **el**=\EK$<3>, and padding characters are supplied by *tputs* to provide this delay. The delay can be either a number, e.g., '20', or a number followed by an '*', i.e., '3*'. A '*' indicates that the padding required is proportional to the number of lines affected

by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of *lines* affected. This is always one unless the terminal has **xenl** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. Both **\E** and **\e** map to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \l \r \t \b \f \s** give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include \^ for ^, \\ for \, \, for comma, \: for :, and \0 for null. (\0 will produce \200, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a \.

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *terminfo* and to build up a description gradually, using partial descriptions with *vi* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *terminfo* file to describe it or bugs in *vi*. To easily test a new terminal description you can set the environment variable TERMINFO to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in */usr/lib/terminfo*. To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit /etc/passwd at 9600 baud, delete 16 or so lines from the middle of the screen, then hit the 'u' key several times quickly. If the terminal messes up, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given

by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over, for example, you would not normally use '**cuf1**= ' because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful

for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the *terminfo* file usually assumes that this is on; i.e., **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as

33 | tty33 | tty | model 33 teletype,
bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,

while the Lear Siegler ADM-3 is described as

adm3 | 3 | lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H, cud1=^J,
ind=^J, lines#24,

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with *printf*(3S) like escapes **%x** in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special **%** codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary.

The **%** encodings have the following meanings:

%%	outputs '%'
%d	print pop() as in printf
%2d	print pop() like %2d
%3d	print pop() like %3d
%02d	
%03d	as in printf
%c	print pop() gives %c
%s	print pop() gives %s
%p[1-9]	push ith parm
%P[a-z]	set variable [a-z] to pop()

%g[a-z]	get variable [a-z] and push it
%'c'	char constant c
%{nn}	integer constant nn
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop() op pop())
%& %| %^	bit operations: push(pop() op pop())
%= %> %<	logical operations: push(pop() op pop())
%! %~	unary operations push(op pop())
%i	add 1 to first two parms (for ANSI terminals)

%? expr %t thenpart %e elsepart %;

if-then-else, %e elsepart is optional.
else-if's are possible ala Algol 68:
%? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e %;
c_i are conditions, b_i are bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use "%gx%{5}%-".

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is cup=6\E&%p2%2dc%p1%2dY.

The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, cup=^T%p1%c%p2%c. Terminals which use %c need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n** **^D** and **\r**, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cup=\E=%p1%' '%+%c%p2%' '%+%c. After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values)

and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence (as with the hp2645) and can be used in preference to **cup** . If there are parameterized local motions (e.g., move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on HP terminals cannot be used for **home**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **Ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**. If the terminal has a settable scrolling region (like the vt100) the command to set this can be described with the **csr**

capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command – the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *terminfo*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type abc def using local cursor motions (not spaces) between the abc and the def. Then position the cursor before the abc and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the abc shifts over to the def which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for insert null. While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode

cannot be described with the single attribute.

Terminfo can describe both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals which send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir**/**rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* *characters*, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode*, representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given

as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking) **bold** (bold or extra bright) **dim** (dim or half-bright) **invis** (blanking or invisible text) **prot** (protected) **rev** (reverse video) **sgr0** (turn off *all* attribute modes) **smacs** (enter alternate character set mode) and **rmacs** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking 7 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 7 parameters are, in order: standout, underline, reverse, blink, dim, bold, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist.

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **flash**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of both of these modes.

NAME

telnet – user interface to TELNET protocol

SYNOPSIS

/usr/local/bin/telnet [node]

DESCRIPTION

Telnet establishes connections to other nodes using the TELNET protocol. It is more general than *rlogin*(1N) because TELNET servers run under a wider variety of operating systems. However, *rlogin* is more convenient to use.

Establishing a Single Connection

If *node* is specified, *telnet* establishes a connection to that node. *Node* can be a node name or a DARPA Internet address in dot notation (see *hosts*(4N)). While the connection remains open, *telnet* is in input mode (see below). When the connection is closed, *telnet* terminates. Usually, the remote system closes the connection when you log out. To close the connection yourself, use the escape character to enter the **close** command (see below).

Command Mode

If *node* is not specified, *telnet* enters command mode. Telnet prints its prompt ("telnet>") and understands the following commands. *Telnet* understands a truncated command name as long as it isn't ambiguous ("ope" is valid; "op" is not).

? [*command*]	Give summaries of commands. If *command* is specified, give summary of just that command.
AO	Send the telenet command **AO** (abort output) and an out-of-band signal to the remote server with **DM** as the synchronizing mark.
AYT	Send the telenet command **AYT** (are you there?) and an out-of-band signal to the remote server with **DM** as the synchronizing mark.
BREAK	Send the telnet command **BREAK** to the remote server.
EC	Send the telnet command *EC* (erase character) to the remote server.

EL Send the telnet command *EL* (erase line) to the remote server.

IP Send the telenet command **IP** (interrupt process) and an out-of-band signal to the remote server with **DM** as the synchronizing mark.

SYNCH Send an out-of-band signal to the remote server with **DM** as the synchronizing mark.

crmod Toggle carriage return mode. Initially carriage return mode is off. When carriage return mode is on, carriage return characters from the remote host are expanded to a carriage return followed by a line feed.

close Close the current connection. Useful only with the escape character (see "Input Mode," below).

do *option* Tell the remote server to process *option*. This command is used mostly for testing option negotiation.

dont *option* Tell the remote server to stop processing *option*. This command is used mostly for testing option negotiation.

escape Change the escape character used in input mode (see below). *Telnet* prompts for a new escape character; press a key that generates a single character, then press the RETURN key. To leave the escape character unchanged, press RETURN without entering a character.

help [*command*] Give summaries of commands. If *command* is specified, give summary of just that command.

open *node*	Open a connection to *node*. While the connection remains open, *telnet* is in input mode (see below). If you close the connection with a *telnet* command from input mode, *telnet* returns to command mode; if the connection is closed from the other end, *telnet* terminates. Usually, the remote system closes the connection when you log out.
options	Toggle viewing of TELNET options negotiations. Initially viewing is off. When viewing is on, *telnet* shows its negotiations with the *telnetd*.
quit	Close any open connection and terminate *telnet*.
status	Show the current connection and escape character.
will *option*	Tell the remote server we will process *option*. This command is used mostly for testing option negotiation.
wont *option*	Tell the remote server we won't process *option*. This command is used mostly for testing option negotiation.
escape character	Send the escape character to the remote host.

Input Mode

Telnet enters input mode when a connection is opened and leaves it when a connection is closed. In input mode all text typed goes to the remote node except when the escape character is typed.

To enter a single *telnet* command without first closing the connection, press the escape character at any time in input mode. Initially the escape character is control-[(ASCII GS; octal 035); *Telnet* gives its prompt ("telnet>") and executes a single command line instead of sending it to the remote node. After you press the RETURN key and the command is executed, *telnet* resumes sending your input to the remote node, unless your command closed the connection (**close** or **quit**).

Use the **escape** command to change the escape character.

Telnet Options

Once a connection is established, both sides negotiate various options to get the best possible service. The following options are recognized:

BINARY
Controls transmission of binary data.

ECHO
Controls echoing.

SGA Suppress go ahead.

STATUS
Status of options.

TM Timing Mark.

EXOPL
Extended Options List.

SEE ALSO

rlogin(1N), telnetd(1NM).

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd [**–d**] [port]

DESCRIPTION

Telnetd is a server which supports the DARPA standard TELNET virtual terminal protocol. The TELNET server operates at the port indicated in the "telnet" service description; see *services*(4). This port number may be overridden (for debugging purposes) by specifying a port number on the command line. If the **–d** option is specified, each socket created by *telnetd* will have debugging enabled (see SO_DEBUG in *socket*(2N)).

Telnetd operates by allocating a virtual-terminal device (see *vt*(7)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. *Telnetd* manipulates the master side of the pseudo terminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, *telnetd* sends a TELNET option to the client side indicating a willingness to do "remote echo" of characters. The pseudo terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*(7)). Aside from this initial setup, the only mode changes *telnetd* will carry out are those required for echoing characters at the client side of the connection.

The following options are recognized:

BINARY
Controls transmission of binary data.

ECHO
Controls echoing.

SGA Suppress go ahead.

STATUS
Status of options.

TM Timing Mark.

EXOPL
Extended Options List.

SEE ALSO

telnet(1N).

NAME

test – condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

Test evaluates the expression ***expr*** and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct ***expr***:

–r *file*	true if *file* exists and is readable.
–w *file*	true if *file* exists and is writable.
–x *file*	true if *file* exists and is executable.
–f *file*	true if *file* exists and is a regular file.
–d *file*	true if *file* exists and is a directory.
–c *file*	true if *file* exists and is a character special file.
–b *file*	true if *file* exists and is a block special file.
–p *file*	true if *file* exists and is a named pipe (fifo).
–u *file*	true if *file* exists and its set-user-ID bit is set.
–g *file*	true if *file* exists and its set-group-ID bit is set.
–k *file*	true if *file* exists and its sticky bit is set.
–s *file*	true if *file* exists and has a size greater than zero.
–t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
–z *s1*	true if the length of string *s1* is zero.
–n *s1*	true if the length of the string *s1* is non-zero.
s1 = *s2*	true if strings *s1* and *s2* are identical.
s1 != *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 **–eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **–ne**, **–gt**, **–ge**, **–lt**, and **–le** may be used in place of **–eq**.

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

expr(1), 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.

Some UNIX systems do not recognize the second form of the command.

NAME

tftp – user interface to the DARPA TFTP protocol

SYNOPSIS

tftp [host [port]]

DESCRIPTION

Tftp is the user interface to the ARPANET standard Trivial File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *tftp* is to communicate may be specified on the command line. If this is done, *tftp* will immediately attempt to establish a connection to a TFTP server on that host. Otherwise, *tftp* will enter its command interpreter and await instructions from the user. When *tftp* is awaiting commands from the user, the prompt

tftp>

appears. The following commands are recognized by *tftp*:

connect
connect to remote *tftp*

mode set file to transfer mode

put send file

get receive file

quit exit *tftp*

verbose
toggle verbose mode

trace toggle packet tracing

status show current status

rexmt set total retransmission

? print help information

timeout
set total retransmission timeout

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftp* will allow only publicly readable files to be accessed.

FILES

/etc/hosts

SEE ALSO

tftpd(1NM).

WARNINGS

Due to the unreliability of the transport protocol (UDP) and the scarcity of TFTP implementations, it is uncertain whether it really works.

The search permissions of the directories leading to the files accessed are not checked.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

/etc/tftpd [**–d**] [port]

DESCRIPTION

Tftpd is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(4). This port number may be overridden (for debugging purposes) by specifying a port number on the command line. If the **–d** option is specified, each socket created by *tftpd* will have debugging enabled (see SO_DEBUG in *socket*(2N)).

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling tftp service.

SEE ALSO

tftp(1N), netman(1N).

BUGS

This server is known only to be self consistent (i.e. it operates with the user TFTP program, *tftp*(1N)). Due to the unreliability of the transport protocol (UDP) and the scarcity of TFTP implementations, it is uncertain whether it really works.

The search permissions of the directories leading to the files accessed are not checked.

NAME

tic – terminfo compiler

SYNOPSIS

tic [**–v**[*n*]] file ...

DESCRIPTION

Tic translates terminfo files from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo**.

The **–v** (verbose) option causes *tic* to output trace information showing its progress. If the optional integer is appended, the level of verbosity can be increased.

Tic compiles all terminfo descriptions in the given files. When a **use=** field is discovered, *tic* searches first the current file, then the master file, which is "./terminfo.src".

If the environment variable TERMINFO is set, the results are placed there instead of **/usr/lib/terminfo**.

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/*/* compiled terminal capability data base

SEE ALSO

curses(3X), terminfo(4).

BUGS

Instead of searching **./terminfo.src**, it should check for an existing compiled entry.

NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

timex(1), times(2).

NAME

timex - time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

Options are:

-p List process accounting records for *command* and all its children. Suboptions **f, h, k, m, r,** and **t** modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.

-o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

-s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

SEE ALSO

acctcom(1), sar(1).

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

EXAMPLES

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex -opskmt sh
        session commands
EOT
```

NAME

toc – graphical table of contents routines

SYNOPSIS

dtoc [directory]
ttoc mm-file
vtoc [**–cdhnimsvn**] [TTOC 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

ttoc Output is the table of contents generated by the .TC macro of *mm*(1) translated to TTOC format. The input is assumed to be an *mm* file that uses the .H family of macros for section headers. If no *file* is given, the standard input is assumed.

vtoc *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:
n, 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 (**\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).

d Connect the boxes with diagonal lines.

h*n* Horizontal interbox space is *n*% of box width.

i Suppress the box *id*.

m Suppress the box *mark*.

s Do not compact boxes horizontally.

v*n* Vertical interbox space is *n*% of box height.

SEE ALSO
graphics(1G), gps(4).

NAME

touch – update access and modification times of a file

SYNOPSIS

touch [**-amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used. The **-a** and **-m** options cause touch to update only the access or modification times respectively (default is **-am**). The **-c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1), utime(2).

NAME

tplot – graphics filters

SYNOPSIS

tplot [**–T**terminal [**–e** raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(4)) 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 **$TERM** (see *environ*(5)) is used. Known *terminal*s 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(4), term(5).

NAME

tput – query terminfo database

SYNOPSIS

tput [**–T**type] capname

DESCRIPTION

Tput uses the *terminfo(4)* database to make terminal-dependent capabilities and information available to the shell. *Tput* outputs a string if the attribute (**cap**ability **name**) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, tput simply sets the exit code (0 for TRUE, 1 for FALSE), and does no output.

–T*type*	indicates the type of terminal. Normally this flag is unnecessary, as the default is taken from the environment variable **$TERM.**
Capname	indicates the attribute from the *terminfo* database. See *terminfo(4).*

EXAMPLES

tput clear	Echo clear-screen sequence for the current terminal.
tput cols	Print the number of columns for the current terminal.
tput –T450 cols	Print the number of columns for the 450 terminal.
bold=‘tput smso‘	Set shell variable "bold" to stand-out mode sequence for current terminal. This might be followed by a prompt: **echo "${bold}Please type in your name: \c"**
tput hc	Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/etc/term/?/*	Terminal descriptor files
/usr/include/term.h	Definition files
/usr/include/curses.h	

DIAGNOSTICS

Tput prints error messages and returns the following error codes on error:

–1	Usage error.
–2	Bad terminal type.
–3	Bad capname.

In addition, if a capname is requested for a terminal that has no value for that capname (e.g., **tput -T450 lines**), **-1** is printed.

SEE ALSO

stty(1), terminfo(4).

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 considered octal; otherwise, n is taken to be decimal. A zero or missing n is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

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(5).

BUGS

Will not handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

troff – typeset text

SYNOPSIS

troff [options] [files]

DESCRIPTION

Troff formats text contained in *files* (standard input by default) for a Wang Laboratories, Inc., C/A/T phototypesetter. Its 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:

–o*list*	Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end. (See *BUGS* below.)
–n*N*	Number first generated page *N*.
–s*N*	Stop every *N* pages. *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.
–r*aN*	Set register *a* (which must have a one-character name) to *N*.
–i	Read standard input after *files* are exhausted.
–q	Invoke the simultaneous input-output mode of the **.rd** request.
–z	Print only messages generated by **.tm** (terminal message) requests.
–m*name*	Prepend to the input *files* the non-compacted (ASCII text) macro file **/usr/lib/tmac/tmac.***name*.
–c*name*	Prepend to the input *files* the compacted macro files **/usr/lib/macros/cmp.**[**nt**].[**dt**].*name* and **/usr/lib/macros/ucmp.**[**nt**].*name*.
–k*name*	Compact the macros used in this invocation of *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).
–t	Direct output to the standard output instead of the phototypesetter.
–f	Refrain from feeding out paper and stopping phototypesetter at the end of the run.

-w Wait until phototypesetter is available, if it is currently busy.

-b Report whether the phototypesetter is busy or available. No text processing is done.

-a Send a printable ASCII approximation of the results to the standard output.

-p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

-g Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output (this option is not usable on most systems). This option is not compatible with the **-s** option; furthermore, when this option is invoked, all **.fp** (font position) requests (if any) in the *troff* input must come before the first break, and *no* **.tl** requests may come before the first break.

-T*name* Use font-width tables for device *name* (the font tables are found in **/usr/lib/font/***name***/***). Currently, no *name*s are supported.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta$#	temporary file
/usr/lib/tmac/tmac.*	standard macro files and pointers
/usr/lib/macros/*	standard macro files
/usr/lib/font/*	font width tables for *troff*

SEE ALSO

NROFF/TROFF User's Manual and *A TROFF Tutorial* in the *UNIX System Document Processing Guide*.
cw(1), eqn(1), mmt(1), nroff(1), tbl(1), tc(1), mm(5), mv(5).

BUGS

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 *troff* generates may be off by one day from your idea of what the date is.
When *troff* is used with the **-o***list* option inside a pipeline (e.g., with one or more of *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 *list*.

NAME

trpt – print protocol trace

SYNOPSIS

trpt [**-a**] [**-s**] [**-t**] [**-j**] [**-p** hex-address] [system [core]]

DESCRIPTION

Trpt interrogates the buffer of TCP trace records created when a socket is marked for debugging (see *setsockopt*(2N)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

-s in addition to the normal output, print a detailed description of the packet sequencing information,

-t in addition to the normal output, print the values for all timers at each point in the trace,

-j just give a list of the protocol control block addresses for which there are trace records,

-p show only trace records associated with the protocol control block who's address follows,

-a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **-A** option to *netstat*(1N). Then run *trpt* with the **-p** option, supplying the associated protocol control block addresses. If there are many sockets using the debugging option, the **-j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

CONFIGURATION

To use *trpt*, your kernel must be configured for network debugging. See *config*(1M) and *master*(4). The **tcptrace** parameter must be non-zero for tracing to occur.

FILES

/unix
/dev/kmem

SEE ALSO

setsockopt(2N), netstat(1N)

DIAGNOSTICS

"no namelist" when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

WARNINGS

Trace will be incomplete for connections using intelligent network controllers.

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tset – set terminal, terminal inteface, and terminal environment

SYNOPSIS

tset [options] [**–m** [*pseudotype*][*test speed*]:*type* ...] [*type*]

DESCRIPTION

Tset initializes your terminal. Its primary use is in login scripts (see *profile*(4)) to set terminal options, terminal interface options, and environment variables. Its secondary use is to restore the terminal interface and terminal after an editor or other video program has crashed.

To restore the terminal interface and terminal, just type "tset". It may be necessary to end the command with Control-J or the NEXT key instead of the RETURN key.

To set up login initialization, construct a command with the options and arguments you need and place it in your login script.

An argument indicates a terminal type to use in place of the **TERM** environment variable. If the argument begins with a question mark, *tset* prompts you for a terminal type; if you enter a blank line, you get the type specified by the argument. Terminal type arguments (in conjunction with the – or **–s** options) are useful at installations where none of the terminals are permanently connected to the host.

Tset accepts the following options.

– Print the terminal type. This is useful for setting the **TERM** environment variable in the *.profile* file:

```
export TERM
TERM=`tset – ´?adm3a´`
```

–s Print commands that will set the **TERM** and **TERMCAP** environment variables. The value for **TERMCAP** contains a description of the terminal; this makes it unnecessary for programs to read the terminal capability file each time they start up. For example:

```
eval `tset –s ´?adm3a´`
```

Tset uses the **SHELL** environment variable to decide the kind of commands to print.

-S Prints values for **TERM** and **TERMCAP** Useful only in *.login*; if you use the values to set a shell variable, you get a two-element array.

-e*c* Set the erase character to *c*. Indicate a control character with a ^. If *c* is missing, *tset* uses the value of your backspace key; this is usually control-H. You also get control-H if your terminal lacks a backspace key.

-k*c* Set the kill character to *c*. Indicate a control character with a ^. If *c* is missing *tset* uses control-X.

-I Don't initialize the terminal.

-Q Don't remind user of erase and kill values.

-m*pseudotype test speed:type*

Use one or more **-m** options in place of a type argument when you want *tset* to figure out your terminal type for you. *Pseudotype* should be a type that your installation has reserved for a class of "soft" connections, such as **dialup**, **arpanet**, or **plugboard**. A missing pseudotype means "any type.". *Test* and *speed* indicate a class of baud rates. *Test* is = or @ for "equals"; < for "less than"; > for "greater than"; or !=, !@, !<, or !> for negations. A missing speed indication means "all speeds." *Type* is the type to assume if the pseudotype and terminal speeds match. *Type* can begin with a question mark to indicate a user query. Thus,

tset - -m 'dialup@300:trs80' -m 'dialup:t0'

prints "trs80" if **TERM** is "dialup" and the baud rate is 300; "t0" if **TERM** is "dialup" and the baud rate isn't 300; and the value of TERM otherwise.

FILES

/etc/ttytype	type wired to each port
/etc/termcap	terminal capability database

SEE ALSO

sh(1), stty(1), profile(4), ttytype(4), termcap(4), environ(5).

DIAGNOSTICS

Nonzero return status if it could not process all options and user input. This is useful to confirm that user entered known terminal type: see *profile*(4) for an example.

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty – get the name of the terminal

SYNOPSIS

tty [**–s**]

DESCRIPTION

Tty prints the path name of the user's terminal. The **–s** option inhibits printing of the terminal path name, allowing one to test just the exit code.

EXIT CODES

2	if invalid options were specified,
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

ul – do underlining

SYNOPSIS

ul [**–i**] [**–t** *terminal*] [*name* ...]

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurances of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. The **–t** option overrides the terminal kind specified in the environment. The file */etc/termcap* is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

The **–i** option causes *ul* to indicate underlining onto by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

AUTHOR

Mark Horton wrote *ul*. The **–i** option was originally a option of the editor *ex*(1), then an *iul* command.

BUGS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

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 CTIX system

SYNOPSIS

uname [**–snrvma**]

DESCRIPTION

Uname prints the name of the CTIX system on the standard output file. It is mainly useful to determine which 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.

–m print the machine hardware name.

–a print all the above information.

Arguments not recognized default the command to the **–s** option.

SEE ALSO

uname(2).

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

unget [**–r**SID] [**–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.

–r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

–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), help(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**–udc** [+n] [–n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **–u** flag is used, just the lines that are not repeated in the original file are output. The **–d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **–u** and **–d** mode outputs.

The **–c** option supersedes **–u** and **–d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

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 Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME

update – provide disk synchronization

SYNOPSIS

update [*s*]

DESCRIPTION

Update implements regular synchronization. Run it only once, in background; normally there is an *update* command in **/etc/rc**.

Update executes an infinite loop with two actions:

- A *sleep* for *s* seconds (30 seconds default).
- A *sync* system call. This feature calls for less overhead than the practice of having the *cron* command run *sync*.

FILES

/dev/dsk/* – disk interfaces

SEE ALSO

ioctl(2), sleep(3), fp(7).

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/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.

–d*directory* Clean *directory* instead of the spool directory. If *directory* is not a valid spool directory it cannot contain "work files" i.e., files whose names start with "C.". These files have special meaning to **uuclean** pertaining to **uucp** job statistics.

–p*pre* 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.

–n*time* Files whose age is more than *time* hours will be deleted if the prefix test is satisfied. (default time is 72 hours)

–w*file* The default action for *uuclean* is to remove files which are older than a specified time (see **–n** option). The **–w** option is used to find those files older than *time* hours, however, the files are not deleted. If the argument *file* is present the warning is placed in *file*, otherwise, the warnings will go to the standard output.

–s*sys* Only files destined for system *sys* are examined. Up to 10 **–s** arguments may be specified.

–m*file* The **–m** option sends mail to the owner of the file when it is deleted. If a *file* is specified then an entry is placed in *file*.

This program is typically started by *cron*(1M).

FILES

/usr/lib/uucp directory with commands used by *uuclean* internally

/usr/spool/uucp spool directory

SEE ALSO

cron(1M), uucp(1C), uux(1C).

NAME

uucp, uulog, uuname – CTIX system to CTIX system copy

SYNOPSIS

uucp [options] source-files destination-file

uulog [options]

uuname [**–l**] [**–v**]

DESCRIPTION

Uucp.

Uucp copies files named by the ***source-file*** arguments to the ***destination-file*** argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where ***system-name*** is taken from a list of system names which *uucp* knows about. The ***system-name*** may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, and only to a destination in PUBDIR (see below). Care should be taken to insure that intermediate nodes in the route are willing to foward information.

The shell metacharacters **?**, ***** and **[. . .]** appearing in ***path-name*** will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by ~***user*** where ***user*** is a login name on the specified system and is replaced by that user's login directory;

(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 PUBDIR;

(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*file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

-n*user*
Notify *user* on the remote system that a file was sent.

-e*sys* 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**.)

-r Queue job but do not start the file transfer process. By default a file transfer process is started each time *uucp* is invoked.

-j Control writing of the *uucp* job number to standard output (see below).

Uucp associates a job number with each request. This job number can be used by *uustat* to obtain status or terminate the job.

The environment variable **JOBNO** and the **-j** option are used to control the listing of the *uucp* job number on standard output. If the environment variable **JOBNO** is undefined or set to **OFF**, the job number will not be listed (default). If *uucp* is then invoked with the **-j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time uucp is invoked. In this case, the **-j** option will supress output of the job number.

Uulog
Uulog queries a summary log of *uucp* and *uux*(1C) transactions in the file **/usr/spool/uucp/LOGFILE.**

The options cause *uulog* to print logging information:

-s*sys* Print information about work involving system *sys*. If *sys* is not specified, then logging information for all systems will be printed.

-u*user*
Print information about work done for the specified, *user*. If *user* is not specified then logging information for all users will be printed.

Uuname.
Uuname lists the uucp names of known systems. The **-l** option returns the local system name. The **-v** option will print additional information about each system. A description will be printed for each system that has a line of information in **/usr/lib/uucp/ADMIN**. The format of ADMIN is: *sysname* tab *description* tab.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory for receiving and sending (PUBDIR)
/usr/lib/uucp/*	other data and program files

SEE ALSO
mail(1), uux(1C), chmod(2).

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 ~).

NOTES
In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

Uucp will not generate a job number for a strictly local transaction.

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.

The **-m** option will not work if all transactions are local or if **uucp** is executed remotely via the **-e** option.
The **-n** option will function only when the source and destination are not on the same machine.
Only the first six characters of a *system-name* are significant. Any excess characters are ignored.

NAME

uucpd – network uucp server

SYNOPSIS

/etc/uucpd

DESCRIPTION

Uucpd is the network server for CTIX network file transfer using the ***uucp*** user interface and protocols. It is similar to the ***rshd*** (1NM) server, except:

1) The remote socket need not be privileged, and

2) The shell invoked must be **/usr/lib/uucp/uucico.**

A network ***uucp*** connection is indicated with the **INET** keyword in **/usr/lib/uucp/L.sys**. *Uucpd* is normally executed by the startup file, **/etc/rc**.

SEE ALSO

rshd(1N), uucp(1C).

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [options]

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:

–j*jobn* Report the status of the *uucp* request *jobn*. If **all** is used for *jobn*, the status of all *uucp* requests is reported. An argument must be supplied otherwise the usage message will be printed and the request will fail.

–k*jobn* Kill the *uucp* request whose job number is *jobn*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

–r*jobn* Rejuvenate *jobn*. That is, *jobn* is touched so that its modification time is set to the current time. This prevents *uuclean* from deleting the job until the jobs modification time reaches the limit imposed by *uuclean*.

–c*hour* Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user **uucp** or the super-user.

–u*user* Report the status of all *uucp* requests issued by *user*.

–s*sys* Report the status of all *uucp* requests which communicate with remote system *sys*.

–o*hour* Report the status of all *uucp* requests which are older than *hour* hours.

–y*hour* Report the status of all *uucp* requests which are younger than *hour* hours.

–m*mch* Report the status of accessibility of machine *mch*. If *mch* is specified as **all**, then the status of all machines known to the local *uucp* are provided.

–M*mch* This is the same as the *–m* option except that two times are printed. The time that the last status was obtained and the time that the last successful transfer to that system occurred.

–O Report the *uucp* status using the octal status codes listed below. If this option is not specified, the verbose description is printed

with each *uucp* request.

-q List the number of jobs and other control files queued for each machine and the time of the oldest and youngest file queued for each machine. If a lock file exists for that system, its date of creation is listed.

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**, **-r**, can be used with the rest of the other options.

For example, the command:

uustat -uhdc -smhtsa -y72

will print the 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:

status

job-number user remote-system command-time status-time

where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
000001	the copy failed, but the reason cannot be determined
000002	permission to access local file is denied
000004	permission to access remote file is denied
000010	bad *uucp* command is generated
000020	remote system cannot create temporary file
000040	cannot copy to remote directory
000100	cannot copy to local directory
000200	local system cannot create temporary file
000400	cannot execute *uucp*
001000	copy (partially) succeeded
002000	copy finished, job deleted
004000	job is queued
010000	job killed (incomplete)
020000	job killed (complete)

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).

NAME

uusub – monitor uucp network

SYNOPSIS

/usr/lib/uucp/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:

–a*sys*	Add ***sys*** to the subnetwork.
–d*sys*	Delete ***sys*** from the subnetwork.
–l	Report the statistics on connections.
–r	Report the statistics on traffic amount.
–f	Flush the connection statistics.
–u*hr*	Gather the traffic statistics over the past *hr* hours.
–c*sys*	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, and ***#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 **–u***hr* 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 CTIX-to-CTIX system 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 *uuname*). *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

NOTES

In order to send files that begin with a dot (e.g., .profile) the files must be qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

SEE ALSO

mail(1), uuclean(1M), uucp(1C), uustat(1C), uux(1C).

NAME

uux – CTIX-to-CTIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail*(1)) via *uux*.

The *command-string* is made up of one or more arguments that look like a Shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(1) a full path name;

(2) a path name preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

As an example, the command

uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff"

will get the **f1** 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. Executable commands are listed in **/usr/lib/uucp/L.cmds** on the

remote system. The format of the **L.cmds** file is:

cmd,machine1,machine2,...

If no machines are specified, then any machine can execute **cmd**. If machines are specified, only the listed machines can execute **cmd**. If the desired command is not listed in **L.sys** then no machine can execute that command.

Redirection of standard input and output is usually restricted to files in PUBDIR. Directories into which redirection is allowed must be specified in **/usr/lib/uucp/USERFILE** by the system administrator.

The following *options* are interpreted by ***uux***:

- The standard input to ***uux*** is made the standard input to the ***command-string***.

-n Send no notification to user.

-m*file* Report status of the transfer in *file*. If *file* is omitted, send mail to the requester when the copy is completed.

-j Control writing of the ***uucp*** job number to standard output.

Uux associates a job number with each request. This job number can be used by ***uustat*** to obtain status or terminate the job.

The environment variable **JOBNO** and the **-j** option are used to control the listing of the ***uux*** job number on standard output. If the environment variable **JOBNO** is undefined of set to **OFF**, the job number will not be listed (default). If ***uuco*** is then invoked with the **-j** option, the job number will be listed. If the environment variable **JOBNO** is set to **ON** and is exported, a job number will be written to standard output each time ***uux*** is invoked. In this case, the **-j** option will suppress output of the job number.

FILES

/usr/spool/uucp	spool directory
/usr/spool/uucppublic	public directory (PUBDIR)
/usr/lib/uucp/*	other data and programs

SEE ALSO

mail(1), uuclean(1M), uucp(1C).

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.
Only the first six characters of the *system-name* are significant. Any excess characters are ignored.

NAME

val - validate SCCS file

SYNOPSIS

val -

val [**-s**] [**-r**SID] [**-m**name] [**-y**type] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

Val has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

-s The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

-r*SID* The argument value *SID* (*S*CCS *ID*entification String) is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e. g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

-m*name* The argument value *name* is compared with the SCCS %M% keyword in *file*.

-y*type* The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **-y** mismatch;
bit 7 = %M%, **-m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned – a logical OR of the codes generated for each command line and file processed.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc – version control

SYNOPSIS

vc [**–a**] [**–t**] [**–c**char] [**–s**] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the **–t** keyletter (see below). The default control character is colon (**:**), except as modified by the **–c** keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The **–a** keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

–a Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.

-t	All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.
-c*char*	Specifies a control character to be used in place of :.
-s	Silences warning messages (not 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.

SEE ALSO
ed(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 – normal
1 – any error

NAME

vi – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**–t** *tag*] [**–r** *file*] [**–l**] [**–w***n*] [**–R**] [+*command*] name ...

view [**-t** *tag*] [**–r** *file*] [**–l**] [**–w***n*] [**–R**] [+*command*] name ...

vedit [**–t** *tag*] [**–r** *file*] [**–l**] [**–w***n*] [**–R**] [+*command*] name ...

DESCRIPTION

Vi (visual) is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

INVOCATION

The following invocation options are interpreted by *vi*:

–t *tag*	Edit the file containing the *tag* and position the editor at its definition.
–r*file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
–l	LISP mode; indents appropriately for lisp code, the () {} [[and]] commands in *vi* and *open* are modified to have meaning for *lisp* .
–w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
–R	Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.
+*command*	The specified *ex* command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice**

flags are set. These defaults make it easier to get started learning the editor.

VI MODES

Command	Normal and initial mode. Other modes return to command mode upon completion. ESC (escape; GO on Convergent Technologies terminals) is used to cancel a partial command.
Input	Entered by **a i A I o O c C s S R**. Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.
Last line	Reading input for **:** **/** **?** or **!**; terminate with CR to execute, interrupt to cancel.

COMMAND SUMMARY

The following sequences represent special keys:

ESC	Escape key. GO on Convergent Technologies terminals.
^*x*	Control key: hold down the **CTRL** key (**CODE** on Convergent Technologies terminals) and press *x*.
CR	**RETURN** or **CARRIAGE RETURN** key.
↑	Circumflex (^). On teletypewriter-style terminals, usually an up arrow (↑).

Sample commands

← **↓** **↑** **→**	arrow keys move the cursor
h j k l	same as arrow keys
i*text*ESC	insert *text*
cw*new*ESC	change word to *new*
ea*s*ESC	pluralize word
x	delete a character
dw	delete a word
dd	delete a line
3dd	... 3 lines
u	undo previous change
ZZ	exit vi, saving changes
:q!CR	quit, discarding changes
/*text*CR	search for *text*
^U ^D	scroll up or down
:*ex cmd*CR	any ex or ed command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**
scroll amount	**^D ^U**
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
^?	(delete or rubout) interrupts
^L	reprint screen if **^?** scrambles it
^R	reprint screen if ^L is → key

File manipulation

:wCR	write back changes
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file
^^	synonym for **:e #**
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:sh CR	run shell, then return
:!*cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	to tag file entry *tag*
^]	**:ta**, following word is *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a **CR**.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/*pat*	next line matching *pat*
?*pat*	prev line matching *pat*
n	repeat last / or ?
N	reverse last / or ?
/*pat***/+***n*	nth line after *pat*
?*pat***?−***n*	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence

)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z–CR	... at bottom
z .CR	... at center
/*pat*/z–CR	*pat* line at bottom
z*n* .CR	use *n* line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

``	move cursor to previous context
''	... at first non-white in line
m*x*	mark current position with letter *x*
`*x*	move cursor to mark *x*
'*x*	... at first non-white in line

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
–	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character positioning

^	first non white
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	f backward
t*x*	up to *x* forward
T*x*	back up to *x*
;	repeat last f F t or T
,	inverse of ;
|	to specified column
%	find matching ({) or }

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back **W**
E	to end of **W**

Commands for LISP Mode

)	Forward s-expression
}	... but do not stop at atoms
(	Back s-expression
{	... but do not stop at atoms

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as **^H**
kill	your kill, erase input this line
****	quotes **^H**, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before cursor
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g. **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift

>	right shift
!	filter through command
=	indent for LISP

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	... before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"*x***p**	put from buffer *x*
"*x***y**	yank to buffer *x*
"*x***d**	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d* **p**	retrieve *d*'th last delete

AUTHOR

Vi and *ex* were developed by The University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

ex (1).
CTIX Programmer's Guide.

CAVEATS AND BUGS

Software tabs using **^T** work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

There should be an interactive *help* facility and a tutorial suited for beginners.

Due to export restrictions, encryption features are not available outside the United States.

```
                /* Address of the board; in MightyFrame I/O space */
                uint    address;

                /* Amount of address space taken up by the board */
                uint    length;

                /* Pointer to an optional initialization function */
                int     (*initfp)();
        } slots[VME_SLOTS];

        /* Reserve the rest for controller code */
        char    drivers[7860];
};

#define VMEE_DIAG       0  /* Diag has cleared/set EEPROM */
#define VMEE_LOADED     1  /* unix has loaded driver information */

#define VMET_CMC        1  /* CMC Ethernet controller */
#define VMET_V3200      2  /* Interphase SMD controller */
```

FILES

/dev/vme/a16	64K bytes of short address space
/dev/vme/a24	32M bytes of standard address space
/dev/vme/a32l	low 2 gigabytes of extended address space
/dev/vme/a32h	high 2 gigabytes of extended address space
/dev/vme/eeprom	8K VME interface EEPROM

SEE ALSO

ldeeprom(1M), system(4), mem(7).
MightyFrame VME Expansion Manual.

NAME

volcopy, labelit – copy file systems with label checking

SYNOPSIS

/etc/volcopy [options] 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. *Options* are:

- **-a** invoke a verification sequence requiring a positive operator response instead of the standard 10-second delay before the copy is made,
- **-s** (default) invoke the DEL **if wrong** verification sequence.
- **-to** The output file is a disk section (also called slice or partition), but is to be treated like a tape.
- **-ti** The input file is a disk section, but is to be treated like a tape.

Other *options* are used only with tapes:

- **-bpi**density bits-per-inch (i.e., **800/1600/6250**)
- **-feet**size size of reel in feet (i.e., **1200/2400**),
- **-reel**num beginning reel number for a restarted copy,
- **-buf** use double buffered I/O.
- **-Q** Use **-bpi** and **-feet** values appropriate for quarter-inch tape cartridge.

If **-ti** or **-to** is specified, the "reel" capacity is simply the size of the disk section; the "reel" is assumed to be on a removable disk, such as a floppy.

For a true tape such as half-inch reel-to-reel or quarter-inch cartridge, capacity is derived from tape length and density. The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. 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 or more drives. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (e.g., *labelit*) and return to *volcopy* by exiting the new shell.

The *fsname* argument represents the mounted name (e.g.: **root**, **u1**, etc.) of the filsystem being copied.

The *special* should be the physical disk section or tape (e.g., **/dev/rdsk/c0d0s5**, **/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 six or fewer characters. *Volname* may be – to use the existing volume name.

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 device 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). The **–t** option puts tape headers on media other than tape.

FILES

/etc/log/filesave.log a record of file systems/volumes copied

EXAMPLE

The following command backs up the root file system to a tape:

```
volcopy –a root /dev/rdsk/c0d0s1 d0 /dev/rmt0 d0 epoch1
```

SEE ALSO

sh(1), fs(4).

WARNINGS

Labelit applied to a mounted file system will appear to succeed, but the next reboot or *umount* will remove the label.

BUGS

Only device names beginning **/dev/rmt** are automatically treated as tapes. Tape record sizes are determined both by density and by drive type. Records are 5,120 bytes long at 800 and 1600 bits-per-inch, and 25,600 bytes long at 6250 bits-per-inch.

NAME

vt – virtual terminal

DESCRIPTION

A virtual terminal provides a terminal-like communication channel between two processes. Each virtual terminal consists of two devices: a slave device, whose name is of the form **/dev/ttyp***xx*, where *xx* is the virtual terminal number; and a master device, whose name is of the form **/dev/vt***xx*, where *xx* is the virtual terminal number. The slave device responds to system calls just like a real terminal (see *termio*(7)) so that it can control interactive programs such as *vi*. But instead of doing actual input/output, reads and writes on the slave device are written and read on the corresponding master device by another process. A typical use of a virtual terminal is to put a network server on the master device and login program on the slave.

The number of virtual terminals must be configured. See *config*(1M).

The process on the master device can exercise flow control on the slave device, much as a real terminal would use XON/XOFF to exercise flow control on a terminal device. The parameterless *ioctl*(2) TIOCSTOP stops output to the slave device as if with an XOFF character; the parameterless *ioctl*(2) TIOCSTART restarts output, as if with an XON character.

FILES

/dev/ttyp??	slave devices
/dev/vt??	master devices

SEE ALSO

config(1M), ttyname(3C), termio(7).

NAME

window – window management primitives

SYNOPSIS

#include <sys/window.h>

DESCRIPTION

Window managment (*wm*(1)) provides a superset of windowless terminal features. This entry describes terminal file features special to window management. Window management features are designed not to interfere with programs that do not know about window management. Such design includes simple extensions to the UNIX System's standard concepts of file descriptor and control terminal.

- Each terminal file descriptor has an associated window number, a small positive integer that identifies a window. A window number is the most primitive way to refer to a window, and should not be confused with the window ID used by window management subroutines. A new window gets the smallest window number not already in use. Closing a window frees its number for possible assignment to a later window. Output and control calls on the file descriptor apply only to the descriptor's window; input calls succeed only when the window is active.

  A file descriptor created by a *dup*(2) or inherited across a *fork*(2) inherits the original descriptor's window number. All the file descriptors in such a chain of inheritance, provided they belong to processes in the same process group, are affected when *ioctl* changes the window number of any of them.

- When a process group's control terminal is under window managment, the process group is actually controlled by a particular window. Such can have more than one process group, each controlled by a different window. Keyboard-generated signals (*interrupt* and *quit*) go to the process group controlled by the active window.

When the user creates a new window by using the SPLIT key, the window manager forks a process for that window. The new process inherits file descriptors for standard input (0), standard output (1), and standard error (2) that are associated with the new window. The

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), wait(2).

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot 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 must be super-user to override any protections the users may have invoked (see *mesg*(1)).

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 [**–lwc**] [names]

DESCRIPTION

Wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **–lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

what – identify SCCS files

SYNOPSIS

what [**–s**] 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 follows until the first ", >, new-line, \, or null character. For example, if the C program in file **f.c** contains

char ident[] = " @(#)identification information ";

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

what f.c f.o a.out

will print

f.c:
identification information

f.o:
identification information

a.out:
identification information

What is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

s Quit after finding the first occurrence of pattern in each file.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Exit status if - if any matches are found, otherwise 1. Use *help*(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

who – who is on the system

SYNOPSIS

who [**–uTHlpdbrtasq**] [file]

who am i

who am I

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current CTIX system user. It examines the **/etc/utmp** file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

Who with the **am i** or **am I** option identifies the invoking user.

Except for the default **–s** option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

–u This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

–T This option is the same as the **–u** option, except that the *state* of the terminal line is printed. The

state describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a − appears if it is not. **Root** can write to all lines having a + or a − in the *state* field. If a bad line is encountered, a **?** is printed.

−l This option lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

−H This option will print column headings above the regular output.

−q This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

−p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

−d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(2)), of the dead process. This can be useful in determining why a process terminated.

−b This option indicates the time and date of the last reboot.

−r This option indicates the current *run-level* of the *init* process. Following the run-level and date information are three fields which indicate the current state, the number of times that state was previously entered, and the previous state.

−t This option indicates the last change to the system clock (via the *date*(1) command) by **root**. See *su*(1).

−a This option processes **/etc/utmp** or the named *file* with all options turned on.

−s This option is the default and lists only the *name*, *line* and *time* fields.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

init(1M) date(1), login(1), mesg(1), su(1), wait(2), inittab(4), utmp(4).

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.

FILES

etc/passwd

SEE ALSO

ps(1), who(1).

NAME

wm – window management

SYNOPSIS

exec /usr/local/bin/wm [**-k**][**-s**][**--**] [passparam]

DESCRIPTION

Wm is the window manager. It provides services to application programs running under its control and to users using terminals under its control. The window manager can divide the terminal screen into windows that the user can use like separate terminals. Other services include placement, size, scrolling, and synchronization of windows. *Wm* requires a Convergent Technologies Programmable Terminal or Graphics Terminal on a cluster line. The window manager must be running for the window management library functions to work.

The window manager is normally executed in place of the user's login shell by the **exec** command in **/etc/profile** or the user's own **.profile**. The window manager then executes the user's shell each time the user splits a window. The SHELL environment variable (normally set by *login*(1M)*to* **/bin/sh**) provides the full pathname of the initial program run in the windows.

When *wm* starts, the user sees four regions on the screen, going from top to bottom:

- A *message line*. A single line, always at the top of the screen. It holds messages and prompts from application programs.
- A *tag line*. A single line, always above each window, which labels the particular application program or display that is active in the window.
- The *window*. The main display area used by programs. Text input and output to the shell or an application program goes here. The window is a window into a *virtual display*. An application program can use the virtual display as a 28-line screen, regardless of the size of the window. The virtual display is usually larger than the window. Normally the window manager automatically positions the window over the part of the virtual display that contains the cursor. If the user program moves the cursor to a part of the virtual display not in the window, the window manager scrolls the window until the cursor is visible again. The user can also scroll the display (see below).

- The *function key line*. A single line, always at the bottom of the screen, that labels the function keys for the currently active window.

Wm accepts user commands activated by the ACTION key; such commands are not seen by the user program. Use the ACTION key like the CODE or SHIFT keys: hold down the ACTION key and press the other key used with it. Holding down the ACTION key changes the function key line to show how ACTION changes the meanings of the function keys.

Here are the valid *wm* user commands:

ACTION-F10 (SPLIT)

Split the active window, creating a new window. The new window and its tag line replace the bottom half of the window being split. Any program running in the old window is unaffected. The virtual display of the old window is unchanged, though less of it is visible. The user shell then starts up in the new window.

The new window is *active*; all other windows are *inactive*. Programs running in inactive windows continue to run, but input calls will not return until the user reactivates the window and types something. Keyboard input goes to the active window.

Each window, whether active on inactive, has its own message line, function key line, and cursor, but the terminal only displays them if they belong to the active window. (Application programs can also make the cursor invisible.) If an application program in a inactive window writes to the message line, the message is not visible until you make that window active again.

On Programmable Terminals the active window's tag line is displayed full intensity, with the other tag lines displayed half intensity. On Graphics Terminals the active window's tag line is displayed in bold, with the other tag lines displayed without bold.

When the SPLIT key creates a new window, *wm* automatically provides a program to run in the window. The program is a process group leader; the new process group is controlled by the new window and has terminal file descriptors associated with the new window. The program is a shell unless *wm* was run with the **-k** option.

When all processes in the process group die, *wm* automatically closes the window.

Programs can also have the window manager create windows, using *wmop*(3X). The window manager does not automatically provide programs for such windows.

The SPLIT key becomes inoperative if the terminal already displays its maximum number of windows or if a user program has disabled window splitting.

ACTION-F9 (BELOW)
The window below the active window becomes the active window with the old active window becoming inactive. The new active window takes over the message line and the function key line, and its cursor becomes visible.

ACTION-↓ is the same as ACTION-F9.

ACTION-F8 (ABOVE)
The window above the active window becomes the active window. ACTION-↑ is the same as ACTION-F8.

ACTION-*n*
Activate window *n*, where *n* is a number from 1 to 4. A window's number is assigned when it's first created, with a new window getting the lowest unused number. Unless erased by a user program, the window number is displayed on the left end of the tag line.

ACTION-F7 (SWAP ↓)
The active window and the window below it trade places.

ACTION-F6 (SWAP ↑)
The active window and the window above it trade places.

ACTION-F5 (SHRINK)
The active window decreases in size by 1 line. Ignored if the window is already 0 lines long (only the tag line visible).

ACTION-SHIFT-F5
The active window decreases in size by 4 lines. If the window is already less than 4 lines long, it becomes 0 lines long.

ACTION-CODE-F5
The active window becomes 0 lines long.

Shrinking the top window increases the size of the window below; shrinking any other window increases the size of the window above.

ACTION-F4 (GROW)
: The active window increases in size by 1 line. Ignored if the other windows are all 0 lines long.

ACTION-SHIFT-F4
: The active window increases in size by 4 lines. If the other windows don't have 4 lines to spare, the active window increases until all other windows are 0 lines long.

ACTION-F3 (MAX)
ACTION-CODE-F4
: The active window increases in size until all other windows are 0 lines long.

Growing the top window decreases the size of the window below; growing any other window window decreases the size of window below. If the window that would otherwise shrink is already 0 lines long, the next window shrinks. If all the windows below the second or third window are 0 lines long, space comes from the windows above.

ACTION-SCROLL UP
: The active window is scrolled up a line. Ignored if the window already shows the very bottom of the virtual display or if the cursor is on the window's top line.

ACTION-SCROLL DOWN
: The active window is scrolled down a line. Ignored if the window already shows the very top of the virtual display or if the cursor is on the window's bottom line.

Wm understands the following options:

-k Run ***keyprompt***(1) in the first window and in manually-created (SPLIT key) windows instead of the shell.

-s Disable the SPLIT key. The user cannot create new windows, but programs running under ***wm*** still can.

-c Run the CTIX Office Applications Interface in the first window instead of the shell; disable the SPLIT key (as with

-s above). The Office Applications Interface is a separate product and must have been previously installed for this option to work.

-- End of *wm* options. Subsequent parameters are passed to the shell, *keyprompt*, or the Office Applications Interface, even if they begin with a dash (-).

Parameters other than options are passed unchanged to programs executed by *wm*.

Wm uses or sets the following environment parameters:

TERM If already set, *wm* passes it unchanged to its own children. If not already set, *wm* has the terminal identify itself and sets **TERM** to **pt** or **gt** accordingly.

SHELL Name of the shell's executable file. If **-k** and **-c** aren't specified, **SHELL** is the initial program in the first window and in user-created (SPLIT windows. If **-k** or **-c** is specified, **SHELL** must still have a useful value, such as */bin/sh*.

KEYPROMPT
The name of *keyprompt's* executable file. If **KEYPROMPT** is not defined, **/usr/oa/keyprompt** is used.

CTIXOA The name of the Office Application Interface's executable file. If **CTIXOA** is not defined, **/usr/oa/ctixoa** is used.

TERMPARM If the user's terminal is a Graphics Terminal, *wm* reads the 32 bytes in the terminal's EAPROM, codes them in hexadecimal, and provides its children with those 64 digits in **TERMPARM**.

SEE ALSO
sh(1).

WARNING
If a program quickly outputs two things at the virtual display's top and bottom, the user can easily miss one of them. This normally is the fault of programs, originally designed for terminals without window features, that use the bottom line as a message line. Use the terminal message line instead.

BUGS

Write(1) messages appear only in the first window.
Some *csh*(1) features may not work with *wm*.

NAME

write – write to another user

SYNOPSIS

write user [line]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* **(tty**???**)** [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty000**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, in particular *nroff*(1) and *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character ! is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute !

SEE ALSO

mail(1), mesg(1), nroff(1), pr(1), sh(1), who(1).

DIAGNOSTICS

"*user not logged on*" if the person you are trying to *write* to is not logged on.

"*Permission denied*" if the person you are trying to *write* to denies that permission (with *mesg*).

"*Warning: cannot respond, set mesg -y*" if your terminal is set to *mesg n* and the recipient cannot respond to you.

"*Can no longer write to user*" if the recipient has denied permission (*mesg n*) after you had started writing.

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 quote. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the ***initial-arguments***, followed by some number of arguments read from standard input (Exception: see **–i** flag). Flags **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these flags are coded, the ***initial-arguments*** are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **–l** vs. **–n**), the last flag has precedence. *Flag* values are:

–l*number*	*Command* is executed for each non-empty ***number*** lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than ***number*** remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line.

If *number* is omitted, 1 is assumed. Option **-x** is forced.

-i*replstr*
: Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **-x** is also forced. **{ }** is assumed for *replstr* if not specified.

-n*number*
: Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **-x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

-t
: Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

-p
: Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**-t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

-x
: Causes *xargs* to terminate if any argument list would be greater than *size* characters; **-x** is forced

by the options **-i** and **-l**. When neither of the options **-i**, **-l**, or **-n** are coded, the total length of all arguments must be within the *size* limit.

-s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **-s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

-e*eofstr* *Eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if **-e** is not coded. The value **-e** with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical EOF string is encountered.

Xargs will terminate if either it receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **-1**.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1.  ls | xargs -p -1 ar r arch
2.  ls | xargs -p -l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO

sh(1).

DIAGNOSTICS

Self-explanatory.

NAME

xstr – extract and share strings in C programs

SYNOPSIS

xstr **–c** *source*

xstr

xstr *source*

DESCRIPTION

Xstr creates a version of a C program in which all strings are contained in a single external array, **xstr**. This optimizes the program in two ways:

- Redundant characters are removed from the object file. A string that is identical to a string earlier in the program is eliminated. A string that is a terminal substring of a longer string is also eliminated, but only if *xstr* sees the longer string first.
- The *xstr* array can be made read-only (shared), reducing space and swapping.

Compiling and linking a program with *xstr* requires three changes in the usual procedure:

1. Instead of compiling the source files, pass each source file to *xstr* with the **–c** option (see first synopsis above). This produces a file *x.c* which is compiled in place of *source*.

   X.c contains the same code as *source* but with each string replaced by an expression of the form (**&xstr**[*number*]), where *number* is the appropriate offset in **xstr**. *Xstr* also creates or updates the file *strings* in the current directory to include strings encountered in *source*.

   Source can be a –, indicating standard input. This is useful when the C preprocessor produces or suppresses strings. The command to use the preprocessor with *xstr* takes the form

   cc –E *source* | **xstr –c –**

2. Run *xstr* without parameters (second synopsis above). *Xstr* uses *strings* to create *xs.c*, a file that declares the **xstr** array. Compile **xs.c**.

3. Link the object file compiled from *xs.c* (normally called *xs.o*) together with all the object files produced in step 1.

Strings is only touched when a string is added or removed. Thus *make*(1) can speed things up by making *xs.o* dependent on *strings*.

If a program has a single source file, pass it to *xstr* without the **-c** option (third synopsis above). This creates *x.c* and *xs.c* without touching *strings*.

EXAMPLE

The following makefile uses *xstr* to produce a program from three source files: *main.c*, *uno.c*, and *omega.c*.

```
a.out:  main.o uno.o omega.o xs.o
        cc main.o uno.o omega.o xs.o

xs.o:   strings
        xstr
        cc -c xs.c

.c.o:
        cc -E $*.c | xstr -c -
        cc -c x.c
        mv x.o $*.o
```

FILES

strings	strings found in source
x.c	massaged C source
xs.c	definition of xstr array
/tmp/xs*	Temp file

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [**–vdlt**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **–v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **–d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **–l** flag is given, the code produced in **y.tab.c** will *not* contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **–t** option is used, this debugging code will be compiled by default. Independent of whether the **–t** option was used, the runtime debugging code is under the control of YYDEBUG, a pre-processor symbol. If YYDEBUG has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp,	

yacc.debug, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1), malloc(3X).
CTIX Programmer's Guide, Section 18.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.