AT&T System V.3 Administrator's Reference Manual

Version A 89-11-01
© Diab Data AB
089-9717-00

Diab Data AB
Box 2029
S-183 02 TÄBY
SWEDEN
☎ +46 8 638 94 00

DIAB DATA

1

NAME

intro - introduction to maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used along with those listed in Section 1 of the *User's Reference Manual* and Sections 1, 2, 3, 4, and 5 of the *Programmer's Reference Manual*. References of the form *name* (1), (2), (3), (4) and (5) refer to entries in the above manuals. References of the form *name* (1M), *name* (7) or *name* (8) refer to entries in this manual.

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) in the *User's Reference Manual.*

getopt(3C) in the *Programmer's Reference Manual.*

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

Regrettably, not all commands adhere to the aforementioned syntax.

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

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, "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 (i.e., UNIX System V) 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), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acctcom(1), acct(2), acct(4), utmp(4).

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

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).

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 -i <file2 >file1
```

SEE ALSO

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

NAME

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).

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).

CAVEAT

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

NAME

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 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 specifed 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/pacctincr** (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).

NAME

captoinfo - convert a termcap description into a terminfo description

SYNOPSIS

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

DESCRIPTION

captoinfo looks in *file* for *termcap* descriptions. For each one found, an equivalent *terminfo* (4) description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the *termcap* *tc* = field) will be reduced to the minimum superset before being output.

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

-v	print out tracing information on standard error as the program runs. Specifying additional **-v** options will cause more detailed information to be printed.
-V	print out the version of the program in use on standard error and exit.
-1	cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.
-w	change the output to *width* characters.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database

CAVEATS

Certain *termcap* defaults are assumed to be true. For example, the bell character *(terminfo bel)* is assumed to be ^G. The linefeed capability *(termcap nl)* is assumed to be the same for both *cursor_down* and *scroll_forward (terminfo cud1* and *ind,* respectively.) Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for *termcap* fields such as *cursor_position (termcap cm, terminfo cup)* will sometimes produce a string which, though technically correct, may not be optimal. In particular, the rarely used *termcap* operation **%n** will produce strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a *termcap* entry, a hold-over from an earlier version of the UNIX system, has been removed.

DIAGNOSTICS

tgetent failed with return code n (reason).

The termcap entry is not valid. In particular, check for an invalid 'tc=' entry.

unknown type given for the termcap code *cc*.

The termcap description had an entry for *cc* whose type was not boolean, numeric or string.

wrong type given for the boolean (numeric, string) termcap code *cc*.

The boolean *termcap* entry *cc* was entered as a numeric or string capability.

the boolean (numeric, string) termcap code *cc* **is not a valid name.**

An unknown *termcap* code was specified.

tgetent failed on TERM=term.

The terminal type specified could not be found in the *termcap* file.

TERM=term: cap *cc* **(info** *ii***) is NULL: REMOVED**

The *termcap* code was specified as a null string. The correct way to cancel an entry is with an '@', as in ':bs@:'. Giving a null string could cause incorrect assumptions to be made by the software which uses *termcap* or *terminfo*.

a function key for *cc* **was specified, but it already has the value** *vv*.

When parsing the **ko** capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

the unknown termcap name *cc* **was specified in the ko termcap capability.**

A key was specified in the **ko** capability which could not be handled.

the *vi* **character** *v* **(info** *ii***) has the value** *xx***, but ma gives** *n*.

The **ma** capability specified a function key with a value different from that specified in another setting of the same key.

the unknown *vi* **key** *v* **was specified in the ma termcap capability.**

A *vi* (1) key unknown to *captoinfo* was specified in the **ma** capability.

Warning: *termcap* **sg (***nn***) and** *termcap* **ug (***nn***) had different values.**

terminfo assumes that the **sg** (now **xmc**) and **ug** values were the same.

Warning: the string produced for *ii* may be inefficient.

The parameterized string being created should be rewritten by hand.

Null termname given.

The terminal type was null. This is given if the environment variable TERM is not set or is null.

cannot open *file* for reading.

The specified file could not be opened.

SEE ALSO

infocmp(1M), tic(1M).

curses (3X), terminfo(4) in the *Programmer's Reference Manual.*

NOTES

captoinfo should be used to convert *termcap* entries to *terminfo* (4) entries because the *termcap* database (from earlier versions of UNIX System V) may not be supplied in future releases.

NAME

chroot - change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

chroot causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the path names is changed for the command and any of its child processes to *newroot.* Furthermore, upon execution, the initial working directory is *newroot.*

Notice, however, that if you redirect the output of the command to a file:

```
chroot newroot command >x
```

will create the file **x** relative to the original root of the command, not the new one.

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

This command can be run only by the super-user.

SEE ALSO

cd(1) in the *D-NIX 5.3 Reference Manual.*

chroot(2) in the *Programmer's Reference Manual.*

BUGS

One should exercise extreme caution when referencing device files in the new root file system.

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 OLDobject 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), mk(8).

NAME

fuser - identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**-ku**] files | resources [**-**] [[**-ku**] **files** | **resources**]

DESCRIPTION

fuser outputs the process IDs of the processes that are using the *files* or remote *resources* specified as arguments. Each process ID is followed by a letter code, interpreted as follows: if the process is using the file as 1) its current directory, the code is **c** , 2) the parent of its current directory (only when the file is being used by the system), the code is **p,** or 3) its root directory, the code is **r.** For block special devices with mounted file systems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource (Remote File Sharing) are reported. (*fuser* cannot use the mount point of the remote resource; it must use the resource name.) For all other types of files (text files, executables, directories, devices, etc.) only the processes using that file are reported.

The following options may be used with *fuser*:

-u	the user login name, in parentheses, also follows the process ID.
-k	the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see *kill* (2)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

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.

You cannot list processes using a particular file from a remote resource mounted on your machine. You can only use the resource name as an argument.

SEE ALSO

mount(1M), ps(1) in the *D-NIX 5.3 Reference Manual.*

kill(2), signal(2) in the *Programmer's Reference Manual.*

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 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 solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to INVALID and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that 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), runacct(1M), ed(1), acct(2), acct(4), utmp(4).

NAME

helpadm - make changes to the Help Facility database

SYNOPSIS

/etc/helpadm

DESCRIPTION

The UNIX system Help Facility Administration command, *helpadm,* allows UNIX system administrators and command developers to define the content of the Help Facility database for specific commands and to monitor use of the Help Facility. The *helpadm* command can only be executed by login root, login bin, or a login that is a member of group bin.

The *helpadm* command prints a menu of 3 types of Help Facility data which can be modified, and 2 choices relating to monitoring use of the Help Facility. The five choices are:

- modify *startup* data
- add, modify, or delete a *glossary* term
- add, modify, or delete command data (description, options, examples, and keywords)
- prevent monitoring use of the Help Facility (login root and login bin only)
- permit monitoring use of the Help Facility (login root and login bin only)

The user may make one of the above choices by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

If one of the first three choices is chosen, then the user is prompted for additional information; specifically, which *startup* screen, *glossary* term definition, or command description is to be modified. The user may also be prompted for information to identify whether the changes to the database are additions, modifications, or deletions. If the user is modifying existing data or adding new data, then they are prompted to make the appropriate modifications/additions. If the user is deleting a *glossary* term or a command from the database, then they must respond affirmatively to the next query in order for the deletion to be done. In any case, before the user's changes are final, they must respond affirmatively when asked whether they are sure they want their requested database changes to be done.

By default, *helpadm* will put the user into *ed* (1) to make additions/modifications to database information. If the user wishes to be put into a different editor, then they should set the environment variable EDITOR in their environment to the desired editor, and then export EDITOR.

If the user chooses to monitor/prevent monitoring use of the Help Facility, the choice made is acted on with no further interaction by the user.

SEE ALSO

ed(1), glossary(1), help(1), locate(1), starter(1), usage(1).

WARNINGS

When the UNIX system is delivered to a customer, **/etc/profile** exports the environment variable LOGNAME . If **/etc/profile** has been changed so that LOGNAME is not exported, then the options to monitor/prevent monitoring use of the Help Facility may not work properly.

FILES

HELPLOG	/usr/lib/help/HELPLOG
helpclean	/usr/lib/help/helpclean

NAME

holidays - format of the holiday table used by accounting

DESCRIPTION

The file **/usr/lib/acct/holidays** contains the prime/nonprime table for the accounting system. The table should be edited to reflect your location's holiday schedule for the year. The format is composed of three types of entries:

1. Comment Lines: Comment lines may appear anywhere in the file as long as the first character in the line is an asterisk.
2. Year Designation Line: This line should be the first data line (noncomment line) in the file and must appear only once. The line consists of three fields of four digits each (leading white space is ignored). For example, to specify the year as 1982, prime time at 9:00 a.m., and nonprime time at 4:30 p.m., the following entry would be appropriate:

   ```
   1982  0900  1630
   ```

   A special condition allowed for in the time field is that the time 2400 is automatically converted to 0000.
3. Company Holidays Lines: These entries follow the year designation line and have the following general format:

   ```
   ay-of-year   Month   Day   Description of Holiday
   ```

   The day-of-year field is number in the range of 1 through 366 indicating the day for the corresponding holiday (leading white space is ignored). The other three fields are actually commentary and are not currently used by other programs.

NAME

id - print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

id outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1) in the *D-NIX 5.3 Reference Manual.*

getuid(2) in the *Programmer's Reference Manual.*

NAME

infocmp - compare or print out terminfo descriptions

SYNOPSIS

infocmp [**-d**] [**-c**] [**-n**] [**-I**] [**-L**] [**-C**] [**-r**] [**-u**] [**-s d**|**i**|**l**|**c**]
[**-v**] [**-V**] [**-1**] [**-w** *width*] [**-A** *directory*] [**-B** *directory*] [termname ...]

DESCRIPTION

infocmp can be used to compare a binary *terminfo* (4) entry with other terminfo entries, rewrite a *terminfo* (4) description to take advantage of the **use** = terminfo field, or print out a *terminfo* (4) description from the binary file (*term* (4)) in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **-I** option will be assumed. If more than one *termname* is specified, the **-d** option will be assumed.

Comparison Options [-d] [-c] [-n]

infocmp compares the *terminfo* (4) description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: F for boolean variables, **-1** for integer variables, and NULL for string variables.

-d	produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.
-c	produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **-u** option is worth using.
-n	produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable TERM will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [-I] [-L] [-C] [-r]

The **-I**,**-L**,and **-C** options will produce a source listing for each terminal named.

-I	use the *terminfo* (4) names

-L	use the long C variable name listed in **<term.h>**
-C	use the *termcap* names
-r	when using **-C**, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable TERM will be used for the terminal name.

The source produced by the **-C** option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. *infocmp* will attempt to convert most of the parameterized information, but that which it doesn't will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo* (4), but which are derivable from other *terminfo* (4) variables, will be output. Not all *terminfo* (4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **-r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo* (4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo* (4) format will not necessarily reproduce the original *terminfo* (4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences, are:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [-u]

-u	produce a *terminfo* (4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames* . It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description,

using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname,*but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname.*

The order of the other *termname* entries is significant. Since the terminfo compiler *tic* (1M) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use=** entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. *infocmp* will flag any other *termname* **use=** fields that were not needed.

Other Options [-s d|i|l|c] [-v] [-V] [-1] [-w width]

-s	sort the fields within each type according to the argument below:
d	leave fields in the order that they are stored in the *terminfo* database.
i	sort by *terminfo* name.
l	sort by the long C variable name.
c	sort by the *termcap* name.

If no **-s** option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **-C** or the **-L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

-v	print out tracing information on standard error as the program runs.
-V	print out the version of the program in use on standard error and exit.
-1	cause the fields to printed out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.
-w	change the output to width characters.

Changing Databases [-A directory] [-B directory]

The location of the compiled *terminfo* (4) database is taken from the environment variable TERMINFO. If the variable is not defined, or the terminal is not found in that location, the system *terminfo* (4) database, usually in **/usr/lib/terminfo**, will be used. The options **-A** and **-B** may be used to override this location. The **-A** option will set TERMINFO for the first *termname* and the **-B** option will set TERMINFO for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo* (4) database for a comparison to be made.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database

DIAGNOSTICS

malloc is out of space!

There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use= order dependency found:

A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=*term*' did not add anything to the description.

A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison to be done.

The **-u**,**-d** and **-c** options require at least two terminal names.

SEE ALSO

tic(1M), curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual.*

captoinfo(1M) in the *Administrator's Reference Manual.*

NOTE

The *termcap* database (from earlier releases of UNIX System V) may not be supplied in future releases.

NAME

install - install commands

SYNOPSIS

/etc/install [**-c** *dira*] [**-f** *dirb*] [**-i**] [**-n** *dirc*] [**-m** *mode*] [**-u** *user*] [**-g** *group*] [**-o**] [**-s**] file [*dirx* ...]

DESCRIPTION

The *install* command is most commonly used in "makefiles" [See *make* (1)] to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, *install* will search a set of default directories (**/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 except **-c** and **-f.**
-n *dirc*	If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options except **-c** and **-f.**
-m *mode*	The mode of the new file is set to *mode*. Only available to the superuser.

-u ***user***	The owner of the new file is set to *user*. Only available to the superuser.
-g ***group***	The group id of the new file is set to *group*. Only available to the superuser.
-o	If *file* is found, this option saves the "found" file by copying it to **OLD*file*** in the directory in which it was found. This option is useful when installing a frequently used file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options except **-c.**
-s	Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

make(1).

NAME

link, unlink - link and unlink files and directories

SYNOPSIS

/etc/link file1 file2

/etc/unlink file

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the *rm* (1) and *rmdir* (1) commands be used instead of the *unlink* command.

The only difference between *ln* (1) and *link/unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the *link* (2) and *unlink* (2) system calls.

SEE ALSO

rm(1) in the *D-NIX 5.3 Reference Manual.*

link(2), unlink(2) in the *Programmer's Reference Manual.*

WARNINGS

These commands can be run only by the super-user.

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 the program-to-use-as-Shell exist. 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) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

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.

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).

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

sar: 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 [-ubdycwaqvmprSDA] [-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 counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, inter-process communications, paging and Remote File Sharing.

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, when booting to a multiuser state, to mark the time at which the counters restart from zero. For example, the **/etc/init.d/perf** file writes the restart mark to the daily data by the command entry:

```
su sys -c "/usr/lib/sa/sadc /usr/adm/sa/sa\*'date +%d\*'"
```

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
20,40 8-17 * * 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 **/usr/spool/cron/crontabs/sys** entry:

```
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

will report important activities hourly during the working day.

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), sar(1) in the *D-NIX 5.3 Reference Manual.*

NAME

trenter - enter a trouble report

SYNOPSIS

trenter [-s]

DESCRIPTION

trenter resides on any machine that must submit machinereadable trouble reports to Customer Support. It prompts the user for the data needed to enter the report, and allows for correction of previously entered data, either in-line, or by invoking a text editor. *trenter* also allows users to specify (in a file), default values for fields that will likely remain constant across reports, such as name, address, company name, etc. In addition, facilities are provided to assist local administrators in handling trouble report flow on their systems.

Fields and Values

Trouble reports consist simply of fields and associated values. Each field has a field name, by which it may be referenced. When invoked, *trenter* prompts for values for the trouble report's fields. The following table lists the prompts that are issued, along with their corresponding field names. All fields accept one line of input, except for the problem description, which is a multi-line field, terminated with a line consisting of only ".". The items marked with a star (*) are explained below.

These first nine fields identify the originator of the report.

- Name (NAME) (*)
- Company (CO) (*)
- Phone (PHONE) (*)
- Room Number (ROOM) (*)
- Address (ADDR) (*)
- City (CITY) (*)
- State (STATE) (*)
- Zip Code (ZIP) (*)
- Country (COUNTRY) (*)

These two fields are AT&T-assigned numbers to identify the customer and the specific site.

- Customer ID (CID) (*)
- Site ID (SID) (*)

The next two fields identify the processor on which the problem occurred.

- CPU serial number (CPUNO) (*)
- Machine type (MACH)

The following fields identify the area in which the problem occurred.

- Trouble Report Type (TYPE) Valid responses: doc (documentation), enh (enhancement), cs (customer support), fw (firmware), hdw (hardware), sw (software), or unk (unknown).
- AT&T Product Name (PROD) Examples: UNIX, BASIC, etc.
- Operating system release (OS_REL) (*) The release of the UNIX system on which the problem occurred.
- Product release (PROD_REL) The release of the product given in response to the AT&T product prompt. If product is unix, this prompt is not issued.

The remaining fields define the body of the trouble report.

- Severity (SEV) The severity of the problem (1-4).
- Required date (RDATE) If the severity of the report is 2, the required date for the fix is prompted. The date given must be at least one week from the date of the trouble report.
- Abstract (ABS) One-line description of the problem.
- Description (DESC) Full description of the problem. Note that description input will not be passed through nroff; however, trenter will recognize the macros **.ES** and **.EE** (example start, example end) indicating an indented example (these may be nested).
- Attachments (yes or no) (ATT)

If **?** is given in response to a prompt, a message explaining the field will be printed.

If *trenter* receives an interrupt during prompting, the trouble report will be aborted.

After a trouble report has been completed, the user is given an opportunity to edit any data that has been supplied. Next, a reprint of the trouble report just entered may be requested. Finally, the user is asked whether another report is to be entered. If so, the values for the starred items in the field table above will be carried over from the first report.

Editing Field Values

In order to provide editing while responding to prompts, the following escapes are recognized on input:

-field	Return to a field for which data has previously been supplied. If the field name is not specified, return to the previous field. The value already assigned to the field is printed, and the user may enter either new data, or another editing command.
!e	Invoke the editor *ed* (1) with any text already supplied for the current prompt in the edit buffer (an alternate editor can be specified: see "Specifying Default Values" below).
>	Move down to the first unfilled field. This is useful, for example, when the - command has been used to fix a single field near the top of the report, and the user wishes to quickly return to the point where they left off.
=field	Print the value currently assigned to the given field.
??	Print a summary of editing functions.

Editing commands are only recognized when they appear at the beginning of the input line; they may be escaped using a backslash (\).

Specifying Default Values

Users may provide default values for any fields marked with (*) above. These values are specified in a file **.trdef** in the user's home directory. Entries in this file are of form:

```
field=value
```

where field is a field name from the table above.

The editor to be used for field editing can be overridden with a **.trdef** entry by assigning the name of the desired program to the field EDITOR.

During prompting, *trenter* will print any values supplied for fields from a **.trdef** file. By default, it will stop at each such field and wait for either a carriage return (indicating confirmation), an edit command, or new data. If invoked with a **-s** option, *trenter* will print the supplied values, but will not stop for confirmation.

Default values specified in **.trdef** files may be changed, on a per-report basis, using the editing functions described above.

FILES

.trdef	default value file
/usr/spool/trenter	spool directory

NAME

uucheck - check the uucp directories and permissions file

SYNOPSIS

/usr/lib/uucp/uucheck [**-v**] [**-x***debug_level*]

DESCRIPTION

uucheck checks for the presence of the *uucp* system required files and directories. Within the *uucp* makefile, it is executed before the installation takes place. It also checks for some obvious errors in the Permissions file (**/usr/lib/uucp/Permissions**). When executed with the **-v** option, it gives a detailed explanation of how the uucp programs will interpret the Permissions file. The **-x** option is used for debugging. *debug-option* is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that *uucheck* can only be used by the super-user or *uucp*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucico(1M), uusched(1M).

uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual.*

BUGS

The program does not check file/directory modes or some errors in the Permissions file such as duplicate login or machine name.

NAME

uucico - file transport program for the uucp system

SYNOPSIS

/usr/lib/uucp/uucico [**-r** *role_number*] [**-x** *debug_level*]

[**-i** *interface*] [**-d** *spool_directory*] **-s** *system_name*

DESCRIPTION

uucico is the file transport program for *uucp* work file transfers. Role numbers for the **-r** are the digit 1 for master mode or 0 for slave mode (default). The **-r** option should be specified as the digit 1 for master mode when *uucico* is started by a program or *cron*. *uux* and *uucp* both queue jobs that will be transferred by *uucico*. It is normally started by the scheduler, *uusched,*but can be started manually; this is done for debugging. For example, the shell *uutry* starts *uucico* with debugging turned on. A single digit must be used for the **-x** option with higher numbers for more debugging.

The **-i** option defines the interface used with *uucico*. This interface only affects slave mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface), and TLIS (Transport Layer Interface with Streams modules, read/write).

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Devconfig
/usr/lib/uucp/Sysfiles
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uusched(1M), uutry(1M).

uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual.*

NAME

uucleanup - uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**-C*time***] [**-W*time***] [**-X*time***] [**-m*string***]
[**-o*time***] [**-s*system***]

DESCRIPTION

uucleanup will scan the spool directories for old files and take appropriate action to remove them in a useful way:

- Inform the requestor of send/receive requests for systems that can not be reached.
- Return mail, which cannot be delivered, to the sender.
- Delete or execute rnews for rnews type files (depending on where the news originated--locally or remotely).
- Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that *uucleanup* will process as if all option *times* were specified to the default values unless *time* is specifically set.

The following options are available.

-C*time*	Any **C.** files greater or equal to *time* days old will be removed with appropriate information to the requestor. (default 7 days)
-D*time*	Any **D.** files greater or equal to *time* days old will be removed. An attempt will be made to deliver mail messages and execute rnews when appropriate. (default 7 days)
-W*time*	Any **C.** files equal to *time* days old will cause a mail message to be sent to the requestor warning about the delay in contacting the remote. The message includes the *JOBID*, and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (**-m** option). (default 1 day)
-X*time*	Any **X.** files greater or equal to *time* days old will be removed. The **D.** files are probably not present (if they were, the **X.** could get executed). But if there are **D.** files, they will be taken care of by D. processing. (default 2 days)
-m*string*	This line will be included in the warning message generated by the **-W** option.

-o*time*	Other files whose age is more than *time* days will be deleted. (default 2 days) The default line is "See your local administrator to locate the problem".
-s*system*	Execute for *system* spool directory only.
-x*debug_level*	The **-x** debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information. (If **uucleanup** was compiled with -DSMALL, no debugging output will be available.).

This program is typically started by the shell *uudemon.cleanup*, which should be started by *cron* (1M).

FILES

/usr/lib/uucp	directory with commands used by *uucleanup* internally
/usr/spool/uucp	spool directory

SEE ALSO

cron(1M).

uucp(1C), uux(1C) in the *User's Reference Manual.*

NAME

uusched - the scheduler for the uucp file transport program

SYNOPSIS

/usr/lib/uucp/uusched [**-x** *debug_level*] [**-u** *debug_level*]

DESCRIPTION

uusched is the *uucp* file transport scheduler. It is usually started by the daemon *uudemon.hour* that is started by *cron* (1M) from an entry in **/usr/spool/cron/crontab**:

```
39 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.hour > /dev/null"
```

The two options are for debugging purposes only; **-x** *debug_level* will output debugging messages from *uusched* and **-u** *debug_level* will be passed as **-x** *debug_level* to *uucico*. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucico(1M).

uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual*.

NAME

Uutry - try to contact remote system with debugging on

SYNOPSIS

/usr/lib/uucp/Uutry [**-x** *debug_level*] [**-r**] *system_name*

DESCRIPTION

Uutry is a shell that is used to invoke *uucico* to call a remote site. Debugging is turned on (default is level 5); **-x** will override that value. The **-r** overrides the retry time in **/usr/spool/uucp/.status.** The debugging output is put in file **/tmp/***system_name*. A tail **-f** of the output is executed. A DELETE or BREAK will give control back to the terminal while the *uucico* continues to run, putting its output in **/tmp/***system_name*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*
/tmp/system_name

SEE ALSO

uucico(1M).

uucp(1C), uux(1C) in the *User's Reference Manual.*

NAME

uuxqt - execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [**-s** *system*] [**-x** *debug_level*]

DESCRIPTION

uuxqt is the program that executes remote job requests from remote systems generated by the use of the *uux* command. (*mail* uses *uux* for remote mail requests). *uuxqt* searches the spool directories looking for *X.* files. For each *X.* file, *uuxqt* checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The *Permissions* file is used to validate file accessibility and command execution permission.

There are two environment variables that are set before the *uuxqt* command is executed:

UU_MACHINE is the machine that sent the job (the previous one).

UU_USER is the user that sent the job.

These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

The **-x** *debug_level* is a single digit between 0 and 9. Higher numbers give more detailed debugging information.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*

SEE ALSO

uucico(1M).

uucp(1C), uustat(1C), uux(1C), mail(1) in the *User's Reference Manual.*

NAME

whodo - who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

whodo produces formatted and dated output from information in the */etc/utmp* and */etc/ps_data* files.

The display is headed by the date, time and machine name. For each user logged in, device name, user-id and login time is shown, followed by a list of active processes associated with the user-id. The list includes the device name, process-id, cpu minutes and seconds used, and process name.

EXAMPLE

The command:

```
whodo
```

produces a display like this:

```
Tue Mar 12 15:48:03 1985 bailey
tty09    mcn        8:51
    tty09   28158    0:29 sh
tty52    bdr       15:23
    tty52   21688    0:05 sh
    tty52   22788    0:01 whodo
    tty52   22017    0:03 vi
    tty52   22549    0:01 sh
xt162    lee       10:20
    tty08    6748    0:01 layers
    xt162    6751    0:01 sh
    xt163    6761    0:05 sh
    tty08    6536    0:05 sh
```

FILES

/etc/passwd
/etc/ps_data
/etc/utmp

SEE ALSO

ps(1), who(1) in the *D-NIX 5.3 Reference Manual.*

2

NAME

intro - introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable. Usually, the header files containing these structure declarations can be found in the directories **/usr/include** or **/usr/include/sys.** For inclusion in C language programs, however, the syntax **#include <filename.h>** or **#include <sys/filename.h>** should be used.

NAME

a.out - common assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

The file name **a.out** is the default output file name from the link editor *ld* (1). The link editor will make *a.out* executable if there were no errors in linking. The output file of the assembler *as* (1), also follows the common object file format of the *a.out* file although the default file name is different.

A common object file consists of a file header, a UNIX system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

```
File header.
UNIX system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
... Section n line numbers.
Symbol table.
String table.
```

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **-s** option of *ld* (1) or if they were removed by *strip* (1). Also note that the relocation information will be absent after linking unless the **-r** option of *ld* (1) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack.

The **a.out** file produced by *ld* (1) has the magic number 0413 in the first field of the UNIX system header. The headers (file header, UNIX system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address will equal 0x10000 plus the size of the headers, and will vary depending upon the number of section headers in the **a.out** file. In an **a.out** file with three sections (.text, .data, and .bss), the first text address is at 0x100A8 on the DS90 computer. The text segment is not writable by the program; if other processes are executing the same **a.out** file, the processes will share a single text segment.

The data segment starts at the next 64K boundary past the last text address. The first data address is determined by the following: If an **a.out** file were split into 8K chunks, one of the chunks would contain both the end of text and the beginning of data. When the core image is created, that chunk will appear twice; once at the end of text and once at the beginning of data (with some unused space in between). The duplicated chunk of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the page size without having to realign the beginning of the data section to a page boundary. Therefore the first data address is the sum of the next segment boundary past the end of text plus the remainder of the last text address divided by 8K. If the last text address is a multiple of 8K no duplication is necessary.

On the DS90 computer the stack begins at location 0x80000000 and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the *brk* (2) system call.

For relocatable files the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an "external symbol", and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

File Header

The format of the **filehdr** header is

```
struct filehdr
{
      unsigned short  f_magic;    /* magic number */
      unsigned short  f_nscns;    /* number of sections */
      long            f_timdat;   /* time and date stamp */
      long            f_symptr;   /* file ptr to symtab */
      long            f_nsyms;    /* # symtab entries */
      unsigned short  f_opthdr;   /* sizeof(opt hdr) */
      unsigned short  f_flags;    /* flags */
};
```

UNIX System Header

The format of the UNIX system header is

```
typedef struct aouthdr
{
      short       magic;          /* magic number */
      short       vstamp;         /* version stamp */
      long        tsize;          /* text size in bytes, padded */
      long        dsize;          /* initialized data (.data) */
      long        bsize;          /* uninitialized data (.bss) */
      long        entry;          /* entry point */
      long        text_start;     /* base of text used for this file */
      long        data_start;     /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is

```
struct scnhdr
{
      char            s_name[SYMNMLEN];   /* section name */
      long            s_paddr;            /* physical address */
      long            s_vaddr;            /* virtual address */
      long            s_size;             /* section size */
      long            s_scnptr;           /* file ptr to raw data */
      long            s_relptr;           /* file ptr to relocation */
      long            s_lnnoptr;          /* file ptr to line numbers */
      unsigned short  s_nreloc;           /* # reloc entries */
      unsigned short  s_nlnno;            /* # line number entries */
      long            s_flags;            /* flags */
};
```

Relocation

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 */
      ushort      r_type;         /* relocation type */
};
```

The start of the relocation information is *s_relptr* from the section header. If there is no relocation information, *s_relptr* is 0.

Symbol Table

The format of each symbol in the symbol table is

```
#define SYMNMLEN     8
#define FILNMLEN     14
#define DIMNUM 4

struct syment
{
        union       /* all ways to get a symbol name */
        {
                char        _n_name[SYMNMLEN];  /* name of symbol */
                struct
                {
                        long _n_zeroes;     /* == 0L if in string table */
                        long _n_offset;     /* location in string table */
                } _n_n;
                char        '*_n_nptr[2];   /* allows overlaying */
        } _n;
        long            n_value;        /* value of symbol */
        short           n_scnum;        /* section number */
        unsigned short  n_type;         /* type and derived type */
        char            n_sclass;       /* storage class */
        char            n_numaux;       /* number of aux entries */
};

#define n_offset        _n._n_n._n_offset
#define n_nptr          _n._n_nptr[1]
```

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent {
        struct {
        long        x_tagndx;
                union {
                        struct {
                                unsigned short  x_lnno;
                                unsigned short  x_size;
                        } x_lnsz;
                        long  x_fsize;
                } x_misc;
                union {
                        struct {
                                long  x_lnnoptr;
                                long  x_endndx;
                        }  x_fcn;
                        struct {
                                unsigned short  x_dimen[DIMNUM];
                        } x_ary;
                } x_fcnary;
                unsigned short  x_tvndx;
        } x_sym;
        struct {
              char  x_fname[FILNMLEN];
        } x_file;
        struct {
              long     x_scnlen;
                    unsigned short  x_nreloc;
              unsigned short  x_nlinno;
        } x_scn;
        struct {            long          x_tvfill;
              unsigned short   x_tvlen;
              unsigned short   x_tvran[2];
        } x_tv;
};
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is *f_symptr* (from the file header) bytes from the beginning of the file. If the symbol table is stripped, *f_symptr* is 0. The string table (if one exists) begins at *f_symptr* + (*f_nsyms* * SYMESZ) bytes from the beginning of the file.

SEE ALSO

as(1), cc(1), ld(1), brk(2), filehdr(4), ldfcn(4), linenum(4), reloc(4), scnhdr(4), syms(4).

NAME

acct - per-process accounting file format

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

Files produced as a result of calling *acct* (2) have records in the form defined by **<sys/acct.h>**,whose contents are:

```
typedef  ushort comp_t;        /* "floating point" */
                               /* 13-bit fraction, 3-bit exponent */

struct      acct
{
      char        ac_flag;         /* Accounting flag */
      char        ac_stat;         /* Exit status */
      ushort      ac_uid;          /* Accounting user ID */
      ushort      ac_gid;          /* Accounting group ID */
      dev_t       ac_tty;          /* control typewriter */
      time_t      ac_btime;        /* Beginning time */
      comp_t      ac_utime;        /* acctng user time in clock ticks */
      comp_t      ac_stime;        /* acctng system time in clock ticks */
      comp_t      ac_etime;        /* acctng elapsed time in clock ticks */
      comp_t      ac_mem;          /* memory usage in clicks */
      comp_t      ac_io;           /* chars trnsfrd by read/write */
      comp_t      ac_rw;           /* number of block reads/writes */
      char        ac_comm[8];      /* command name */
};

extern      struct      acct        acctbuf;
 extern     struct      inode       *acctp;     /* inode of accounting file */

#define     AFORK       01          /* has executed fork, but no exec */
#define     ASU         02          /* used super-user privileges */
#define     ACCTF       0300        /* record type: 00 = acct */
```

In *ac_flag,* the AFORK flag is turned on by each *fork* (2) and turned off by an *exec* (2). The *ac_comm* field is inherited from the parent process and is reset by any *exec*. Each time the system charges the process with a clock tick, it also adds to *ac_mem* the current process size, computed as follows:

```
    (data size) + (text size)/(number of in-core processes using text)
```

The value of *ac_mem / (ac_stime+ac_utime)* can be viewed as an approximation to the mean process size, as modified by text-sharing.

The structure **tacct.h,**which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```
/*
 * total accounting (for acct period), also for day
 */
struct      tacct {
            uid_t            ta_uid;          /* userid */
            char             ta_name[8];      /* login name */
            float            ta_cpu[2];       /* cum. cpu time, p/np (mins) */
            float            ta_kcore[2];     /* cum kcore-minutes, p/np */
            float            ta_con[2];       /* cum. connect time, p/np,
                                                     mins */
            float            ta_du;           /* cum. disk usage */
            long             ta_pc;           /* count of processes */
            unsigned short   ta_sc;           /* count of login sessions */
            unsigned short   ta_dc;           /* count of disk samples */
            unsigned short   ta_fee;          /* fee for special services */
};
```

SEE ALSO

acct(2), exec(2), fork(2).

acct(1M) in the *Administrator's Reference Manual.*

acctcom(1) in the *User's Reference Manual.*

BUGS

The *ac_mem* value for a short-lived command gives little information about the actual size of the command, because *ac_mem* may be incremented while a different command (e.g., the shell) is being executed by the process.

NAME

ar - common archive file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The archive command *ar* (1) is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor *ld* (1).

Each archive begins with the archive magic string.

```
#define ARMAG  "!<arch>\n        /* magic string */
#define SARMAG 8                 /* length of magic string */
```

Each archive which contains common object files [see *a.out* (4)] includes an archive symbol table. This symbol table is used by the link editor *ld* (1) to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by *ar.*

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
#define ARFMAG        "`\n              /* header trailer string */

struct ar_hdr                           /* file member header */
{
     char      ar_name[16];             /* '/' terminated file member name */
     char      ar_date[12];             /* file member date */
     char      ar_uid[6];               /* file member user identification */
     char      ar_gid[6];               /* file member group identification */
     char      ar_mode[8];              /* file member mode (octal) */
     char      ar_size[10];             /* file member size */
     char      ar_fmag[2];              /* header trailer string */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for *ar_mode* which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

The *ar_name* field is blank-padded and slash (/) terminated. The *ar_date* field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command *ar* (1) is used. Conversion tools such as *convert* (1) exist to aid in the transportation of non-common format archives to this format.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

If the archive symbol table exists, the first file in the archive has a zero length name (i.e., **ar_name[0] == '/'**). The contents of this file are as follows:

- The number of symbols. Length: 4 bytes.
- The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".
- The name string table. Length: *ar_size* - (4 bytes * ("the number of symbols" + 1)).

The number of symbols and the array of offsets are managed with *sgetl* and *sputl.* The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

SEE ALSO

ar(1), ld(1), strip(1), sputl(3X), a.out(4).

WARNINGS

strip (1) will remove all archive symbol entries from the header. The archive symbol entries must be restored via the **ts** option of the *ar* (1) command before the archive can be used with the link editor *ld* (1).

NAME

checklist - list of file systems processed by fsck and ncheck

DESCRIPTION

checklist resides in directory **/etc** and contains a list of, at most, 15 *special file* names. Each *special file* name is contained on a separate line and corresponds to a file system. Each file system will then be automatically processed by the *fsck* (1M) command.

FILES

/etc/checklist

SEE ALSO

fsck(1M) in the *D-NIX 5.3 Reference Manual.*

ncheck(1M) in the *System Administrator's Reference Manual.*

NAME

core - format of core image file

DESCRIPTION

The UNIX system writes out a core image of a terminated process when any of various errors occur. See *signal* (2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The first section of the core image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter *usize,* which is defined in **<sys/param.h>.** The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in **<sys/user.h>.** Not included in this file are the locations of the registers. These are outlined in **<sys/reg.h>.**

SEE ALSO

sdb(1), setuid(2), signal(2).

crash(1M) in the *Administrator's Reference Manual.*

NAME

cpio - format of cpio archive

DESCRIPTION

The *header* structure, when the **-c** option of *cpio* (1) is not used, is:

```
struct {
      short       h_magic,
                  h_dev;
      ushort      h_ino,
                  h_mode,
                  h_uid,
                  h_gid;
      short       h_nlink,
                  h_rdev,
                  h_mtime[2],
                  h_namesize,
                  h_filesize[2];
      char        h_name[h_namesize rounded to word];
} Hdr;
```

When the **-c** option is used, the *header* information is described by:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
      &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
      &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
      &Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize* , respectively. The contents of each file are recorded in an element of the array of varying length structures, *archive,* together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in *stat* (2). The length of the null-terminated path name *h_name* , including the null byte, is given by *h_namesize.*

The last record of the *archive* always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with h_filesize equal to zero.

SEE ALSO

stat(2).

cpio(1), find(1) in the *D-NIX 5.3 Reference Manual.*

NAME

dir - format of directories

SYNOPSIS

#include <sys/dir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry [see *fs* (4)]. The structure of a directory entry as given in the include file is:

```
#ifndef    DIRSIZ
#define    DIRSIZ     14
#endif
struct     direct
{
           ushort     d_ino;
           char       d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are for . and ... The first is an entry for the directory itself. The second is for the parent directory. The meaning of .. is modified for the root directory of the master file system; there is no parent, so .. has the same meaning as ..

SEE ALSO

fs(4).

NAME

dirent - file system independent directory entry

SYNOPSIS

#include <sys/dirent.h>
#include <sys/types.h>

DESCRIPTION

Different file system types may have different directory entries. The *dirent* structure defines a file system independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the *getdents* (2) system call.

The *dirent* structure is defined below.

```
struct     dirent {
                long              d_ino;
                off_t             d_off;
                unsigned short    d_reclen;
                char              d_name[1];
           };
```

The *d_ino* is a number which is unique for each file in the file system. The field *d_off* is the offset of that directory entry in the actual file system directory. The field *d_name* is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most MAXNAMLEN characters. This results in file system independent directory entries being variable length entities. The value of *d_reclen* is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that it will always result in the next entry being on a long boundary.

FILES

/usr/include/sys/dirent.h

SEE ALSO

getdents(2).

NAME

filehdr - file header for common object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct      filehdr
{
      unsigned short  f_magic ;   /* magic number */
      unsigned short  f_nscns ;   /* number of sections */
      long            f_timdat ;  /* time & date stamp */
      long            f_symptr ;  /* file ptr to symtab */
      long            f_nsyms ;   /* # symtab entries */
      unsigned short  f_opthdr ;  /* sizeof(opt hdr) */
      unsigned short  f_flags ;   /* flags */
};
```

F_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek* (3S) to position an I/O stream to the symbol table. The UNIX system optional header is 28-bytes. The valid magic numbers are given below:

```
#define   MC68020MAGIC     0630     /* Same as NCR's */
#define   MC68KMAGIC       0520
#define   MC68KWRMAGIC     0620     /* writable text segment */
#define   MC68TVMAGIC      0521
#define   MC68KROMAGIC     0625     /* readonly sharable text
                                    segments */
#define   MC68KPGMAGIC     0620     /* demand paged text segment */
#define   M68MAGIC         0210
#define   M68TVMAGIC       0211
```

The value in *f_timdat* is obtained from the *time* (2) system call. Flag bits currently defined are:

```
#define   F_RELFLG     0000001   /* relocation entries stripped */
#define   F_EXEC       0000002   /* file is executable */
#define   F_LNNO       0000004   /* line numbers stripped */
#define   F_LSYMS      0000010   /* local symbols stripped */
#define   F_MINMAL     0000020   /* minimal object file */
#define   F_UPDATE     0000040   /* update file, ogen produced */
#define   F_SWABD      0000100   /* file is "pre-swabbed" */
#define   F_AR16WR     0000200   /* 16-bit DEC host */
#define   F_AR32WR     0000400   /* 32-bit DEC host */
#define   F_AR32W      0001000   /* non-DEC host */
#define   F_PATCH      0002000   /* "patch" list in opt hdr */
#define   F_BM32ID     0160000   /* WE32000 family ID field */
#define   F_BM32B      0020000   /* file contains WE 32100 code */
#define   F_BM32MAU    0040000   /* file reqs MAU to execute */
#define   F_BM32RST    0010000   /* this object file contains restore
                                       work around [3B5/3B2 only] */
```

SEE ALSO

time(2), fseek(3S), a.out(4).

NAME

fspec - format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the UNIX system with non-standard tabs, (i.e., tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns;
2. a - followed immediately by an integer *n* , indicating tabs at intervals of *n* columns;
3. a - followed by the name of a "canned" tab specification.

Standard tabs are specified by **t-8**, or equivalently, **t1,9,17,25, etc.** The canned tabs which are recognized are defined by the *tabs* (1) command.

s*size* The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t-8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

SEE ALSO

ed(1) in the *D-NIX 5.3 Reference Manual.*

newform(1), tabs(1) in the *User's Reference Manual.*

NAME

fstab - file-system-table

DESCRIPTION

The **/etc/fstab** file contains information about file systems for use by *mount* (1M). Each entry in **/etc/fstab** has the following format:

column 1	block special file name of file system or advertised remote resource
column 2	mount-point directory
column 3	**-r** if to be mounted read-only; **-d[r]** if remote
column 4	(optional) file system type string
column 5+	ignored

White-space separates columns. Lines beginning with "# " are comments. Empty lines are ignored.

A file-system-table might read:

```
/dev/dsk/c1d0s2 /usr  S51K
/dev/dsk/c1d1s2 /usr/src -r
adv_resource /mnt -d
```

FILES

/etc/fstab

SEE ALSO

mount(1M), rmountall(1M) in the *Administrator's Reference Manual.*

NAME

gettydefs - speed and terminal settings used by getty

DESCRIPTION

The **/etc/gettydefs** file contains information used by *getty* (1M) to set up the speed and terminal settings for a line. It supplies information on what the *login* prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a BREAK character.

Each entry in **/etc/gettydefs** has the following format:

```
label# initial-flags # final-flags # login-prompt #next-label
```

Each entry is followed by a blank line. The various fields can contain quoted characters of the form **b** , **n** , **c** , etc., as well as ***nnn***, where *nnn* is the octal value of the desired character. The various fields are:

label	This is the string against which *getty* tries to match its second argument. It is often the speed, such as **1200**, at which the terminal is supposed to run, but it need not be (see below).
initial-flags	These flags are the initial *ioctl* (2) settings to which the terminal is to be set if a terminal type is not specified to *getty*. The flags that *getty* understands are the same as the ones listed in **/usr/include/sys/termio.h** [see *termio* (7)]. Normally only the speed flag is required in the *initial-flags*. *getty* automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until *getty* executes *login* (1).
final-flags	These flags take the same values as the *initial-flags* and are set just prior to *getty* executes *login*. The speed flag is again required. The composite flag **SANE** takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are **TAB3,** so that tabs are sent to the terminal as spaces, and **HUPCL,** so that the line is hung up on the final close.
login-prompt	This entire field is printed as the *login-prompt.* Unlike the above fields where white space is ignored (a space, tab or new-line), they are included in the *login-prompt* field.
next-label	If this entry does not specify the desired speed, indicated by the user typing a BREAK character, then *getty* will search for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds are linked together in this fashion, into a closed set; For instance, **2400** linked to **1200**,

which in turn is linked to **300**, which finally is linked to **2400**.

If *getty* is called without a second argument, then the first entry of **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. It is also used if *getty* can not find the specified *label.* If **/etc/gettydefs** itself is missing, there is one entry built into the command which will bring up a terminal at **300** baud.

It is strongly recommended that after making or modifying **/etc/gettydefs,** it be run through *getty* with the check option to be sure there are no errors.

FILES

/etc/gettydefs

SEE ALSO

ioctl(2).

getty(1M), termio(7) in the *Administrator's Reference Manual.*

login(1) in the *D-NIX 5.3 Reference Manual.*

NAME

group - group file

DESCRIPTION

group contains for each group the following information:

group name
encrypted password
numerical group ID
comma-separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory **/etc.** Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

passwd(4).

passwd(1), newgrp(1M) in the *D-NIX 5.3 Reference Manual.*

NAME

inittab - script for the init process

DESCRIPTION

The *inittab* file supplies the script to *init's* role as a general process dispatcher. The process that constitutes the majority of *init's* process dispatching activities is the line process **/etc/getty** that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

The *inittab* file is composed of entries that are position dependent and have the following format:

```
id:rstate:action:process
```

Each entry is delimited by a newline, however, a backslash (\) preceding a newline indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the *process* field using the *sh* (1) convention for comments. Comments for lines that spawn *getty* s are displayed by the *who* (1) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the *inittab* file. The entry fields are:

id	This is one or two characters used to uniquely identify an entry.
rstate	This defines the *run-level* in which this entry is to be processed. *run-levels* effectively correspond to a configuration of processes in the system. That is, each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by a number ranging from **0** through **6.** As an example, if the system is in *run-level* **1** , only those entries having a **1** in the *rstate* field will be processed. When *init* is requested to change *run-levels,* all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (**SIGTERM**) and allowed a 20-second grace period before being forcibly terminated by a kill signal (**SIGKILL**). The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from **0-6**. If no *run-level* is specified, then the process is assumed to be valid at all *run-levels* **0-6.** There are three other values, **a** , **b** and **c** , which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* [see *init* (1M)] process requests them to be run (regardless of the current *run-level* of the system). They differ from *run-levels* in that *init* can never enter *run-level* **a** , **b** or **c.** Also, a request for the execution of any of these processes does not change the current *run-level.* Fur-

	thermore, a process started by an **a** , **b** or **c** command is not killed when *init* changes levels. They are only killed if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab** , or *init* goes into the *SINGLE USER* state.
action	Key words in this field tell *init* how to treat the process specified in the *process* field. The actions recognized by *init* are as follows:
respawn	If the process does not exist then start the process, do not wait for its termination (continue scanning the *inittab* file), and when it dies restart the process. If the process currently exists then do nothing and continue scanning the *inittab* file.
wait	Upon *init's* entering the *run-level* that matches the entry's *rstate,* start the process and wait for its termination. All subsequent reads of the *inittab* file while *init* is in the same *run-level* will cause *init* to ignore this entry.
once	Upon *init's* entering a *run-level* that matches the entry's *rstate,* start the process, do not wait for its termination. When it dies, do not restart the process. If upon entering a new *run-level,* where the process is still running from a previous *run-level* change, the program will not be restarted.
boot	The entry is to be processed only at *init's* boot-time read of the *inittab* file. *Init* is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match *init's run-level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.
bootwait	The entry is to be processed the first time *init* goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process will run right after the boot.) *Init* starts the process, waits for its termination and, when it dies, does not restart the process.
powerfail	Execute the process associated with this entry only when *init* receives a power fail signal [SIGPWR see *signal* (2)].
powerwait	Execute the process associated with this entry only when *init* receives a power fail signal (SIGPWR) and wait until it terminates before continuing any processing of *inittab*.
off	If the process associated with this entry is currently running, send the warning signal (SIGTERM) and wait

20 seconds before forcibly terminating the process via the kill signal (SIGKILL). If the process is nonexistent, ignore the entry.

ondemand This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with *run-levels.* This is used only with the **a** , **b** or **c** values described in the *rstate* field.

initdefault An entry with this *action* is only scanned when *init* initially invoked. *Init* uses this entry, if it exists, to determine which *run-level* to enter initially. It does this by taking the highest *run-level* specified in the **rstate** field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **0123456** and so *init* will enter *run-level* **6.** Additionally, if *init* does not find an **initdefault** entry in **/etc/inittab,** then it will request an initial *run-level* from the user at reboot time.

sysinit Entries of this type are executed before *init* tries to access the console (i.e., before the **Console Login:** prompt). It is expected that this entry will be only used to initialize devices on which *init* might try to ask the *run-level* question. These entries are executed and waited for before continuing.

process This is a *sh* command to be executed. The entire **process** field is prefixed with *exec* and passed to a forked *sh* as **sh -c 'exec** *command'*. For this reason, any legal *sh* syntax can appear in the *process* field. Comments can be inserted with the ***; # comment*** syntax.

FILES

/etc/inittab

SEE ALSO

exec(2), open(2), signal(2).

getty(1M), init(1M), sh(1), who(1) in the *D-NIX 5.3 Reference Manual.*

NAME

inode - format of an i-node

SYNOPSIS

#include <sys/types.h>
#include <sys/ino.h>

DESCRIPTION

An i-node for a plain file or directory in a file system has the following structure defined by **<sys/ino.h>.**

```
/* Inode structure as it appears on a disk block. */

struct      dinode
{
        ushort    di_mode;          /* mode and type of file */
        short     di_nlink;         /* number of links to file */
        ushort    di_uid;           /* owner's user id */
        ushort    di_gid;           /* owner's group id */
        off_t     di_size;          /* number of bytes in file */
        char      di_addr[40];      /* disk block addresses */
        time_t    di_atime;         /* time last accessed */
        time_t    di_mtime;         /* time last modified */
        time_t    di_ctime;         /* time of last file status change */
};
/*
 *      the 40 address bytes:
 *      39 used; 13 addresses
 *      of 3 bytes each.
 */
```

For the meaning of the defined types *off_t* and *time_t* see *types* (5).

SEE ALSO

stat(2), fs(4), types(5).

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 *lines* file.

FILES

/etc/issue

SEE ALSO

login(1) in the *D-NIX 5.3 Reference Manual.*

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 common object files and archives containing common object files. 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.
IOPTR(**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

ldopen (3X) and *ldaopen* [see *ldopen* (3X)]

open a common object file

ldclose (3X) and *ldaclose* [see *ldclose* (3X)]

close a common object file

(2) functions that read header or symbol table information

ldahread (3X)

read the archive header of a member of an archive file *ldfhread* (3X)

read the file header of a common object file

ldshread (3X) and *ldnshread* [see *ldshread* (3X)]

read a section header of a common object file

ldtbread (3X)

read a symbol table entry of a common object file

ldgetname (3X)

retrieve a symbol name from a symbol table entry or from the string table

(3) functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

ldohseek (3X)

seek to the optional file header of a common object file

ldsseek (3X) and *ldnsseek* [see *ldsseek* (3X)]

seek to a section of a common object file

ldrseek (3X) and *ldnrseek* [see *ldrseek* (3X)]

seek to the relocation information for a section of a common object file

ldlseek (3X) and *ldnlseek* [see *ldlseek* (3X)]

seek to the line number information for a section of a common object file

ldtbseek (3X)

seek to the symbol table of a common object file

(4) the function *ldtbindex* (3X) which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages.

All the functions except *ldopen* (3X), *ldgetname* (3X), *ldtbindex* (3X) return either SUCCESS or FAILURE , both constants defined in **ldfcn.h.** *ldopen* (3X) and *ldaopen* [(see *ldopen* (3X)] both return pointers to an LDFILE structure.

Additional access to an object file is provided through a set of macros defined in **ldfcn.h.** These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the LDFILE structure into a reference to its file descriptor field.

The following macros are provided:

```
GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
```

```
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)
```

The STROFFSET macro calculates the address of the string table. See the manual entries for the corresponding standard input/output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access routine library **libld.a.**

SEE ALSO

fseek(3S), ldahread(3X), ldclose(3X), ldgetname(3X), ldfhread(3X), ldlread(3X), ldlseek(3X), ldohseek(3X), ldopen(3X), ldrseek(3X), ldlseek(3X), ldshread(3X), ldtbindex(3X), ldtbread(3X), ldtbseek(3X), stdio(3S), intro(5).

WARNING

The macro FSEEK defined in the header file **ldfcn.h** translates into a call to the standard input/output function *fseek* (3S). FSEEK should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members!

NAME

limits - file header for implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

The header file *<limits.h>* is a list of magnitude limitations imposed by a specific implementation of the operating system. All values are specified in decimal.

```
#define ARG_MAX    5120       /* max length of arguments to exec */
#define CHAR_BIT   8          /* # of bits in a "char" */
#define CHAR_MAX   127        /* max integer value of a "char" */
#define CHAR_MIN   -128       /* min integer value of a "char" */
#define CHILD_MAX  50         /* max # of processes per user id */
#define CLK_TCK    64         /* # of clock ticks per second */
#define DBL_DIG    16         /* digits of precision of a "double" */
#define DBL_MAX    1.79769313486231470e+308 /*max decimal value of a
                                               "double"*/
#define DBL_MIN    4.94065645841246544e-324 /*min decimal value of a
                                               "double"*/
#define FCHR_MAX   4294967296 /* max size of a file in bytes */
#define FLT_DIG    7          /* digits of precision of a "float" */
#define FLT_MAX    3.40282346638528860e+38 /*max decimal value of a
                                               "float" */
#define FLT_MIN    1.40129846432481707e-45 /*min decimal value of a
                                               "float" */
#define HUGE_VAL   _infinity()     /*error value returned by Math lib*/
#define INT_MAX    2147483647      /* max decimal value of an "int" */
#define INT_MIN    -2147483648     /* min decimal value of an "int" */
#define LINK_MAX   1000       /* max # of links to a single file */
#define LONG_MAX   2147483647      /* max decimal value of a "long" */
#define LONG_MIN   -2147483648     /* min decimal value of a "long" */
#define NAME_MAX   14         /* max # of characters in a file name */
#define OPEN_MAX   72         /* max # of files a process can have
                                 open */
#define PASS_MAX   8          /* max # of characters in a password */
#define PATH_MAX   1024       /* max # of characters in a path name */
#define PID_MAX    30000      /* max value for a process ID */
#define PIPE_BUF   5120       /* max # bytes atomic in write to a pipe */
#define PIPE_MAX   5120       /* max # bytes written to a pipe in a
                                 write */
#define SHRT_MAX   32767      /* max decimal value of a "short" */
#define SHRT_MIN   -32767     /* min decimal value of a "short" */
#define STD_BLK    2048       /* # bytes in a physical I/O block */
#define SYS_NMLN   9          /* # of chars in uname-returned strings */
#define UID_MAX    30000      /* max value for a user or group ID */
#define USI_MAX    4294967296 /* max decimal value of an "unsigned" */
#define WORD_BIT   32         /* # of bits in a "word" or "int" */
```

NOTE

The contents of the header file may vary between releases.

NAME

linenum - line number entries in a common object file

SYNOPSIS

#include <linenum.h>

DESCRIPTION

The *cc* command generates an entry in the object file for each C source line on which a breakpoint is possible [when invoked with the **-g** option; see *cc* (1)]. Users can then reference line numbers when using the appropriate software test system [see *sdb* (1)]. The structure of these line number entries appears below.

```
struct      lineno
{
      union
      {
            long  l_symndx ;
            long  l_paddr ;
      }           l_addr ;
      unsigned short   l_lnno ;
} ;
```

Numbering starts with one for each function. The initial line number entry for a function has *l_lnno* equal to zero, and the symbol table index of the function's entry is in *l_symndx*. Otherwise, *l_lnno* is non-zero, and *l_paddr* is the physical address of the code for the referenced line. Thus the overall structure is the following:

```
l_addr                  l_lnno

function symtab index   0
physical address        line
physical address        line
...
function symtab index   0
physical address        line
physical address        line
...
```

SEE ALSO

cc(1), sdb(1), a.out(4).

NAME

mnttab - mounted file system table

SYNOPSIS

#include <mnttab.h>

DESCRIPTION

mnttab resides in directory /**etc** and contains a table of devices, mounted by the *mount* (1M) command, in the following structure as defined by **<mnttab.h>** :

```
struct      mnttab {
      char        mt_dev[32];
      char        mt_filsys[32];
      short       mt_ro_flg;
      time_t      mt_time;
};
```

Each entry is 70 bytes in length; the first 32 bytes are the null-padded name of the place where the *special file* is mounted; the next 32 bytes represent the null-padded root name of the mounted special file; the remaining 6 bytes contain the mounted *special file's* read/write permissions and the date on which it was mounted.

SEE ALSO

mount(1M) in the *D-NIX 5.3 Reference Manual.*

setmnt(1M) in the *Administrator's Reference Manual.*

NAME

passwd - password file

DESCRIPTION

passwd contains for each user the following information:

login name
encrypted password
numerical user ID
numerical group ID
GCOS job number, box number, optional GCOS user ID
initial working directory
program to use as shell

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The GCOS field is used only when communicating with that system, and in other installations can contain any desired information. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the shell field is null, the shell itself is used.

This file resides in directory /**etc.** Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user IDs to names.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (**.** , **/** , **0-9** , **A-Z** , **a-z**), except when the password is null, in which case the encrypted password is also null. Password aging is effected for a particular user if his encrypted password in the password file is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.)

The first character of the age, *M* say, denotes the maximum number of weeks for which a password is valid. A user who attempts to login after his password has expired will be forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks which must expire before the password may be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) *M* and *m* have numerical values in the range 0-63 that correspond to the 64-character alphabet shown above (i.e., / = 1 week; **z** = 63 weeks). If *m* = *M* = *0* (derived from the string **.** or **..**) the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the password file). If *m* > *M* (signified, e.g., by the string **./**) only the super-user will be able to change the password.

FILES

/etc/passwd

SEE ALSO

a64l(3C), getpwent(3C), group(4).

login(1), passwd(1) in the *D-NIX 5.3 Reference Manual.*

NAME

pnch - file format for card images

DESCRIPTION

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

NAME

profile - setting up an environment at login time

SYNOPSIS

/etc/profile
$HOME/.profile

DESCRIPTION

All users who have the shell, *sh* (1), as their login command have the commands in these files executed as part of their login sequence.

/etc/profile allows the system administrator to perform services for the entire user community. Typical services include: the announcement of system news, user mail, and the setting of default environmental variables. It is not unusual for */etc/profile* to execute special actions for the **root** login or the *su* (1) command. Computers running outside the Eastern time zone should have the line

```
. /etc/TIMEZONE
```

included early in */etc/profile* (see *timezone* (4)).

The file *$HOME*/.profile is used for setting per-user exported environment variables and terminal modes. The following example is typical (except for the comments):

```
# Make some environment variables global"
export MAIL PATH TERM
# Set file creation mask"
umask 027
# Tell me when new mail comes in
MAIL=/usr/mail/$LOGNAME
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
while :
do echo "terminal: \c"
   read TERM
   if [ -f ${TERMINFO:-/usr/lib/terminfo}/?/$TERM ]
   then break
   elif [ -f /usr/lib/terminfo/?/$TERM ]
   then break
   else echo "invalid term $TERM" 1>&2
   fi
done
# Initialize the terminal and set tabs
# The environmental variable TERM must have been exported
# before the "tput init" command is executed.
tput init
# Set the erase character to backspace
stty erase '^H' echoe
```

FILES

/etc/TIMEZONE	timezone environment
$HOME/.profile	user-specific environment
/etc/profile	system-wide environment

SEE ALSO

terminfo(4), timezone(4), environ(5), term(5).

env(1), tput(1) in the *User's Reference Manual.*

su(1M), login(1), mail(1), sh(1), stty(1), su(1), in the *D-NIX 5.3 Reference Manual.*

NOTES

Care`must be taken in providing system-wide services in */etc/profile*. Personal **.profile** files are better for serving all but the most global needs.

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 */
      ushort      r_type ;    /* relocation type */ } ;
#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.

R_RELBYTE	A direct 8-bit reference to the symbol's virtual address.
R_RELWORD	A direct 16-bit reference to the symbol's virtual address.
R_RELLONG	A direct 32-bit reference to the symbol's virtual address.
R_PCRBYTE	A "PC-relative" 8-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.
R_PCRWORD	A "PC-relative" 16-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.
R_PCRLONG	A "PC-relative" 32-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.

A relocation entry with a symbol index of -1 indicates that the relative difference between the current segment's start address and the program's load address is added to the relocation address.

More relocation types exist for other processors. Equivalent relocation types on different processors have equal values and meanings. New relocation types will be defined (with new values) as they are needed.

Relocation entries are generated automatically by the assembler and automatically used by the link editor. Link editor options exist for both preserving and removing the relocation entries from object files.

SEE ALSO

as(1), ld(1), a.out(4), syms(4).

NAME

sccsfile - format of sccs file

DESCRIPTION

An sccs (Source Code Control System) file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines).

Throughout an sccs file there are lines which begin with the **ASCII SOH** (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character.

Entries of the form **DDDDD** represent a five-digit string (a number between 00000 and 99999).

Each logical part of an sccs file is described in detail below.

Checksum

The checksum is the first line of an sccs file. The form of the line is:

@**hDDDDD**

The value of the checksum is the sum of all characters, except those of the first line. The @**h** provides a *magic number* of (octal) 064001.

Delta table

The delta table consists of a variable number of entries of the form:

```
@s DDDDD/DDDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da/hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
.
.
.
@c <comments> ...
.
.
.
 @e
```

The first line (@**s**) contains the number of lines inserted/deleted/unchanged, respectively. The second line (@**d**) contains the type of the delta (currently, normal: D, and removed: **R**), the sccs ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The **@i**, **@x**, and **@g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The **@m** lines (optional) each contain one **MR** number associated with the delta; the **@c** lines contain comments associated with the delta.

The **@e** line ends the delta table entry.

User names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **@u** and **@U**. An empty list allows anyone to make a delta. Any line starting with a **!** prohibits the succeeding group or user from making deltas.

Flags

Keywords used internally. [See *admin* (1) for more information on their use.] Each flag line takes the form:

```
@f <flag>  <optional text>
```

The following flags are defined:

```
@f t  <type of program>
@f v  <program name>
@f i  <keyword string>
@f b
@f m  <module name>
@f f  <floor>
@f c  <ceiling>
@f d  <default-sid>
@f n
@f j
@f l  <lock-releases>
@f q  <user defined>
@f z  <reserved for use in interfaces>
```

The **t** flag defines the replacement for the %Y% identification keyword. The **v** flag controls prompting for **MR** numbers in addition to comments; if the optional text is present it defines an **MR** number validity checking program. The **i** flag controls the warning/error aspect of the "No id keywords" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a "fatal" error (the file will not be gotten, or the delta will not be made). When the **b** flag is present the **-b** keyletter may be used on the *get* command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the %M% identification keyword. The **f** flag defines the "floor" release; the release below which no deltas may be added. The **c** flag defines the "ceiling" release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a *get* command. The **n** flag causes *delta* to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes *get* to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing [*get* (1) with the **-e** keyletter]. The **q** flag defines the replacement for the %Q% identification keyword. The **z** flag is used in certain specialized interface programs. *Comments* Arbitrary text is surrounded by the bracketing lines **@t** and **@T**. The comments section typically will contain a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: *insert, delete,* and *end,* represented by:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

NAME

scnhdr - section header for a common object file

SYNOPSIS

#include <scnhdr.h>

DESCRIPTION

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct     scnhdr
{
      char             s_name[SYMNMLEN];    /* section name */
      long             s_paddr;             /* physical address */
      long             s_vaddr;             /* virtual address */
      long             s_size;              /* section size */
      long             s_scnptr;            /* file ptr to raw data */
      long             s_relptr;            /* file ptr to relocation */
      long             s_lnnoptr;           /* file ptr to line numbers */
      unsigned short   s_nreloc;            /* # reloc entries */
      unsigned short   s_nlnno;             /* # line number entries */
      long             s_flags;             /* flags */
};
```

File pointers are byte offsets into the file; they can be used as the offset in a call to **FSEEK** [see *ldfcn* (4)]. If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. It has a size, symbols defined in it, and symbols that refer to it. But it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for *s_scnptr, s_relptr, s_lnnoptr, s_nreloc,* and *s_nlnno* are zero.

SEE ALSO

ld(1), fseek(3S), a.out(4).

NAME

scr_dump - format of curses screen image file.

SYNOPSIS

scr_dump (file)

DESCRIPTION

The *curses* (3X) function *scr_dump ()* will copy the contents of the screen into a file. The format of the screen image is as described below.

The name of the *tty* is 20 characters long and the modification time (the *mtime* of the tty that this is an image of) is of the type *time_t*. All other numbers and characters are stored as *chtype* (see **<curses.h>).** No newlines are stored between fields.

```
<magic number:  octal 0433>
<name of tty>
<mod time of tty>
<columns> <lines>
<line length> <chars in line>      for each line on the screen
<line length> <chars in line>
   .
   .
   .
<labels?>   1, if soft screen labels are present
<cursor row> <cursor column>
```

Only as many characters as are in a line will be listed. For example, if the *<line length>* is **0**, there will be no characters following *<line length>*. If *<labels?>* is TRUE, following it will be

```
<number of labels>
<label width>
<chars in label 1>
<chars in label 2>
    .
    .
    .
```

SEE ALSO

curses(3X).

NAME

syms - common object file symbol table format

SYNOPSIS

#include <syms.h>

DESCRIPTION

Common object files contain information to support symbolic software testing [see *sdb* (1)]. Line number entries, *linenum* (4), and extensive symbolic information permit testing at the C *source* level. Every object file's symbol table is organized as shown below.

```
File name 1.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 1.
File name 2.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 2.
...
Defined global symbols.
Undefined global symbols.
```

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure is given below.

```
#define SYMNMLEN        8
#define FILNMLEN        14
#define DIMNUM 4
struct syment
{
        union           /* all ways to get symbol name */
        {
                char    _n_name[SYMNMLEN];  /* symbol name */
                struct
                {
                        long     _n_zeroes;     /* == 0L when in string table */
                        long     _n_offset;     /* location of name in table */
                } _n_n;
                char             *_n_nptr[2];   /* allows overlaying */
        } _n;
        long                     n_value;       /* value of symbol */
        short                    n_scnum;       /* section number */
        unsigned short           n_type;        /* type and derived type */
        char                     n_sclass;      /* storage class */
        char                     n_numaux;      /* number of aux entries */
};
#define n_name                   _n._n_name
#define n_zeroes                 _n._n_n._n_zeroes
```

```
#define n_offset          _n._n_n._n_offset
#define n_nptr            _n._n_nptr[1]
```

Meaningful values and explanations for them are given in both **syms.h** and *Common Object File Format.* Anyone who needs to interpret the entries should seek more information in these sources. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent
{
      struct
      {
            long        x_tagndx;
            union
            {
                  struct
                  {
                        unsigned short  x_lnno;
                        unsigned short  x_size;
                  } x_lnsz;
                  long  x_fsize;
                  } x_misc;
            union
            {
                  struct
                  {
                        long  x_lnnoptr;
                        long  x_endndx;
                  }     x_fcn;
                  struct
                  {
                        unsigned short  x_dimen[DIMNUM];
                  }     x_ary;
            }
            x_fcnary;
            unsigned short  x_tvndx;
      }     x_sym;
      struct
      {
            char  x_fname[FILNMLEN];
      }     x_file;
      struct
      {
            long  x_scnlen;
            unsigned short  x_nreloc;
            unsigned short  x_nlinno;
      }     x_scn;
      struct
      {
            long        x_tvfill;
            unsigned short  x_tvlen;
            unsigned short  x_tvran[2];
      }     x_tv;
};
```

Indexes of symbol table entries begin at *zero.*

SEE ALSO

sdb(1), a.out(4), linenum(4).

WARNINGS

On machines on which **int s** are equivalent to **long s,** all **long s** have their type changed to **int.** Thus the information about which symbols are declared as **long s** and which, as **int s,** does not show up in the symbol table.

NAME

term - format of compiled term file.

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

Compiled *terminfo* (4) descriptions are placed under the directory **/usr/lib/terminfo**. In order to avoid a linear search of a huge UNIX 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, **att4425** can be found in the file **/usr/lib/terminfo/a/att4425.** 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-bit byte is assumed, but no assumptions about byte ordering or sign extension are made. Thus, these binary *terminfo* (4) files can be transported to other hardware with 8-bit bytes.

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**, and the value **-2** is represented by **0376,0377**; other negative values are illegal. Computers where this does not correspond to the hardware read the integers as two bytes and compute the result, making the compiled entries portable between machine types. The **-1** generally means that a capability is missing from this terminal. The **-2** means that the capability has been cancelled in the *terminfo* (4) source and also is to be considered missing.

The compiled file is created from the source file descriptions of the terminals (see the **-I** option of *infocmp* (1M)) by using the *terminfo* (4) compiler, *tic* (1M), and read by the routine **setupterm ().** (See *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.

The terminal names section comes next. It contains the first line of the *terminfo* (4) description, listing the various names for the terminal, separated by the bar (|) character (see *term* (5)). The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either **0** or **1** as the flag is present or absent. The value of **2** means that the flag has been cancelled. The capabilities are in the same order as the file **< term.h >.**

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the boolean flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is **-1** or **-2**, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of **-1** or **-2** means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^x or \c notation are stored in their interpreted form, not the printing representation. Padding information ($<nn>) and parameter information (%x) are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for **setupterm ()** to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm ()** has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm ()** must be prepared for both possibilities - this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the AT&T Model 37 KSR is included:

```
37|tty37|AT&T model 37 teletype,
   hc, os, xon,
   bel=^G, cr=\r, cubl=\b, cudl=\n, cuul=\E7, hd=\E9,
   hu=\E8, ind=\n,

0000 032 001      \0 032  \0 013  \0 021 001   3  \0   3   7   |   t
0020   t   y   3   7   |   A   T   &   T       m   o   d   e   l
0040   3   7       t   e   l   e   t   y   p   e  \0  \0  \0  \0  \0
0060  \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0 001  \0  \0  \0  \0
0100 001  \0  \0  \0  \0  \0 377 377 377 377 377 377 377 377 377 377
0120 377 377 377 377 377 377 377 377 377 377 377 377 377 377   &  \0
0140      \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0160 377 377   "  \0 377 377 377 377   (  \0 377 377 377 377 377 377
0200 377 377   0  \0 377 377 377 377 377 377 377 377   -  \0 377 377
0220 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0520 377 377 377 377 377 377 377 377 377 377 377 377 377 377   $  \0
0540 377 377 377 377 377 377 377 377 377 377 377 377 377 377   *  \0
0560 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
1160 377 377 377 377 377 377 377 377 377 377 377 377 377 377   3   7
1200   |   t   t   y   3   7   |   A   T   &   T       m   o   d   e
1220   l       3   7       t   e   l   e   t   y   p   e  \0  \r  \0
1240  \n  \0  \n  \0 007  \0  \b  \0 033   8  \0 033   9  \0 033   7
1260  \0  \0
1261
```

Some limitations: total compiled entries cannot exceed 4096 bytes; all entries in the name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/include/term.h	*terminfo* (4) header file

SEE ALSO

curses(3X), terminfo(4), term(5).

infocmp(1M) in the *Administrator's Reference Manual.*

BUGS

For compatability reasons, the terminfo database on a DS90 is stored with most significant byte first and least significant byte last.

NAME

terminfo - terminal capability data base

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

terminfo is a compiled database (see *tic* (1M)) describing the capabilities of terminals. Terminals are described in *terminfo* source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, such as *vi* (1) and *curses* (3X), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the **-I** option of *infocmp* (1M).

Entries in *terminfo* source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the *terminfo* database gives the name by which *terminfo* knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to use to set the environment variable TERM in **$HOME/.profile** ; see *profile* (4)), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. See *term* (5) for examples and more information on choosing names and synonyms.

CAPABILITIES

In the table below, the **Variable** is the name by which the C programmer (at the *terminfo* level) accesses the capability. The **Capname** is the short name for this variable used in the text of the database. It is used by a person updating the database and by the *tput* (1) command when asking what the value of the capability is for a particular terminal. The **Termcap Code** is a two-letter code that 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. 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.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the table below, have names beginning with **key_**. The following indicators may appear at the end of the **Description** for a variable.

(G)	indicates that the string is passed through **tparm**() with parameters (parms) as given (#*i*).
(*)	indicates that padding may be based on the number of lines affected.
(#*i*)	indicates the i^{th} *parameter.*

Booleans:

Variable	Cap-name	Termcap Code	Description
auto_left_margin	bw	bw	**cub1** wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
no_esc_ctlc	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
hard_cursor	chts	HC	Cursor is hard to see.
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
needs_xon_xoff	nxon	nx	Padding won't work, xon/xoff required
non_rev_rmcup	nrrmc	NR	**smcup** does not reverse **rmcup**
no_pad_char	npc	NP	Pad character doesn't exist
over_strike	os	os	Terminal overstrikes on hard-copy terminal
prtr_silent	mc5i	5i	Printer won't echo on screen.
status_line_esc_ok	eslok	es	Escape can be used on the status line
dest_tabs_magic_smso	xt	xt	Destructive tabs, magic **smso** char (t1061)
tilde_glitch	hz	hz	Hazeltine; can't print tildes(~)
transparent_underline	ul	ul	Underline character overstrikes
xon_xoff	xon	xo	Terminal uses xon/xoff handshaking

Numbers:

Variable	Cap-name	Termcap Code	Description
columns	cols	co	Number of columns in a line
init_tabs	it	it	Tabs initially every # spaces.
label_height	lh	lh	Number of rows in each label
label_width	lw	lw	Number of cols in each label
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 blank chars left by **smso** or **rmso**
num_labels	nlab	Nl	Number of labels on screen (start at 1)
padding_baud_rate	pb	pb	Lowest baud rate where padding needed
virtual_terminal	vt	vt	Virtual terminal number (UNIX system)
width_status_line	wsl	ws	Number of columns in status line

Strings:

acs_chars	acsc	ac	Graphic charset pairs aAbBcC - def=vt100+
back_tab	cbt	bt	Back tab
bell	bel	bl	Audible signal (bell)
carriage_return	cr	cr	Carriage return (*)
change_scroll_region	csr	cs	Change to lines #1 thru #2 (vt100) (G)
char_padding	rmp	rP	Like ip but when in replace mode
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear left and right soft margins
clear_screen	clear	cl	Clear screen and home cursor (*)
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display (*)
column_address	hpa	ch	Horizontal position absolute (G)
command_character	cmdch	CC	Term. settable cmd char in prototype
cursor_address	cup	cm	Cursor motion to row #1 col #2 (G)
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 (G)
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 (*)
delete_line	dl1	dl	Delete line (*)
dis_status_line	dsl	ds	Disable status line
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate char set
enter_alt_charset_mod	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins
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
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 standout mode
enter_underline_mode	smul	us	Start underscore mode
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking
erase_chars	ech	ec	Erase #1 characters (G)
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
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 standout mode
exit_underline_mode	rmul	ue	End underscore mode
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking
flash_screen	flash	vb	Visible bell (may not move cursor)
form_feed	ff	ff	Hardcopy terminal page eject (*)
from_status_line	fsl	fs	Return from status line
init_1string	is1	i1	Terminal initialization string

init_2string	is2	is	Terminal initialization string
init_3string	is3	i3	Terminal initialization string
init_file	if	if	Name of initialization file containing **is**
init_prog	iprog	iP	Path name of program for init.
insert_character	ich1	ic	Insert character
insert_line	il1	al	Add new blank line (*)
insert_padding	ip	ip	Insert pad after character inserted (*)
key_a1	ka1	K1	KEY_A1, 0534, Upper left of keypad
key_a3	ka3	K3	KEY_A3, 0535, Upper right of keypad
key_b2	kb2	K2	KEY_B2, 0536, Center of keypad
key_backspace	kbs	kb	KEY_BACKSPACE, 0407, Sent by backspace key
key_beg	kbeg	@1	KEY_BEG, 0542, Sent by beg(inning) key
key_btab	kcbt	kB	KEY_BTAB, 0541, Sent by back-tab key
key_c1	kc1	K4	KEY_C1, 0537, Lower left of keypad
key_c3	kc3	K5	KEY_C3, 0540, Lower right of keypad
key_cancel	kcan	@2	KEY_CANCEL, 0543, Sent by cancel key
key_catab	ktbc	ka	KEY_CATAB, 0526, Sent by clear-all-tabs key
key_clear	kclr	kC	KEY_CLEAR, 0515, Sent by clear-screen or erase key
key_close	kclo	@3	KEY_CLOSE, 0544, Sent by close key
key_command	kcmd	@4	KEY_COMMAND, 0545, Sent by cmd (command) key
key_copy	kcpy	@5	KEY_COPY, 0546, Sent by copy key
key_create	kcrt	@6	KEY_CREATE, 0547, Sent by create key
key_ctab	kctab	kt	KEY_CTAB, 0525, Sent by clear-tab key
key_dc	kdch1	kD	KEY_DC, 0512, Sent by delete-character key
key_dl	kdl1	kL	KEY_DL, 0510, Sent by delete-line key
key_down	kcud1	kd	KEY_DOWN, 0402, Sent by terminal down-arrow key
key_eic	krmir	kM	KEY_EIC, 0514, Sent by **rmir** or **smir** in insert mode
key_end	kend	@7	KEY_END, 0550, Sent by end key
key_enter	kent	@8	KEY_ENTER, 0527, Sent by enter/send key
key_eol	kel	kE	KEY_EOL, 0517, Sent by clear-to-end-of-line key
key_eos	ked	kS	KEY_EOS, 0516, Sent by clear-to-end-of-screen key
key_exit	kext	@9	KEY_EXIT, 0551, Sent by exit key
key_f0	kf0	k0	KEY_F(0), 0410, Sent by function key f0
key_f1	kf1	k1	KEY_F(1), 0411, Sent by function key f1
key_f2	kf2	k2	KEY_F(2), 0412, Sent by function key f2
key_f3	kf3	k3	KEY_F(3), 0413, Sent by function key f3
key_f4	kf4	k4	KEY_F(4), 0414, Sent by function key f4
key_f5	kf5	k5	KEY_F(5), 0415, Sent by function key f5
key_f6	kf6	k6	KEY_F(6), 0416, Sent by function key f6
key_f7	kf7	k7	KEY_F(7), 0417, Sent by function key f7
key_f8	kf8	k8	KEY_F(8), 0420, Sent by function key f8
key_f9	kf9	k9	KEY_F(9), 0421, Sent by function key f9
key_f10	kf10	k;	KEY_F(10), 0422, Sent by function key f10
key_f11	kf11	F1	KEY_F(11), 0423, Sent by function key f11
key_f12	kf12	F2	KEY_F(12), 0424, Sent by function key f12
key_f13	kf13	F3	KEY_F(13), 0425, Sent by function key f13
key_f14	kf14	F4	KEY_F(14), 0426, Sent by function key f14
key_f15	kf15	F5	KEY_F(15), 0427, Sent by function key f15
key_f16	kf16	F6	KEY_F(16), 0430, Sent by function key f16
key_f17	kf17	F7	KEY_F(17), 0431, Sent by function key f17
key_f18	kf18	F8	KEY_F(18), 0432, Sent by function key f18
key_f19	kf19	F9	KEY_F(19), 0433, Sent by function key f19
key_f20	kf20	FA	KEY_F(20), 0434, Sent by function key f20
key_f21	kf21	FB	KEY_F(21), 0435, Sent by function key f21
key_f22	kf22	FC	KEY_F(22), 0436, Sent by function key f22

key_f23	kf23	FD	KEY_F(23), 0437, Sent by function key f23
key_f24	kf24	FE	KEY_F(24), 0440, Sent by function key f24
key_f25	kf25	FF	KEY_F(25), 0441, Sent by function key f25
key_f26	kf26	FG	KEY_F(26), 0442, Sent by function key f26
key_f27	kf27	FH	KEY_F(27), 0443, Sent by function key f27
key_f28	kf28	FI	KEY_F(28), 0444, Sent by function key f28
key_f29	kf29	FJ	KEY_F(29), 0445, Sent by function key f29
key_f30	kf30	FK	KEY_F(30), 0446, Sent by function key f30
key_f31	kf31	FL	KEY_F(31), 0447, Sent by function key f31
key_f32	kf32	FM	KEY_F(32), 0450, Sent by function key f32
key_f33	kf33	FN	KEY_F(33), 0451, Sent by function key f33
key_f34	kf34	FO	KEY_F(34), 0452, Sent by function key f34
key_f35	kf35	FP	KEY_F(35), 0453, Sent by function key f35
key_f36	kf36	FQ	KEY_F(36), 0454, Sent by function key f36
key_f37	kf37	FR	KEY_F(37), 0455, Sent by function key f37
key_f38	kf38	FS	KEY_F(38), 0456, Sent by function key f38
key_f39	kf39	FT	KEY_F(39), 0457, Sent by function key f39
key_f40	kf40	FU	KEY_F(40), 0460, Sent by function key f40
key_f41	kf41	FV	KEY_F(41), 0461, Sent by function key f41
key_f42	kf42	FW	KEY_F(42), 0462, Sent by function key f42
key_f43	kf43	FX	KEY_F(43), 0463, Sent by function key f43
key_f44	kf44	FY	KEY_F(44), 0464, Sent by function key f44
key_f45	kf45	FZ	KEY_F(45), 0465, Sent by function key f45
key_f46	kf46	Fa	KEY_F(46), 0466, Sent by function key f46
key_f47	kf47	Fb	KEY_F(47), 0467, Sent by function key f47
key_f48	kf48	Fc	KEY_F(48), 0470, Sent by function key f48
key_f49	kf49	Fd	KEY_F(49), 0471, Sent by function key f49
key_f50	kf50	Fe	KEY_F(50), 0472, Sent by function key f50
key_f51	kf51	Ff	KEY_F(51), 0473, Sent by function key f51
key_f52	kf52	Fg	KEY_F(52), 0474, Sent by function key f52
key_f53	kf53	Fh	KEY_F(53), 0475, Sent by function key f53
key_f54	kf54	Fi	KEY_F(54), 0476, Sent by function key f54
key_f55	kf55	Fj	KEY_F(55), 0477, Sent by function key f55
key_f56	kf56	Fk	KEY_F(56), 0500, Sent by function key f56
key_f57	kf57	Fl	KEY_F(57), 0501, Sent by function key f57
key_f58	kf58	Fm	KEY_F(58), 0502, Sent by function key f58
key_f59	kf59	Fn	KEY_F(59), 0503, Sent by function key f59
key_f60	kf60	Fo	KEY_F(60), 0504, Sent by function key f60
key_f61	kf61	Fp	KEY_F(61), 0505, Sent by function key f61
key_f62	kf62	Fq	KEY_F(62), 0506, Sent by function key f62
key_f63	kf63	Fr	KEY_F(63), 0507, Sent by function key f63
key_find	kfnd	@0	KEY_FIND, 0552, Sent by find key
key_help	khlp	%1	KEY_HELP, 0553, Sent by help key
key_home	khome	kh	KEY_HOME, 0406, Sent by home key
key_ic	kich1	kI	KEY_IC, 0513, Sent by ins-char/enter ins-mode key
key_il	kil1	kA	KEY_IL, 0511, Sent by insert-line key
key_left	kcub1	kl	KEY_LEFT, 0404, Sent by terminal left-arrow key
key_ll	kll	kH	KEY_LL, 0533, Sent by home-down key
key_mark	kmrk	%2	KEY_MARK, 0554, Sent by mark key
key_message	kmsg	%3	KEY_MESSAGE, 0555, Sent by message key
key_move	kmov	%4	KEY_MOVE, 0556, Sent by move key
key_next	knxt	%5	KEY_NEXT, 0557, Sent by next-object key
key_npage	knp	kN	KEY_NPAGE, 0522, Sent by next-page key
key_open	kopn	%6	KEY_OPEN, 0560, Sent by open key
key_options	kopt	%7	KEY_OPTIONS, 0561, Sent by options key
key_ppage	kpp	kP	KEY_PPAGE, 0523, Sent by previous-page key

key_previous	kprv	%8	KEY_PREVIOUS, 0562, Sent by previous-object key
key_print	kprt	%9	KEY_PRINT, 0532, Sent by print or copy key
key_redo	krdo	%0	KEY_REDO, 0563, Sent by redo key
key_reference	kref	&1	KEY_REFERENCE, 0564, Sent by ref(erence) key
key_refresh	krfr	&2	KEY_REFRESH, 0565, Sent by refresh key
key_replace	krpl	&3	KEY_REPLACE, 0566, Sent by replace key
key_restart	krst	&4	KEY_RESTART, 0567, Sent by restart key
key_resume	kres	&5	KEY_RESUME, 0570, Sent by resume key
key_right	kcuf1	kr	KEY_RIGHT, 0405, Sent by terminal right-arrow key
key_save	ksav	&6	KEY_SAVE, 0571, Sent by save key
key_sbeg	kBEG	&9	KEY_SBEG, 0572, Sent by shifted beginning key
key_scancel	kCAN	&0	KEY_SCANCEL, 0573, Sent by shifted cancel key
key_scommand	kCMD	*1	KEY_SCOMMAND, 0574, Sent by shifted command key
key_scopy	kCPY	*2	KEY_SCOPY, 0575, Sent by shifted copy key
key_screate	kCRT	*3	KEY_SCREATE, 0576, Sent by shifted create key
key_sdc	kDC	*4	KEY_SDC, 0577, Sent by shifted delete-char key
key_sdl	kDL	*5	KEY_SDL, 0600, Sent by shifted delete-line key
key_select	kslt	*6	KEY_SELECT, 0601, Sent by select key
key_send	kEND	*7	KEY_SEND, 0602, Sent by shifted end key
key_seol	kEOL	*8	KEY_SEOL, 0603, Sent by shifted clear-line key
key_sexit	kEXT	*9	KEY_SEXIT, 0604, Sent by shifted exit key
key_sf	kind	kF	KEY_SF, 0520, Sent by scroll-forward/down key
key_sfind	kFND	*0	KEY_SFIND, 0605, Sent by shifted find key
key_shelp	kHLP	#1	KEY_SHELP, 0606, Sent by shifted help key
key_shome	kHOM	#2	KEY_SHOME, 0607, Sent by shifted home key
key_sic	kIC	#3	KEY_SIC, 0610, Sent by shifted input key
key_sleft	kLFT	#4	KEY_SLEFT, 0611, Sent by shifted left-arrow key
key_smessage	kMSG	%a	KEY_SMESSAGE, 0612, Sent by shifted message key
key_smove	kMOV	%b	KEY_SMOVE, 0613, Sent by shifted move key
key_snext	kNXT	%c	KEY_SNEXT, 0614, Sent by shifted next key
key_soptions	kOPT	%d	KEY_SOPTIONS, 0615, Sent by shifted options key
key_sprevious	kPRV	%e	KEY_SPREVIOUS, 0616, Sent by shifted prev key
key_sprint	kPRT	%f	KEY_SPRINT, 0617, Sent by shifted print key
key_sr	kri	kR	KEY_SR, 0521, Sent by scroll-backward/up key
key_sredo	kRDO	%g	KEY_SREDO, 0620, Sent by shifted redo key
key_sreplace	kRPL	%h	KEY_SREPLACE, 0621, Sent by shifted replace key
key_sright	kRIT	%i	KEY_SRIGHT, 0622, Sent by shifted right-arrow key
key_srsume	kRES	%j	KEY_SRSUME, 0623, Sent by shifted resume key
key_ssave	kSAV	!1	KEY_SSAVE, 0624, Sent by shifted save key
key_ssuspend	kSPD	!2	KEY_SSUSPEND, 0625, Sent by shifted suspend key
key_stab	khts	kT	KEY_STAB, 0524, Sent by set-tab key
key_sundo	kUND	!3	KEY_SUNDO, 0626, Sent by shifted undo key
key_suspend	kspd	&7	KEY_SUSPEND, 0627, Sent by suspend key
key_undo	kund	&8	KEY_UNDO, 0630, Sent by undo key
key_up	kcuu1	ku	KEY_UP, 0403, 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_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
lab_f10	lf10	la	Labels on function key f10 if not f10
label_off	rmln	LF	Turn off soft labels
label_on	smln	LO	Turn on soft labels
meta_off	rmm	mo	Turn off "meta mode"
meta_on	smm	mm	Turn on "meta mode" (8th bit)
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 (G*)
parm_delete_line	dl	DL	Delete #1 lines (G*)
parm_down_cursor	cud	DO	Move cursor down #1 lines. (G*)
parm_ich	ich	IC	Insert #1 blank chars (G*)
parm_index	indn	SF	Scroll forward #1 lines. (G)
parm_insert_line	il	AL	Add #1 new blank lines (G*)
parm_left_cursor	cub	LE	Move cursor left #1 spaces (G)
parm_right_cursor	cuf	RI	Move cursor right #1 spaces. (G*)
parm_rindex	rin	SR	Scroll backward #1 lines. (G)
parm_up_cursor	cuu	UP	Move cursor up #1 lines. (G*)
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
plab_norm	pln	pn	Prog label #1 to show string #2
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	pO	Turn on the printer for #1 bytes
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 (G*)
req_for_input	rfi	RF	Send next input char (for ptys)
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 (G)
save_cursor	sc	sc	Save cursor position.
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	sr	Scroll text down
set_attributes	sgr	sa	Define the video attributes #1-#9 (G)
set_left_margin	smgl	ML	Set soft left margin
set_right_margin	smgr	MR	Set soft right margin
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 (G)
tab	ht	ta	Tab to next 8 space hardware tab stop.
to_status_line	tsl	ts	Go to status line, col #1 (G)
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
xoff_character	xoffc	XF	X-off character
xon_character	xonc	XN	X-on character

SAMPLE ENTRY

The following entry, which describes the *Concept*-100 terminal, is among the more complex entries in the *terminfo* file as of this writing.

```
concept100|c100|concept|c104|c100-4p|concept 100,
    am, db, eo, in, mir, ul, xenl,
    cols#80, lines#24, pb#9600, vt#8,
    bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>,
    cnorm=\Ew, cr=^M$<9>, cubl=^H, cudl=^J,
    cufl=\E=, cup=\Ea%pl%' '%+%c%p2%' '%+%c,
    cuul=\E;, cvvis=\EW, dchl=\E^A$<16*>, dim=\EE,
    dl1=\E^B$<3*>, ed=\E^C$<16*>, el=\E^U$<16>,
    flash=\Ek$<20>\EK, ht=\t$<8>, ill=\E^R$<3*>,
    ind=^J, .ind=^J$<9>, ip=$<16*>,
    is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\0\Eo&\0\Eo\47\E,
    kbs=^h, kcubl=\E>, kcudl=\E<, kcufl=\E=, kcuul=\E;,
    kfl=\E5, kf2=\E6, kf3=\E7, khome=\E?,
    prot=\EI, rep=\Er%pl%c%p2%' '%+%c$<.2*>,
    rev=\ED, rmcup=\Ev\s\s\s\s$<6>\Ep\r\n,
    rmir=\E\0, rmkx=\Ex, rmso=\Ed\Ee, rmul=\Eg,
    rmul=\Eg, sgr0=\EN\0, smcup=\EU\Ev\s\s8p\Ep\r,
    smir=\E^P, smkx=\EX, smso=\EE\ED, smul=\EG,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Lines beginning with "#" are taken as comment lines. 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 particular features, 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*. The value may be specified in decimal, octal or hexadecimal using normal C conventions.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the two- to five-character capname, an '=', and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in **$<..>** brackets, as in **el=\eEK$<3>**, and padding characters are supplied by **tputs ()** (see *curses* (3X)) to provide this delay. The delay can be either a number, e.g., **20**, or a number followed by an '*****' (i.e., **3***), a '/' (i.e., 5/), or both (i.e., **10*/**). 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 **in** 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 '/' indicates that the padding is mandatory. Otherwise, if the terminal has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon.**

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,** and **\s** give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: **\^** for caret (^); \ for backslash (\); **\,** for comma (,); **\:** for colon (:); and **\0** for null. (**\0** will actually 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 backslash (e.g., \123).

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. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

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* (1) 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 the inability of *vi* (1) to work with that terminal. To test a new terminal description, 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 correct (if the terminal manufacturer did not document it) a severe test is to comment out **xon** , edit a large file at 9600 baud with *vi* (1), delete 16 or so lines from the middle of the screen, then hit the **u** key several times quickly. If the display is corrupted, 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 has a screen, 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 the terminal uses the xon-xoff flow-control protocol, like most terminals, specify **xon.**

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**=\s" 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 screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and should 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 screen 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|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 in the manner of a Reverse Polish Notation (postfix) calculator. 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. Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use **%gx%{5}%-**.

The % encodings have the following meanings:

%%	outputs '%'
%[[:]*flags*][*width*[.*precision*]][**doxXs**]	as in printf, flags are **[-+#]** and space
%c	print pop() gives %c
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to pop()
%g[a-z]	get variable [a-z] and push it
%'*c*'	push char constant *c*
%{*nn*}	push decimal constant *nn*
%l	push strlen(pop())
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop() op pop())
%& %| %^	bit operations: push(pop() op pop())
%= %> %<	logical operations: push(pop() op pop())
%A %O	logical operations: and, or
%! %~	unary operations: push(op pop())
%i	(for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present
%? 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 b_5 %; c_i are conditions, b_i are bodies.

If the "-" flag is used with "%[doxXs]", then a colon (:) must be placed between the "%" and the "-" to differentiate the flag from the binary "%-" operator, .e.g "%:-16.16s".

Consider the Hewlett-Packard 2645, 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 zero-padded as two digits. Thus its **cup** capability is "**cup=**\E&a%p2%2.2dc%p1%2.2dY$<6>".

The Micro-Term 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%'\s'%+%c%p2%'\s'%+%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.

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 Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

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 Hewlett-Packard 2645) 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.

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 beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. 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 destructive 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.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

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 a full screen 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 operations 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 versus 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 padding 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.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp.**

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* (see *curses* (3X)), 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; however, different users have different preferences on different terminals.) 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 Micro-Term 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 a command is necessary before alternate character set mode is entered, give the sequence in **enacs** (enable alternate-character-set mode).

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr,** only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

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 Hewlett-Packard 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. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

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.** The boolean **chts** should also be given. 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 either of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the *Concept* with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the Tektronix 4025, where **smcup** sets the command character to be the one used by **terminfo.** If the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability **ul**. For terminals where a character overstriking another leaves both characters on the screen, give the capability **os.** If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Example of highlighting: assume that the terminal under question needs the following escape sequences to turn on various modes.

tparm parameter	**attribute**	**escape sequence**
	none	\E[0m
p1	standout	\E[0;4;7m
p2	underline	\E[0;3m
p3	reverse	\E[0;4m
p4	blink	\E[0;5m
p5	dim	\E[0;7m
p6	bold	\E[0;3;4m
p7	invis	\E[0;8m
p8	protect	not available
p9	altcharset	^O (off) ^N(on)

Note that each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, since this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m.** The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N** depending on whether it is off or on. If all modes were to be turned on, the sequence would be **\E[0;3;4;5;7;8m^N.**

Now look at when different sequences are output. For example, **;3** is output when either **p2** or **p6** is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

sequence	**when to output**	**terminfo translation**
\E[0	always	\E[0
;3	if p2 or p6	%?%p2%p6%|%t;3%;
;4	if p1 or p3 or p6	%?%p1%p3%|%p6%|%t;4%;
;5	if p4	%?%p4%t;5%;
;7	if p1 or p5	%?%p1%p5%|%t;7%;
;8	if p7	%?%p7%t;8%;
m	always	m
^N or ^O	if p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the **sgr** sequence gives:

sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p5%t;
5%;%?%p1%p5% |%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1, kcuf1, kcuu1, kcud1,** and**khome** respectively. If there are function keys such as f0, f1, ..., f63, the codes they send can be given as **kf0, kf1, ..., kf63**. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ..., lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1** , **ka3** , **kb2** , **kc1** , and **kc3.** These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be given as **pfkey** , **pfloc** , and **pfx.** A string to program their soft-screen labels can be given as **pln.** Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local mode; and **pfx** causes the string to be transmitted to the computer. The capabilities **nlab** , **lw** and **lh** define how many soft labels there are and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln. smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A "backtab" command which moves leftward to the next tab stop can be given as **cbt.** By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by **tput init** (see *tput* (1)) to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the *terminfo* description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1, is2,** and **is3,** initialization strings for the terminal; **iprog,** the path name of a program to be run to initialize the terminal; and **if,** the name of a file containing long initialization strings.

These strings are expected to set the terminal into modes consistent with the rest of the *terminfo* description. They must be sent to the terminal each time the user logs in and be output in the following order: run the program **iprog;** output **is1;** output **is2;** set the margins using **mgc, smgl** and **smgr;** set the tabs using **tbc** and **hts;** print the file **if;** and finally output **is3.** This is usually done using the **init** option of *tput* (1); see *profile* (4).

Most initialization is done with **is2.** Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3.** Sequences that do a harder reset from a totally unknown state can be given as **rs1, rs2, rf,** and **rs3,** analogous to **is1, is2, is3,** and **if.** (The method using files, **if** and **rf,** is used for a few terminals, from **/usr/lib/tabset/*;** however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput reset,** which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1, rs2, rs3,** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2,** but on some terminals it causes an annoying glitch on the screen and is not normally needed since the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if.**

If there are commands to set and clear margins, they can be given as **mgc** (clear all margins), **smgl** (set left margin), and **smgr** (set right margin).

Delays

Certain capabilities control padding in the *tty* (7) driver. These are primarily needed by hard-copy terminals, and are used by **tput init** to set tty modes appropriately. Delays embedded in the capabilities **cr, ind, cub1, ff,** and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb.**

Status Lines

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl.** (**fsl** must leave the cursor position in the same place it was before **tsl.** If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl.** If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc.** The status line is normally assumed to be the same width as the rest of the screen, e.g., **cols.** If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl.**

Line Graphics

If the terminal has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc.** The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	**vt100+ character**
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	-
diamond	‘
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee	t
right tee	u
bottom tee	v
top tee	w
vertical line	x
bullet	~

The best way to describe a new terminal's line graphics set is to add a third column to the above table with the characters for the new terminal that produce the appropriate glyph when the terminal is in the alternate character set mode. For example,

glyph name	**vt100+ char**	**new tty char**
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in "**acsc**=lRmFkTjGq\,x.".

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep.** The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **tparm(repeat_char, 'x', 10)** is the same as **xxxxxxxxxx.**

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch.** A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some UNIX systems: If the environment variable CC exists, all occurrences of the prototype character are replaced with the character in CC.

Terminal descriptions that do not represent a specific kind of known terminal, such as **switch, dialup, patch,** and **network,** should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to **virtual** terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt.** A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi.**

If the terminal uses xon/xoff handshaking for flow control, give **xon.** Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon.** If the characters used for handshaking are not ^S and ^Q , they may be specified with **xonc** and **xoffc.**

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km.** Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm.**

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm.** A value of **lm #0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0:** print the contents of the screen, **mc4:** turn off the printer, and **mc5:** turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, **mc5p,** takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4,** is transparently passed to the printer while an **mc5p** is in effect.

Special Cases

The working model used by *terminfo* fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by *terminfo.* These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the *terminfo* model implemented.

Terminals which can not display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals which ignore a linefeed immediately after an **am** wrap, such as the *Concept* 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie" therefore, to erase standout mode, it is instead necessary to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control-c characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for control-c.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use.** A capability can be canceled by placing *xx@* to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype\04424 in display function group ii,
        rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/lib/.CORE\term/?/*	subset of compiled terminal description database
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

SEE ALSO

curses(3X), printf(3S), term(5).

captoinfo(1M), infocmp(1M), tic(1M), tty(7) in the *Administrator's Reference Manual".*

tput(1) in the *User's Reference Manual.*

WARNING

As described in the "Tabs and Initialization" section above, a terminal's initialization strings, **is1** , **is2** , and **is3** , if defined, must be output before a *curses* (3X) program is run. An available mechanism for outputting such strings is **tput init** (see *tput* (1) and *profile* (4)).

Tampering with entries in **/usr/lib/.CORE/term/?/*** or **/usr/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs such as *vi* (1) that expect the entry to be present and correct. In particular, removing the description for the "dumb" terminal will cause unexpected problems.

NOTE

The *termcap* database (from earlier releases of UNIX System V) may not be supplied in future releases.

NAME

timezone - set default system time zone

SYNOPSIS

/etc/timezone

DESCRIPTION

This file sets and exports the time zone environmental variable TZ.

This file is "dotted" into other files that must know the time zone.

EXAMPLES

/etc/timezone for the east coast:

```
# Time Zone
TZ=EST5EDT
export TZ
```

SEE ALSO

ctime(3C), profile(4).

NAME

unistd - file header for symbolic constants

SYNOPSIS

#include <unistd.h>

DESCRIPTION

The header file *<unistd.h>* lists the symbolic constants and structures not already defined or declared in some other header file.

```
/* Symbolic constants for the "access" routine: */
#define R_OK        4    /*Test for Read permission */
#define W_OK        2    /*Test for Write permission */
#define X_OK        1    /*Test for eXecute permission */
#define F_OK        0    /*Test for existence of File */
#define F_ULOCK     0    /*Unlock a previously locked region */
#define F_LOCK      1    /*Lock a region for exclusive use */
#define F_TLOCK     2    /*Test and lock a region for exclusive use */
#define F_TEST      3    /*Test a region for other processes locks */

/*Symbolic constants for the "lseek" routine: */
#define SEEK_SET    0    /* Set file pointer to "offset" */
#define SEEK_CUR    1    /* Set file pointer to current plus "offset" */
#define SEEK_END    2    /* Set file pointer to EOF plus "offset" */

/*Pathnames:*/
#define GF_PATH     /etc/group     /*Pathname of the group file */
#define PF_PATH     /etc/passwd    /*Pathname of the passwd file */
```

NAME

utmp, wtmp - utmp and wtmp entry formats

SYNOPSIS

#include <sys/types.h>
#include <utmp.h>

DESCRIPTION

These files, which hold user and accounting information for such commands as *who* (1), *write* (1), and *login* (1), have the following structure as defined by **<utmp.h>** :

```
#define    UTMP_FILE      "/etc/utmp"
#define    WTMP_FILE      "/etc/wtmp"
#define    ut_name        ut_user

struct     utmp {
      char       ut_user[8];             /* User login name */
      char       ut_id[4];               /* /etc/inittab id (usually line #) */
      char       ut_line[12];            /* device name (console, lnxx) */
      short      ut_pid;                 /* process id */
      short      ut_type;                /* type of entry */
      struct     exit_status {
            short      e_termination;    /* Process termination status */
            short      e_exit;           /* Process exit status */
      } ut_exit;                         /* The exit status of a process
                                          * marked as DEAD_PROCESS. */
      time_t     ut_time;                /* time entry was made */
};
/*  Definitions for ut_type  */
#define    EMPTY            0
#define    RUN_LVL          1
#define    BOOT_TIME        2
#define    OLD_TIME         3
#define    NEW_TIME         4
#define    INIT_PROCESS     5      /* Process spawned by "init" */
#define    LOGIN_PROCESS    6      /* A "getty" process waiting for login */
#define    USER_PROCESS     7      /* A user process */
#define    DEAD_PROCESS     8
#define    ACCOUNTING       9
#define    UTMAXTYPE        ACCOUNTING       /* Largest legal value of
                                                ut_type */

/*  Special strings or formats used in the "ut_line" field when  */
/*  accounting for something other than a process  */
/*  No string for the ut_line field can be more than 11 chars +  */
/*  a NULL in length  */
#define    RUNLVL_MSG      "run-level %c"
#define    BOOT_MSG        "system boot"
#define    OTIME_MSG       "old time"
#define    NTIME_MSG       "new time"
```

FILES

/etc/utmp
/etc/wtmp

SEE ALSO

getut(3C).

login(1), who(1), write(1) in the *D-NIX 5.3 Reference Manual.*

3

NAME

intro - introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

NAME

ascii - map of ASCII character set

DESCRIPTION

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

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel |
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si  |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb |
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us  |
|040 sp |041 !  |042 "  |043 #  |044 $  |045 %  |046 &  |047 '   |
|050 (  |051 )  |052 *  |053 +  |054 ,  |055 -  |056 .  |057 /   |
|060 0  |061 1  |062 2  |063 3  |064 4  |065 5  |066 6  |067 7   |
|070 8  |071 9  |072 :  |073 ;  |074 <  |075 =  |076 >  |077 ?   |
|100 @  |101 A  |102 B  |103 C  |104 D  |105 E  |106 F  |107 G   |
|110 H  |111 I  |112 J  |113 K  |114 L  |115 M  |116 N  |117 O   |
|120 P  |121 Q  |122 R  |123 S  |124 T  |125 U  |126 V  |127 W   |
|130 X  |131 Y  |132 Z  |133 [  |134 \  |135 ]  |136 ^  |137 _   |
|140 `  |141 a  |142 b  |143 c  |144 d  |145 e  |146 f  |147 g   |
|150 h  |151 i  |152 j  |153 k  |154 l  |155 m  |156 n  |157 o   |
|160 p  |161 q  |162 r  |163 s  |164 t  |165 u  |166 v  |167 w   |
|170 x  |171 y  |172 z  |173 {  |174 |  |175 }  |176 ~  |177 del |


|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel |
|008 bs |009 ht |00a nl |00b vt |00c np |00d cr |00e so |00f si  |
|010 dle|011 dc1|012 dc2|013 dc3|014 dc4|015 nak|016 syn|017 etb |
|018 can|019 em |01a sub|01b esc|01c fs |01d gs |01e rs |01f us  |
|020 sp |021 !  |022 "  |023 #  |024 $  |025 %  |026 &  |027 '   |
|028 (  |029 )  |02a *  |02b +  |02c ,  |02d -  |02e .  |02f /   |
|030 0  |031 1  |032 2  |033 3  |034 4  |035 5  |036 6  |037 7   |
|038 8  |039 9  |03a :  |03b ;  |03c <  |03d =  |03e >  |03f ?   |
|040 @  |041 A  |042 B  |043 C  |044 D  |045 E  |046 F  |047 G   |
|048 H  |049 I  |04a J  |04b K  |04c L  |04d M  |04e N  |04f O   |
|050 P  |051 Q  |052 R  |053 S  |054 T  |055 U  |056 V  |057 W   |
|058 X  |059 Y  |05a Z  |05b [  |05c \  |05d ]  |05e ^  |05f _   |
|060 `  |061 a  |062 b  |063 c  |064 d  |065 e  |066 f  |067 g   |
|068 h  |069 i  |06a j  |06b k  |06c l  |06d m  |06e n  |06f o   |
|070 p  |071 q  |072 r  |073 s  |074 t  |075 u  |076 v  |077 w   |
|078 x  |079 y  |07a z  |07b {  |07c |  |07d }  |07e ~  |07f del |
```

NAME

environ - user environment

DESCRIPTION

An array of strings called the "environment" is made available by *exec* (2) when a process begins. By convention, these strings have the form "name=value". The following names are used by various commands:

PATH	The sequence of directory prefixes that *sh* (1), *time* (1), *nice* (1), *nohup* (1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (**:**). *login* (1) sets **PATH=:/bin:/usr/bin.**
HOME	Name of the user's login directory, set by *login* (1) from the password file *passwd* (4).
TERM	The kind of terminal for which output is to be prepared. This information is used by commands, such as *mm* (1) or *tplot* (1G), which may exploit special capabilities of that terminal.
TZ	Time zone information. The format is ***xxx n zzz*** where **xxx** is standard local time zone abbreviation, *n* is the difference in hours from GMT, and **zzz** is the abbreviation for the daylight-saving local time zone, if any; for example, **EST5EDT.**

Further names may be placed in the environment by the *export* command and "name=value" arguments in *sh* (1), or by *exec* (2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS.**

SEE ALSO

exec(2).

env(1), login(1), sh(1), nice(1), nohup(1), time(1) in the *D-NIX 5.3 Reference Manual.*

mm(1) in the *DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*

NAME

fcntl - file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The *fcntl* (2) function provides for control over open files. This include file describes *requests* and *arguments* to *fcntl* and *open* (2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open) */
#define O_RDONLY   0
#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 */
/* 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 file lock */
#define F_SETLK    6    /* Set file lock */
#define F_SETLKW   7    /* Set file lock and wait */
#define F_CHKFL    8    /* Check legality of file flag changes */
/* file segment locking control structure */
struct  flock {
        shortl_type;
        shortl_whence;
        long  l_start;
        long  l_len;     /* if 0 then until EOF */
        shortl_sysid;    /* returned with F_GETLK */
        shortl_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

math - math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the *matherr* (3M) error-handling mechanisms, including the following constant used as an error-return value:

HUGE	The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (*e*).
M_LOG2E	The base-2 logarithm of *e*.
M_LOG10E	The base-10 logarithm of *e*.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	π, the ratio of the circumference of a circle to its diameter.
M_PI_2	$\pi/2$.
M_PI_4	$\pi/4$.
M_1_PI	$1/\pi$.
M_2_PI	$2/\pi$.
M_2_SQRTPI	$2/\sqrt{\pi}$.
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

For the definitions of various machine-dependent "constants," see the description of the *<values.h>* header file.

SEE ALSO

intro(3), matherr(3M), values(5).

NAME

prof - profile within a function

SYNOPSIS

#define MARK

#include <prof.h>

void MARK (name)

DESCRIPTION

MARK will introduce a mark called *name* that will be treated the same as a function entry point. Execution of the mark will add to a counter for that mark, and program-counter time spent will be accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

name may be any combination of numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file < *prof.h* > is included. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, i.e:

```
cc -p -DMARK foo.c
```

If MARK is not defined, the MARK *(name)* statements may be left in the source files containing them and will be ignored.

EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with MARK defined on the command line, the marks are ignored.

```
#include <prof.h>

foo( ) {
   int i, j;
   .   .
   .
   MARK(loop1);
   for (i = 0; i < 2000; i++) {
         . . .
   }
   MARK(loop2);
   for (j = 0; j < 2000; j++) {
         . . .
   }
}
```

SEE ALSO

prof(1), profil(2), monitor(3C).

NAME

rcsfile - format of RCS file

DESCRIPTION

An RCS file is an ASCII file. Its contents is described by the grammar below. The text is free format, i.e., spaces, tabs and new lines have no significance except in strings. Strings are enclosed by '@'. If a string contains a '@', it must be doubled.

The meta syntax uses the following conventions: '|' (bar) separates alternatives; '{' and '}' enclose optinal phrases; '{' and '}*' enclose phrases that may be repeated zero or more times; '{' and '}+' enclose phrases that must appear at least once and may be repeated; '<' and '>' enclose nonterminals.

```
<rcstext>      ::=   <admin> {<delta>}* <desc> {<deltatext>}*
<admin>        ::=   head {<num>};
                     access {<id>}*;
                     symbols {<id> : <num>}*;
                     locks {<id> : <num>}*;
                     comment {<string>};
<delta>        ::=   <num>  date <num>;
                     author <id>;
                     state {<id>};
                     branches {<num>}*;
                     next {<num>};
<desc>         ::=   desc <string>
<deltatext>          ::=    <num>  log <string>  text <string>
<num>          ::=   {<digit>{.}}+
<digit>        ::=   0 | 1 | ... | 9
<id>           ::=   <letter>{<idchar>}*
<letter>       ::=   A | B | ... | Z | a | b | ... | z
<idchar>       ::=   Any printing ASCII character except space, tab,
                     carriage return, new line, and <special>.
<special>      ::=   ; | : | , | @
<string>       ::=   @{any ASCII character, with '@' doubled}*@
```

Identifiers are case sensitive. Keywords are in lower case only. The sets of keywords and identifiers may overlap.

The <delta> nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the "trunk", and are linked through the "next" field in order of decreasing numbers. The "head" field in the <admin> node points to the head of that sequence (i.e., contains the highest pair).

All <delta> nodes whose numbers consist of 2n fields (n_2) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first (2n)-1 number fields are identical are linked through the "next" field in order of increasing numbers. For each such sequence, the <delta> node whose number is identical to the first 2(n-1) number fields of the deltas on that sequence is called the branchpoint. The "branches" field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

SEE ALSO

ci(1B), co(1B), ident(1B), rcs(1B), rcsdiff(1B), rcsintro(1B), rcsmerge(1B), rlog(1B), sccstorcs(8B).

NAME

regexp - regular expression compile and match routines

SYNOPSIS

#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>
#include <regexp.h>

char *compile (instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;
int step (string, expbuf)"
char *string, *expbuf;
extern char *loc1, *loc2, *locs;
extern int circf, sed, nbra;

DESCRIPTION

This page describes general-purpose regular expression matching routines in the form of *ed* (1), defined in **<regexp.h>.** Programs such as *ed* (1), *sed* (1), *grep* (1), *bs* (1), *expr* (1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the "#include <regexp.h>" statement. These macros are used by the *compile* routine.

GETC()	Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.
PEEKC()	Return the next character in the regular expression. Successive calls to PEEKC() should return the same character [which should also be the next character returned by GETC()].
UNGETC(c)	Cause the argument *c* to be returned by the next call to GETC() [and PEEKC()]. No more that one character of pushback is ever needed and this character is guaranteed to be the last character read by GETC(). The value of the macro UNGETC(c) is always ignored.
RETURN(pointer)	This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.

ERROR(val) This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

ERROR	**MEANING**
11	Range endpoint too large.
16	Bad number.
25	"\digit" out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{\}.
45	} expected after \.
46	First number exceeds second in \{\}.
49	[ ] imbalance.
50	Regular expression overflow.

The syntax of the *compile* routine is as follows:

```
compile(instring, expbuf, endbuf, eof)
```

The first parameter *instring* is never used explicitly by the *compile* routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the INIT declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of ((char *) 0) for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf* - *expbuf*) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. For example, in *ed* (1), this character is usually a /.

Each program that includes this file must have a **#define** statement for INIT. This definition will be placed right after the declaration for the function *compile* and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point the beginning of the regular expression so that this register variable can be used in the declarations for GETC(), PEEKC() and UNGETC(). Otherwise it can be used to declare external variables that might be used by GETC(), PEEKC() and UNGETC(). See the example below of the declarations taken from *grep* (1).

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

```
step(string, expbuf)
```

The first parameter to *step* is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter *expbuf* is the compiled regular expression which was obtained by a call of the function *compile.*

The function *step* returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step.* The variable set in *step* is *loc1.* This is a pointer to the first character that matched the regular expression. The variable *loc2,* which is set by the function *advance,* points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* will point to the first character of *string* and *loc2* will point to the null at the end of *string.*

step uses the external variable *circf* which is set by *compile* if the regular expression begins with ^. If this is set then *step* will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed the value of *circf* should be saved for each compiled expression and *circf* should be set to that saved value before each call to *step.*

The function *advance* is called from *step* with the same arguments as *step.* The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called; simply call *advance.*

When *advance* encounters a * or \{\} sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, *advance* will back up along the string until it finds a match or reaches the point in the string that initially matched the * or \{\}. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer *locs* is equal to the point in the string at sometime during the backing up process, *advance* will break out of the loop that backs up and will return zero. This is used by *ed* (1) and *sed* (1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like **s/y*//g** do not loop forever.

The additional external variables *sed* and *nbra* are used for special purposes.

EXAMPLES

The following is an example of how the regular expression macros and calls look from *grep* (1):

```
#define INIT                    register char *sp = instring;
#define GETC()                  (*sp++)
#define PEEKC()                 (*sp)
#define UNGETC(c)               (--sp)
#define RETURN(c)               return;
```

```
#define ERROR(c)                    regerr()

#include <regexp.h>
. . .
        (void) compile(*argv, expbuf, &expbuf[ESIZE], \0;
. . .
        if (step(linebuf, expbuf))
                                    succeed();
```

SEE ALSO

ed(1), expr(1), grep(1), sed(1) in the *D-NIX 5.3 Reference Manual.*

NAME

stat - data returned by stat system call

SYNOPSIS

#include <sys/types.h>

#include <sys/stat.h>

DESCRIPTION

The system calls *stat* and *fstat* return data whose structure is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

Structure of the result of stat

```
struct  stat {
dev_t        st_dev;
   ushort    st_ino;
   ushort    st_mode;
   short     st_nlink;
   ushort    st_uid;
   ushort    st_gid;
   dev_t     st_rdev;
   off_t     st_size;
   time_t    st_atime;
   time_t    st_mtime;
   time_t    st_ctime;
};
#define S_IFMT     0170000    /* type of file */
#define S_IFDIR    0040000    /* directory */
#define S_IFCHR    0020000    /* character special */
#define S_IFBLK    0060000    /* block special */
#define S_IFREG    0100000    /* regular */
#define S_IFIFO    0010000    /* fifo */
#define S_ISUID    04000      /* set user id on execution */
#define S_ISGID    02000      /* set group id on execution */
#define S_ISVTX    01000      /* save swapped text even after use */
#define S_IREAD    00400      /* read permission, owner*/
#define S_IWRITE   00200      /* write permission, owner */
#define S_IEXEC    00100      /* execute/search permission, owner*/
#define S_ENFMT    S_ISGID    /* record locking enforcement flag */
#define S_IRWXU    00700      /* read,write, execute: owner */
#define S_IRUSR    00400      /* read permission: owner */
#define S_IWUSR    00200      /* write permission: owner */
#define S_IXUSR    00100      /* execute permission: owner */
#define S_IRWXG    00070      /* read, write, execute: group */
#define S_IRGRP    00040      /* read permission: group */
#define S_IWGRP    00020      /* write permission: group */
#define S_IXGRP    00010      /* execute permission: group */
#define S_IRWXO    00007      /* read, write, execute: other */
#define S_IROTH    00004      /* read permission: other */
#define S_IWOTH    00002      /* write permission: other */
#define S_IXOTH    00001      /* execute permission: other */
```

SEE ALSO

stat(2), types(5).

NAME

term - conventional names for terminals

DESCRIPTION

These names are used by certain commands (e.g., *man* (1), *tabs* (1), *tput* (1), *vi* (1) and *curses* (3X)) and are maintained as part of the shell environment in the environment variable TERM (see *sh* (1), *profile (4),* and *environ* (5)).

Entries in *terminfo* (4) source files consist of a number of comma-separated fields. (To obtain the source description for a terminal, use the **-I** option of *infocmp* (1M).) White space after each comma is ignored. The first line of each terminal description in the *terminfo* (4) database gives the names by which *terminfo* (4) knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to use to set the environment variable TERMINFO in **$HOME/.profile**; see *profile* (4)), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from [a-z0-9], make up a basic terminal name. Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, an AT&T 4425 terminal in 132 column mode would be **att4425-w**. The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	att4425-w
-am	With auto. margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-n	Number of lines on the screen	aaa-60
-na	No arrow keys (leave them in local)	c100-na
-np	Number of pages of memory	c100-4p
-rv	Reverse video	att4415-rv

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (e.g., **-w**), it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the *terminfo* (4) database unique. Terminal entries that are present only for inclusion in other entries via the **use=** facilities should have a '+' in their name, as in **4415+nl.**

Some of the known terminal names may include the following (for a complete list, type: **ls -C /usr/lib/terminfo/?**):

adm3a	Lear Siegler ADM 3A
ansi	Generic ansi standard terminal
vt100	DEC VT100
vt100-sl	Slow DEC VT100
vt100-w	DEC VT100 with 132 columns (advanced video)
vt52	DEC VT52
twist	Facit Twist landscape mode
twi72	Facit Twist portrait mode
vt220	DEC VT220
vt220-w	DEC VT220 with 132 columns
vt220-s	DEC VT220 with status line
com	Comex 8000 with 24 lines
comw	Comex 8000 with 34 lines
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences

Commands whose behavior depends on the type of terminal should accept arguments of the form **-T** ***term*** where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **TERM,** which, in turn, should contain *term*.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

SEE ALSO

curses(3X), profile(4), terminfo(4), environ(5).

sh(1), stty(1) in the *D-NIX 5.3 Reference Manual.*

man(1), tabs(1), tput(1), vi(1) in the *User's Reference Manual.*

infocmp(1M) in the *Administrator's Reference Manual.*

NOTES

Not all programs follow the above naming conventions.

NAME

values - machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(***type*** **)**	The number of bits in a specified type (e.g., int).
HIBITS	The value of a short integer with only the high-order bit set (in most implementations, 0x8000).
HIBITL	The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).
HIBITI	The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).
MAXSHORT	The maximum value of a signed short integer (in most implementations, 0x7FFF = 32767).
MAXLONG	The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF = 2147483647).
MAXINT	The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).
MAXFLOAT, LN_MAXFLOAT	The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE, LN_MAXDOUBLE	The maximum value of a double-precision floating-point number, and its natural logarithm.
MINFLOAT, LN_MINFLOAT	The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE, LN_MINDOUBLE	The minimum positive value of a double-precision floating-point number, and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.

SEE ALSO

intro(3), math(5).

NAME

varargs - handle variable argument list

SYNOPSIS

#include <varargs.h>
va_alist
va_dcl
void va_start(pvar)
va_list pvar;
***type* va_arg(pvar, *type*)**
va_list pvar;
void va_end(pvar)
va_list pvar;

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists [such as *printf* (3S)] but do not use *varargs* are inherently nonportable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for *va_alist.* No semicolon should follow *va_dcl.*

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize *pvar* to the beginning of the list.

va_arg will return the next argument in the list pointed to by *pvar. type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by *va_start* ... *va_end,* are possible.

EXAMPLE

This example is a possible implementation of *execl* (2).

```
#include <varargs.h>
#define MAXARGS   100

/* execl is called by
            execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
   va_list ap;
   char *file;
   char *args[MAXARGS];
   int argno = 0;
   va_start(ap);
   file = va_arg(ap, char *);
   while ((args[argno++] = va_arg(ap, char *)) != (char *)0)
         ;
   va_end(ap);
   return execv(file, args);
}
```

SEE ALSO

exec(2), printf(3S), vprintf(3S).

NOTES

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this from the stack frame. For example, *execl* is passed a zero pointer to signal the end of the list. *printf* can tell how many arguments are there by the format.

It is non-portable to specify a second argument of *char*, *short*, or *float* to *va_arg*, since arguments seen by the called function are not *char*, *short*, or *float*. C converts *char* and *short* arguments to *int* and converts *float* arguments to *double* before passing them to a function.

NAME

intro - introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

NAME

ascii - map of ASCII character set

DESCRIPTION

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

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel |
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si  |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb |
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us  |
|040 sp |041 ! |042 " |043 # |044 $ |045 % |046 & |047 '  |
|050 (  |051 )  |052 *  |053 +  |054 ,  |055 -  |056 .  |057 /   |
|060 0  |061 1  |062 2  |063 3  |064 4  |065 5  |066 6  |067 7   |
|070 8  |071 9  |072 :  |073 ;  |074 <  |075 =  |076 >  |077 ?   |
|100 @  |101 A  |102 B  |103 C  |104 D  |105 E  |106 F  |107 G   |
|110 H  |111 I  |112 J  |113 K  |114 L  |115 M  |116 N  |117 O   |
|120 P  |121 Q  |122 R  |123 S  |124 T  |125 U  |126 V  |127 W   |
|130 X  |131 Y  |132 Z  |133 [  |134 \  |135 ]  |136 ^  |137 _   |
|140 '  |141 a  |142 b  |143 c  |144 d  |145 e  |146 f  |147 g   |
|150 h  |151 i  |152 j  |153 k  |154 l  |155 m  |156 n  |157 o   |
|160 p  |161 q  |162 r  |163 s  |164 t  |165 u  |166 v  |167 w   |
|170 x  |171 y  |172 z  |173 {  |174 |  |175 }  |176 ~  |177 del |


|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel |
|008 bs |009 ht |00a nl |00b vt |00c np |00d cr |00e so |00f si  |
|010 dle|011 dc1|012 dc2|013 dc3|014 dc4|015 nak|016 syn|017 etb |
|018 can|019 em |01a sub|01b esc|01c fs |01d gs |01e rs |01f us  |
|020 sp |021 !  |022 "  |023 #  |024 $  |025 %  |026 &  |027 '   |
|028 (  |029 )  |02a *  |02b +  |02c ,  |02d -  |02e .  |02f /   |
|030 0  |031 1  |032 2  |033 3  |034 4  |035 5  |036 6  |037 7   |
|038 8  |039 9  |03a :  |03b ;  |03c <  |03d =  |03e >  |03f ?   |
|040 @  |041 A  |042 B  |043 C  |044 D  |045 E  |046 F  |047 G   |
|048 H  |049 I  |04a J  |04b K  |04c L  |04d M  |04e N  |04f O   |
|050 P  |051 Q  |052 R  |053 S  |054 T  |055 U  |056 V  |057 W   |
|058 X  |059 Y  |05a Z  |05b [  |05c \  |05d ]  |05e ^  |05f _   |
|060 '  |061 a  |062 b  |063 c  |064 d  |065 e  |066 f  |067 g   |
|068 h  |069 i  |06a j  |06b k  |06c l  |06d m  |06e n  |06f o   |
|070 p  |071 q  |072 r  |073 s  |074 t  |075 u  |076 v  |077 w   |
|078 x  |079 y  |07a z  |07b {  |07c |  |07d }  |07e ~  |07f del |
```

4

NAME

intro - introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals, and UNIX system device drivers. STREAMS [see *intro* (2)] software drivers, modules and the STREAMS-generic set of *ioctl* (2) system calls are also described.

For hardware related files, 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 UNIX system device driver are discussed where applicable.

NAME

clone - open any minor device on a STREAMS driver

DESCRIPTION

clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to *clone* during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate *stream* to a previously unused minor device.

The *clone* driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls (including *close* (2)) require no further involvement of *clone*.

clone will generate an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

CAVEATS

Multiple opens of the same minor device cannot be done through the *clone* interface. Executing *stat* (2) on the file system node for a cloned device yields a different result from executing *fstat* (2) using a file descriptor obtained from opening the node.

NAME

console - console interface

DESCRIPTION

The console provides the operator interface to the DS90 computer.

The file **/dev/console** is the system console, and refers to an asynchronous serial data line originating from the system board. This special file implements the features described in *termio* (7).

The file **/dev/contty** refers to a second asynchronous serial data line originating from the system board. This special file implements the features described in *termio* (7).

FILES

/dev/console
/dev/contty

SEE ALSO

termio(7).

NAME

mem, kmem - core memory

DESCRIPTION

The file /**dev/mem** is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in /**dev/mem** are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file /**dev/kmem** is the same as /**dev/mem** except that kernel virtual memory rather than physical memory is accessed.

FILES

/dev/mem
/dev/kmem

WARNING

Some of /**dev/kmem** cannot be read because of write-only addresses or unequipped memory addresses.

NAME

null - the null file

DESCRIPTION

Data written on the null special file, **/dev/null**, is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

streamio - STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;

DESCRIPTION

STREAMS [see *intro* (2)] ioctl commands are a subset of *ioctl* (2) system calls which perform a variety of control functions on *streams*. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream head*. Certain combinations of these arguments may be passed to a module or driver in the *stream*.

fildes is an open file descriptor that refers to a *stream*. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the *stream* referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. This causes subsequent system calls to fail with *errno* set to this value.

COMMAND FUNCTIONS

The following *ioctl* commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH	Pushes the module whose name is pointed to by *arg* onto the top of the current *stream*, just below the *stream head*. It then calls the open routine of the newly-pushed module. On failure, *errno* is set to one of the following values:
[EINVAL]	Invalid module name.
[EFAULT]	*arg* points outside the allocated address space.
[ENXIO]	Open routine of new module failed.
[ENXIO]	Hangup received on *fildes*.
I_POP	Removes the module just below the *stream head* of the *stream* pointed to by *fildes*. *arg* should be 0 in an I_POP request. On failure, *errno* is set to one of the following values:

[EINVAL]	No module present in the *stream*.
[ENXIO]	Hangup received on *fildes*.
I_LOOK	Retrieves the name of the module just below the *stream head* of the *stream* pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long. An **#include <sys/conf.h>** declaration is required. On failure, *errno* is set to one of the following values:
[EFAULT]	*arg* points outside the allocated address space.
[EINVAL]	No module present in *stream*.
I_FLUSH	This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:
FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.
	On failure, *errno* is set to one of the following values:
[EAGAIN]	Unable to allocate buffers for flush message.
[EINVAL]	Invalid *arg* value.
[ENXIO]	Hangup received on *fildes*.
I_SETSIG	Informs the *stream head* that the user wishes the kernel to issue the SIGPOLL signal [see *signal* (2) and *sigset* (2)] when a particular event has occurred on the *stream* associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:
S_INPUT	A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.
S_HIPRI	A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.
S_OUTPUT	The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the *stream head* read queue.
	A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same Stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, *errno* is set to one of the following values:

[EINVAL]	*arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.
[EAGAIN]	Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, *errno* is set to one of the following values:

[EINVAL]	Process not registered to receive the SIGPOLL signal.
[EFAULT]	*arg* points outside the allocated address space.

I_FIND This request compares the names of all modules currently present in the *stream* to the name pointed to by *arg*, and returns 1 if the named module is present in the *stream*. It returns 0 if the named module is not present. On failure, *errno* is set to one of the following values:

[EFAULT]	*arg* points outside the allocated address space.
[EINVAL]	*arg* does not contain a valid module name.

I_PEEK This request allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures [see *getmsg* (2)] must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to RS_HIPRI, I_PEEK will only look for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the RS_HIPRI flag was set in *flags* and a

	priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or RS_HIPRI. On failure, *errno* is set to the following value:
[EFAULT]	*arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.
I_SRDOPT	Sets the read mode using the value of the argument *arg*. Legal *arg* values are:
RNORM	Byte-stream mode, the default.
RMSGD	Message-discard mode.
RMSGN	Message-nondiscard mode.
	Read modes are described in *read* (2). On failure, *errno* is set to the following value:
[EINVAL]	*arg* is not one of the above legal values.
I_GRDOPT	Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in *read*(2). On failure, *errno* is set to the following value:
[EFAULT]	*arg* points outside the allocated address space.
I_NREAD	Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the *ioctl* return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, *errno* is set to the following value:
[EFAULT]	*arg* points outside the allocated address space.
I_FDINSERT	creates a message from user specified buffer(s), adds information about another *stream* and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. *arg* points to a *strfdinsert* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
int             fd;
int             offset;
```

The *len* field in the *ctlbuf strbuf* structure [see *putmsg* (2)] must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fd* specifies the file descriptor of the other *stream* and *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where I_FDINSERT will store a pointer to the *fd stream*'s driver read queue structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT\ also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether O_NDELAY has been specified. No partial message is sent. On failure, *errno* is set to one of the following values:

[EAGAIN]	A non-priority message was specified, the O_NDELAY flag is set, and the *stream* write queue is full due to internal flow control conditions.
[EAGAIN]	Buffers could not be allocated for the message that was to be created.
[EFAULT]	*arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.
[EINVAL]	One of the following: *fd* in the *strfdinsert* structure is not a valid, open *stream* file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*; *offset* does not specify a properly-aligned location in the data buffer; an undefined value is stored in *flags*.
[ENXIO]	Hangup received on *fildes*.
[ERANGE]	The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_STR

Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send user *ioctl* requests to downstream modules and drivers. It allows information to be sent with the ioctl, and will return to the user any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one I_STR can be active on a *stream*. Further I_STR calls will block until the active I_STR completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O_NDELAY [see *open* (2)] flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int   ic_cmd;          /* downstream command */
int   ic_timout;       /* ACK/NAK timeout */
int   ic_len;          /* length of data arg */
char  *ic_dp;          /* ptr to data arg */
```

ic_cmd is the internal ioctl command intended for a downstream module or driver and *ic_timout* is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I_STR request will wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the *stream* can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream. On failure, *errno* is set to one of the following values:

[EAGAIN] Unable to allocate buffers for the *ioctl* message.

[EFAULT] *arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space.

[EINVAL] *ic_len* is less than 0 or *ic_len* is larger than the maximum configured size of the data part of a message or *ic_timout* is less than -1.

[ENXIO] Hangup received on *fildes*.

[ETIME] A downstream *ioctl* timed out before acknowledgement was received.

An I_STR can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *stream head*. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the ioctl command sent downstream fails. For these cases, I_STR will fail with *errno* set to the value in the message.

I_SENDFD Requests the *stream* associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a *stream* pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor. I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue [see *intro* (2)] of the *stream head* at the other end of the *stream* pipe to which it is connected. On failure, *errno* is set to one of the following values:

[EAGAIN] The sending *stream* is unable to allocate a message block to contain the file pointer.

[EAGAIN] The read queue of the receiving *stream head* is full and cannot accept the message sent by I_SENDFD.

[EBADF] *arg* is not a valid, open file descriptor.

[EINVAL] *fildes* is not connected to a *stream* pipe.

[ENXIO] Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an I_SENDFD *ioctl* over a *stream* pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

fd is an integer file descriptor. *uid* and *gid* are the user id and group id, respectively, of the sending *stream*.

If O_NDELAY is not set [see *open* (2)], I_RECVFD will block until a message is present at the *stream head*. If

O_NDELAY is set, I_RECVFD will fail with *errno* set to EAGAIN if no message is present at the *stream head*.

If the message at the *stream head* is a message sent by an I_SENDFD, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, *errno* is set to one of the following values:

[EAGAIN]	A message was not present at the *stream head* read queue, and the O_NDELAY flag is set.
[EBADMSG]	The message at the *stream head* read queue was not a message containing a passed file descriptor.
[EFAULT]	*arg* points outside the allocated address space.
[EMFILE]	NOFILES file descriptors are currently open.
[ENXIO]	Hangup received on *fildes*.

The following two commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK — Connects two *streams*, where *fildes* is the file descriptor of the *stream* connected to the multiplexing driver, and *arg* is the file descriptor of the *stream* connected to another driver. The *stream* designated by *arg* gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see I_UNLINK) on success, and a -1 on failure. On failure, *errno* is set to one of the following values:

[ENXIO]	Hangup received on *fildes*.
[ETIME]	Time out before acknowledgement message was received at *stream head*.
[EAGAIN]	Unable to allocate STREAMS storage to perform the I_LINK.
[EBADF]	*arg* is not a valid, open file descriptor.
[EINVAL]	*fildes stream* does not support multiplexing.
[EINVAL]	*arg* is not a *stream*, or is already linked under a multiplexor.
[EINVAL]	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a mes-

	sage indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_LINK will fail with *errno* set to the value in the message.
I_UNLINK	Disconnects the two *streams* specified by *fildes* and *arg*. *fildes* is the file descriptor of the *stream* connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the *ioctl* I_LINK command when a *stream* was linked below the multiplexing driver. If *arg* is -1, then all Streams which were linked to *fildes* are disconnected. As in I_LINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, *errno* is set to one of the following values:
[ENXIO]	Hangup received on *fildes*.
[ETIME]	Time out before acknowledgement message was received at *stream head*.
[EAGAIN]	Unable to allocate buffers for the acknowledgement message.
[EINVAL]	Invalid multiplexor ID number.
	An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK will fail with *errno* set to the value in the message.

SEE ALSO

close(2), fcntl(2), intro(2), ioctl(2), open(2), read(2), getmsg(2), poll(2), putmsg(2), signal(2), sigset(2), write(2) in the *Programmer's Reference Manual*.

DIAGNOSTICS

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

termio - general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork* (2). A process can break this association by changing its process group using *setpgrp* (2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, the buffer is flushed and all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\ ). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR	(Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to

	terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal* (2).
QUIT	(Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.
SWTCH	(Control-z or ASCII SUB) is used by the job control facility, *shl,* to change the current layer to the control layer.
ERASE	(#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.
KILL	(@) deletes the entire line, as delimited by a NL, EOF, or EOL character.
EOF	(Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.
NL	(ASCII LF) is the normal line delimiter. It can not be changed or escaped.
EOL	(ASCII NUL) is an additional line delimiter, like NL. It is not normally used.
EOL2	is another additional line delimiter.
STOP	(Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hang-up* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hang-up signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl* (2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>** :

```
#define NCC    8
struct  termio {
        unsigned   shortc_iflag;    /* input modes */
        unsigned   shortc_oflag;    /* output modes */
        unsigned   shortc_cflag;    /* control modes */
        unsigned   shortc_lflag;    /* local modes */
        char       c_line;          /* line discipline */
        unsigned   char c_cc[NCC];  /* control chars */
};
```

The special control characters are defined by the array *c_cc* . The relative positions and initial values for each function are as follows:

```
0  VINTR      DEL
1  VQUIT      FS
2  VERASE     #
3  VKILL      @
4  VEOF       EOT
5  VEOL       NUL
6  reserved
7  SWTCH
```

The *c_iflag* field describes the basic terminal input control:

```
IGNBRK   0000001    Ignore break condition.
BRKINT   0000002    Signal interrupt on break.
IGNPAR   0000004    Ignore characters with parity errors.
PARMRK   0000010    Mark parity errors.
INPCK    0000020    Enable input parity check.
ISTRIP   0000040    Strip character.
INLCR    0000100    Map NL to CR on input.
IGNCR    0000200    Ignore CR.
ICRNL    0000400    Map CR to NL on input.
IUCLC    0001000    Map upper-case to lower-case on input.
IXON     0002000    Enable start/stop output control.
IXANY    0004000    Enable any character to restart output.
IXOFF    0010000    Enable start/stop input control.
```

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if

ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all-bits-clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXTA	0000016	External A

B38400	0000017	38400 baud
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
RCV1EN	0010000	
XMT1EN	0020000	
LOBLK	0040000	Block layer output.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT*S. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for :	*use :*
‘	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **\a,\n** as **\\n** , and **\N** as **\\\n.**

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit, switch, and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl* (2) system calls have the form:

```
   ioctl (fildes, command, arg)
   struct termio *arg;
```

The commands using this form are:

TCGETA	Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg.**

TCSETA	Set the parameters associated with the terminal from the structure referenced by **arg.** The change is immediate.
TCSETAW	Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
TCSETAF	Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl* (2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK	Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).
TCXONC	Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
TCFLSH	If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1) in the *D-NIX 5.3 Reference Manual.*

fork(2), ioctl(2), setpgrp(2), signal(2) in the *Programmer's Reference Manual.*

NAME

tty - controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

SEE ALSO

console(7), ports(7).

5

NAME

intro - introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions of such topics as boot procedures, recovery from crashes, file backups, etc.

NAME

sccstorcs - build RCS file from SCCS file

SYNOPSIS

sccstorcs [-t] [-v] s.file ...

DESCRIPTION

Sccstorcs builds an RCS file from each SCCS file argument. The deltas and comments for each delta are preserved and installed into the new RCS file in order. Also preserved are the user access list and descriptive text, if any, from the SCCS file.

OPTIONS

The following flags are meaningful:

-t	Trace only. Prints detailed information about the SCCS file and lists the commands that would be executed to produce the RCS file. No commands are actually executed and no RCS file is made.
-v	Verbose. Prints each command that is run while it is building the RCS file.

FILES

For each **s.somefile**, *Sccstorcs* writes the files **somefile** and **somefile,v** which should not already exist. *Sccstorcs* will abort, rather than overwrite those files if they do exist.

SEE ALSO

ci(1B), co(1B), rcs(1B).

Walter F. Tichy, *Design, Implementation, and Evaluation of a Revision Control System*, in Proceedings of the 6th International Conference on Software Engineering, IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

All diagnostics are written to stderr. Non-zero exit status on error.

BUGS

Sccstorcs does not preserve all SCCS options specified in the SCCS file. Most notably, it does not preserve removed deltas, MR numbers, and cutoff points.

6

7

8

9

10

11

12

Manual references for D-NIX 5.3

This is a reference section for the D-NIX 5.3 operating system. All commands and functions are listed with a reference to the appropriate manual.

A

a.out(4)	AT&T Administrator's Reference Manual
a641(3C)	AT&T Programmer's Reference Manual
abort(3C)	AT&T Programmer's Reference Manual
abs(3C)	AT&T Programmer's Reference Manual
accept(1M)	D-NIX 5.3 Reference Manual
access(2)	AT&T Programmer's Reference Manual
acct(1M)	AT&T Administrator's Reference Manual
acct(2)	AT&T Programmer's Reference Manual
acct(4)	AT&T Administrator's Reference Manual
acctcms(1M)	AT&T Administrator's Reference Manual
acctcom(1)	AT&T User's Reference Manual
acctcon(1M)	AT&T Administrator's Reference Manual
acctcon1(1M)	See acctcon(1M)
acctcon2(1M)	See acctcon(1M)
acctdisk(1M)	See acct(1M)
acctdusg(1M)	See acct(1M)
acctmerg(1M)	AT&T Administrator's Reference Manual
accton(1M)	See acct(1M)
acctprc(1M)	AT&T Administrator's Reference Manual
acctprc1(1M)	See acctprc(1M)
acctprc2(1M)	See acctprc(1M)
acctsh(1M)	AT&T Administrator's Reference Manual
acctwtmp(1M)	See acct(1M)
acos(3M)	See trig(3M)
adb(1)	AT&T User's Reference Manual
admin(1)	AT&T Programmer's Reference Manual
alarm(2)	AT&T Programmer's Reference Manual
ar(1)	AT&T User's Reference Manual
ar(4)	AT&T Administrator's Reference Manual
arithmetic(6)	AT&T User's Reference Manual
as(1)	AT&T User's Reference Manual
ascii(5)	AT&T Administrator's Reference Manual
asctime(3C)	See ctime(3C)
asin(3M)	See trig(3M)
assert(3X)	AT&T Programmer's Reference Manual
at(1)	AT&T User's Reference Manual
atan(3M)	See trig(3M)
atan2(3M)	See trig(3M)
atof(3C)	See strtod(3C)
atoi(3C)	See strtol(3C)
atol(3C)	See strtol(3C)
awk(1)	AT&T User's Reference Manual

B

back(6)	AT&T User's Reference Manual
badblk(1M)	D-NIX 5.3 Reference Manual
banner(1)	D-NIX 5.3 Reference Manual
basename(1)	D-NIX 5.3 Reference Manual
batch(1)	See at(1)
bc(1)	AT&T User's Reference Manual
bcheckrc(1M)	D-NIX 5.3 Reference Manual
bdiff(1)	AT&T User's Reference Manual
bessel(3M)	AT&T Programmer's Reference Manual
bfs(1)	AT&T User's Reference Manual
bj(6)	AT&T User's Reference Manual
bootpar(1M)	D-NIX 5.3 Reference Manual
brc(1M)	D-NIX 5.3 Reference Manual
brk(2)	AT&T Programmer's Reference Manual
bsearch(3C)	AT&T Programmer's Reference Manual
bup(1)	D-NIX 5.3 Reference Manual

C

cal(1)	AT&T User's Reference Manual
calendar(1)	AT&T User's Reference Manual
calloc(3C)	See malloc(3C)
calloc(3X)	See malloc(3X)
cancel(1)	D-NIX 5.3 Reference Manual
captinfo(1M)	AT&T Administrator's Reference Manual
cat(1)	D-NIX 5.3 Reference Manual
cb(1)	AT&T Programmer's Reference Manual
cc(1)	AT&T Programmer's Reference Manual
cd(1)	D-NIX 5.3 Reference Manual
cdc(1)	AT&T Programmer's Reference Manual
ceil(3M)	See floor(3M)
cflow(1)	AT&T Programmer's Reference Manual
chargefee(1M)	See acctsh(1M)
chdir(2)	AT&T Programmer's Reference Manual
checkeq(1)	See eqn(1)
checklist(4)	AT&T Administrator's Reference Manual
chgrp(1)	D-NIX 5.3 Reference Manual
chmod(1)	D-NIX 5.3 Reference Manual
chmod(2)	AT&T Programmer's Reference Manual
chown(1)	D-NIX 5.3 Reference Manual
chown(2)	AT&T Programmer's Reference Manual
chroot(1M)	AT&T Administrator's Reference Manual
chroot(2)	AT&T Programmer's Reference Manual
ci(1B)	AT&T User's Reference Manual
ckpacct(1M)	See acctsh(1M)
clearerr(3S)	See ferror(3S)
clock(3C)	AT&T Programmer's Reference Manual
clone(7)	AT&T Administrator's Reference Manual
close(2)	AT&T Programmer's Reference Manual
closedir(3X)	See directory(3X)
cmp(1)	D-NIX 5.3 Reference Manual

co(1B)	AT&T User's Reference Manual
col(1)	AT&T User's Reference Manual
comb(1)	AT&T Programmer's Reference Manual
comm(1)	AT&T User's Reference Manual
console(7)	AT&T Administrator's Reference Manual
conv(3C)	AT&T Programmer's Reference Manual
copy(1)	D-NIX 5.3 Reference Manual
core(4)	AT&T Administrator's Reference Manual
cos(3M)	See trig(3M)
cosh(3M)	See sinh(3M)
cp(1)	D-NIX 5.3 Reference Manual
cpio(1)	D-NIX 5.3 Reference Manual
cpio(4)	AT&T Administrator's Reference Manual
cpp(1)	AT&T Programmer's Reference Manual
cpset(1M)	AT&T Administrator's Reference Manual
craps(6)	AT&T User's Reference Manual
creat(2)	AT&T Programmer's Reference Manual
cron(1M)	D-NIX 5.3 Reference Manual
crontab(1)	D-NIX 5.3 Reference Manual
crypt(1)	AT&T User's Reference Manual
crypt(3C)	AT&T Programmer's Reference Manual
crypt(3X)	AT&T Programmer's Reference Manual
csh(1)	AT&T User's Reference Manual
csplit(1)	AT&T User's Reference Manual
ctags(1)	AT&T User's Reference Manual
ctermid(3S)	AT&T Programmer's Reference Manual
ctime(3C)	AT&T Programmer's Reference Manual
ctrace(1)	AT&T Programmer's Reference Manual
ctype(3C)	AT&T Programmer's Reference Manual
cu(1C)	D-NIX 5.3 Reference Manual
cubic(6)	AT&T User's Reference Manual
curses(3X)	AT&T Programmer's Reference Manual
cuserid(3S)	AT&T Programmer's Reference Manual
cut(1)	D-NIX 5.3 Reference Manual
cxref(1)	AT&T Programmer's Reference Manual

D

date(1)	D-NIX 5.3 Reference Manual
dc(1)	AT&T User's Reference Manual
dd(1M)	D-NIX 5.3 Reference Manual
delta(1)	AT&T Programmer's Reference Manual
deroff(1)	AT&T User's Reference Manual
devnm(1M)	D-NIX 5.3 Reference Manual
df(1M)	D-NIX 5.3 Reference Manual
dial(3C)	AT&T Programmer's Reference Manual
diff(1)	AT&T User's Reference Manual
diff(1B)	AT&T User's Reference Manual
diff3(1)	AT&T User's Reference Manual
diff3(1B)	AT&T User's Reference Manual
diffmk(1)	AT&T User's Reference Manual
dir(4)	AT&T Administrator's Reference Manual
dircmp(1)	AT&T User's Reference Manual
directory(3X)	AT&T Programmer's Reference Manual

dirent(4)	AT&T Administrator's Reference Manual
dirname(1)	D-NIX 5.3 Reference Manual
disable(1)	D-NIX 5.3 Reference Manual
dmacs(1)	D-NIX 5.3 Reference Manual
dodisk(1M)	See acctsh(1M)
drand48(3C)	AT&T Programmer's Reference Manual
ds90-00(1)	See machid(1)
ds90-10(1)	See machid(1)
ds90-11(1)	See machid(1)
ds90-20(1)	See machid(1)
ds90-21(1)	See machid(1)
ds90-30(1)	See machid(1)
ds90-31(1)	See machid(1)
du(1M)	D-NIX 5.3 Reference Manual
dump(1)	AT&T Programmer's Reference Manual
dup(2)	AT&T Programmer's Reference Manual
dup2(3C)	AT&T Programmer's Reference Manual

E

echo(1)	D-NIX 5.3 Reference Manual
ecvt(3C)	AT&T Programmer's Reference Manual
ed(1)	D-NIX 5.3 Reference Manual
edata(3C)	See end(3C)
edit(1)	AT&T User's Reference Manual
egrep(1)	AT&T User's Reference Manual
enable(1)	D-NIX 5.3 Reference Manual
encrypt(3C)	See crypt(3C)
end(3C)	AT&T Programmer's Reference Manual
endgrent(3C)	See getgrent(3C)
endpwent(3C)	See getpwent(3C)
endutent(3C)	See getut(3C)
env(1)	D-NIX 5.3 Reference Manual
environ(5)	AT&T Administrator's Reference Manual
eqn(1)	AT&T User's Reference Manual
erand48(3C)	See drand48(3C)
erf(3M)	AT&T Programmer's Reference Manual
erfc(3M)	See erf(3M)
errdemon(1M)	D-NIX 5.3 Reference Manual
errno(3C)	See perror(3C)
etext(3C)	See end(3C)
eval(1)	D-NIX 5.3 Reference Manual
ex(1)	AT&T User's Reference Manual
exec(1)	D-NIX 5.3 Reference Manual
exec(2)	AT&T Programmer's Reference Manual
exit(1)	D-NIX 5.3 Reference Manual
exit(2)	AT&T Programmer's Reference Manual
exp(3M)	AT&T Programmer's Reference Manual
export(1)	D-NIX 5.3 Reference Manual
expr(1)	D-NIX 5.3 Reference Manual

F

fabs(3M)	See floor(3M)
factor(1)	AT&T User's Reference Manual
false(1)	D-NIX 5.3 Reference Manual
fclose(3S)	AT&T Programmer's Reference Manual
fcntl(2)	AT&T Programmer's Reference Manual
fcntl(5)	AT&T Administrator's Reference Manual
fcvt(3C)	See ecvt(3C)
fdopen(3S)	See fopen(3S)
ferror(3S)	AT&T Programmer's Reference Manual
feof(3S)	See ferror(3S)
fflush(3S)	See fclose(3S)
fgetc(3S)	See getc(3S)
fgetgrent(3C)	See getgrent(3C)
fgetpwent(3C)	See getpwent(3C)
fgets(3S)	See gets(3S)
fgrep(1)	D-NIX 5.3 Reference Manual
file(1)	AT&T User's Reference Manual
filehdr(4)	AT&T Administrator's Reference Manual
fileno(3S)	See ferror(3S)
find(1)	D-NIX 5.3 Reference Manual
fish(6)	AT&T User's Reference Manual
floor(3M)	AT&T Programmer's Reference Manual
fmod(3M)	See floor(3M)
fopen(3S)	AT&T Programmer's Reference Manual
fork(2)	AT&T Programmer's Reference Manual
format(1M)	D-NIX 5.3 Reference Manual
fortune(6)	AT&T User's Reference Manual
fprintf(3S)	See printf(3S)
fputc(3S)	See putc(3S)
fputs(3S)	See puts(3S)
fread(3S)	AT&T Programmer's Reference Manual
free(3C)	See malloc(3C)
free(3X)	See malloc(3X)
freopen(3S)	See fopen(3S)
frexp(3C)	AT&T Programmer's Reference Manual
fscanf(3S)	See scanf(3S)
fsck(1M)	D-NIX 5.3 Reference Manual
fscl(1M)	D-NIX 5.3 Reference Manual
fseek(3S)	AT&T Programmer's Reference Manual
fsize(1)	D-NIX 5.3 Reference Manual
fspec(4)	AT&T Administrator's Reference Manual
fstab(4)	AT&T Administrator's Reference Manual
fstat(2)	See stat(2)
fstatfs(2)	See statfs(2)
ftell(3S)	See fseek(3S)
ftok(3C)	See stdipc(3C)
ftw(3C)	AT&T Programmer's Reference Manual
fuser(1M)	AT&T Administrator's Reference Manual
fwrite(3S)	See fread(3S)
fwtmp(1M)	AT&T Administrator's Reference Manual

G

gamma(3M)	AT&T Programmer's Reference Manual
gcvt(3C)	See ecvt(3C)
get(1)	AT&T Programmer's Reference Manual
getc(3S)	AT&T Programmer's Reference Manual
getchar(3S)	See getc(3S)
getcwd(3C)	AT&T Programmer's Reference Manual
getdents(2)	AT&T Programmer's Reference Manual
getenv(3C)	AT&T Programmer's Reference Manual
geteuid(2)	See getuid(2)
getegid(2)	See getuid(2)
getgid(2)	See getuid(2)
getgrent(3C)	AT&T Programmer's Reference Manual
getgrgid(3C)	See getgrent(3C)
getgrnam(3C)	See getgrent(3C)
getlogin(3C)	AT&T Programmer's Reference Manual
getmsg(2)	AT&T Programmer's Reference Manual
getopt(1)	D-NIX 5.3 Reference Manual
getopts(1)	D-NIX 5.3 Reference Manual
getopt(3C)	AT&T Programmer's Reference Manual
getpass(3C)	AT&T Programmer's Reference Manual
getpid(2)	AT&T Programmer's Reference Manual
getpgrp(2)	See getpid(2)
getppid(2)	See getpid(2)
getpw(3C)	AT&T Programmer's Reference Manual
getpwent(3C)	AT&T Programmer's Reference Manual
getpwnam(3C)	See getpwent(3C)
getpwuid(3C)	See getpwent(3C)
gets(3S)	AT&T Programmer's Reference Manual
getty(1M)	D-NIX 5.3 Reference Manual
gettydefs(4)	AT&T Administrator's Reference Manual
getuid(2)	AT&T Programmer's Reference Manual
getut(3C)	AT&T Programmer's Reference Manual
getutent(3C)	See getut(3C)
getutid(3C)	See getut(3C)
getutline(3C)	See getut(3C)
getw(3S)	See getc(3S)
glossary(1)	AT&T User's Reference Manual
gmtime(3C)	See ctime(3C)
grep(1)	D-NIX 5.3 Reference Manual
group(4)	AT&T Administrator's Reference Manual
grpck(1M)	See pwck(1M)
gsignal(3C)	See ssignal(3C)

H

hangman(6)	AT&T User's Reference Manual
hashcheck(1)	See spell(1)
hashmake(1)	See spell(1)
hcreate(3C)	See hsearch(3C)
hdestroy(3C)	See hsearch(3C)
help(1)	AT&T User's Reference Manual

helpadm(1M)	AT&T Administrator's Reference Manual
hsearch(3C)	AT&T Programmer's Reference Manual
hyphen(1)	AT&T User's Reference Manual
hypot(3M)	AT&T Programmer's Reference Manual

I

id(1M)	AT&T Administrator's Reference Manual
ident(1B)	AT&T User's Reference Manual
infocmp(1M)	AT&T Administrator's Reference Manual
init(1M)	D-NIX 5.3 Reference Manual
inittab(4)	AT&T Administrator's Reference Manual
inode(4)	AT&T Administrator's Reference Manual
install(1M)	AT&T Administrator's Reference Manual
ioctl(2)	AT&T Programmer's Reference Manual
ipcrm(1)	D-NIX 5.3 Reference Manual
ipcs(1)	D-NIX 5.3 Reference Manual
isalpha(3C)	See ctype(3C)
is.....(3C)	See ctype(3C)
isatty(3C)	See ttyname(3C)
issue(4)	AT&T Administrator's Reference Manual

J

j*n*(3M)	See bessel(3M)
join(1)	AT&T User's Reference Manual
jotto(6)	AT&T User's Reference Manual
jrand48(3C)	See drand48(3C)

K

kermit(1)	D-NIX 5.3 Reference Manual
kill(1)	D-NIX 5.3 Reference Manual
kill(2)	AT&T Programmer's Reference Manual
kmem(7)	See mem(7)

L

l(1)	D-NIX 5.3 Reference Manual
l3tol(3C)	AT&T Programmer's Reference Manual
l64a(3C)	See a64l(3C)
labelit(1M)	D-NIX 5.3 Reference Manual
lastlogin(1M)	See acctsh(1M)
lc(1)	D-NIX 5.3 Reference Manual
lcong48(3C)	See drand48(3C)
ld(1)	AT&T User's Reference Manual
ldaclose(3X)	See ldclose(3X)
ldahread(3X)	AT&T Programmer's Reference Manual
ldaopen(3X)	See ldopen(3X)
ldclose(3X)	AT&T Programmer's Reference Manual
ldexp(3C)	See frexp(3C)
ldfcn(4)	AT&T Administrator's Reference Manual
ldfhread(3X)	AT&T Programmer's Reference Manual
ldgetname(3X)	AT&T Programmer's Reference Manual

ldlinit(3X)	See ldlread(3X)
ldlitem(3X)	See ldlread(3X)
ldlread(3X)	AT&T Programmer's Reference Manual
ldlseek(3X)	AT&T Programmer's Reference Manual
ldnlseek(3X)	See ldlseek(3X)
ldnrseek(3X)	See ldrseek(3X)
ldnsseek(3X)	See ldsseek(3X)
ldohseek(3X)	AT&T Programmer's Reference Manual
ldopen(3X)	AT&T Programmer's Reference Manual
ldnshread(3X)	See ldshread(3X)
ldrseek(3X)	AT&T Programmer's Reference Manual
ldshread(3X)	AT&T Programmer's Reference Manual
ldsseek(3X)	AT&T Programmer's Reference Manual
ldtbindex(3X)	AT&T Programmer's Reference Manual
ldtbread(3X)	AT&T Programmer's Reference Manual
ldtbseek(3X)	AT&T Programmer's Reference Manual
lex(1)	AT&T Programmer's Reference Manual
lfind(3C)	See lsearch(3C)
limits(4)	AT&T Administrator's Reference Manual
line(1)	D-NIX 5.3 Reference Manual
linenum(4)	AT&T Administrator's Reference Manual
link(1M)	AT&T Administrator's Reference Manual
link(2)	AT&T Programmer's Reference Manual
lint(1)	AT&T Programmer's Reference Manual
ln(1)	D-NIX 5.3 Reference Manual
load(1)	D-NIX 5.3 Reference Manual
localtime(3C)	See ctime(3C)
locate(1)	AT&T User's Reference Manual
lockf(3C)	AT&T Programmer's Reference Manual
log(3M)	See exp(3M)
log10(3M)	See exp(3M)
login(1)	D-NIX 5.3 Reference Manual
logname(1)	D-NIX 5.3 Reference Manual
logname(3X)	AT&T Programmer's Reference Manual
longjmp(3C)	See setjmp(3C)
lorder(1)	AT&T Programmer's Reference Manual
lp(1)	D-NIX 5.3 Reference Manual
lpadmin(1M)	D-NIX 5.3 Reference Manual
lpd(1M)	D-NIX 5.3 Reference Manual
lpmove(1M)	D-NIX 5.3 Reference Manual
lppg(1)	D-NIX 5.3 Reference Manual
lpr(1)	D-NIX 5.3 Reference Manual
lpsched(1M)	D-NIX 5.3 Reference Manual
lpshut(1M)	D-NIX 5.3 Reference Manual
lpstat(1)	D-NIX 5.3 Reference Manual
lpsubmit(1)	D-NIX 5.3 Reference Manual
lrand48(3C)	See drand48(3C)
ls(1)	D-NIX 5.3 Reference Manual
lsearch(3C)	AT&T Programmer's Reference Manual
lseek(2)	AT&T Programmer's Reference Manual
ltol3(3C)	See l3tol(3C)

M

m4(1)	AT&T Programmer's Reference Manual
machid(1)	D-NIX 5.3 Reference Manual
mail(1)	D-NIX 5.3 Reference Manual
mailx(1)	AT&T User's Reference Manual
make(1)	AT&T Programmer's Reference Manual
makekey(1)	AT&T User's Reference Manual
mallinfo(3X)	See malloc(3X)
malloc(3C)	AT&T Programmer's Reference Manual
malloc(3X)	AT&T Programmer's Reference Manual
mallopt(3X)	See malloc(3X)
man(1)	AT&T User's Reference Manual
math(5)	AT&T Administrator's Reference Manual
matherr(3M)	AT&T Programmer's Reference Manual
maze(6)	AT&T User's Reference Manual
mc68k(1)	See machid(1)
mem(7)	AT&T Administrator's Reference Manual
memccpy(3C)	See memory(3C)
memchr(3C)	See memory(3C)
memcmp(3C)	See memory(3C)
memcpy(3C)	See memory(3C)
memory(3C)	AT&T Programmer's Reference Manual
memset(3C)	See memory(3C)
merge(1B)	AT&T User's Reference Manual
mesg(1)	D-NIX 5.3 Reference Manual
mkcfig(1M)	D-NIX 5.3 Reference Manual
mkcont(1)	D-NIX 5.3 Reference Manual
mkdir(1)	D-NIX 5.3 Reference Manual
mkdir(2)	AT&T Programmer's Reference Manual
mkfs(1M)	D-NIX 5.3 Reference Manual
mknod(1M)	D-NIX 5.3 Reference Manual
mknod(2)	AT&T Programmer's Reference Manual
mknodm(1)	D-NIX 5.3 Reference Manual
mksort(1)	D-NIX 5.3 Reference Manual
mktemp(3C)	AT&T Programmer's Reference Manual
mkuser(1M)	D-NIX 5.3 Reference Manual
mntchk(1M)	D-NIX 5.3 Reference Manual
mnttab(4)	AT&T Administrator's Reference Manual
modf(3C)	See frexp(3C)
monacct(1M)	See acctsh(1M)
monitor(3C)	AT&T Programmer's Reference Manual
moo(6)	AT&T User's Reference Manual
mount(1M)	D-NIX 5.3 Reference Manual
mount(2)	AT&T Programmer's Reference Manual
mrand48(3C)	See drand48(3C)
msgctl(2)	AT&T Programmer's Reference Manual
msgget(2)	AT&T Programmer's Reference Manual
msgop(2)	AT&T Programmer's Reference Manual
mv(1)	D-NIX 5.3 Reference Manual
mvdir(1M)	D-NIX 5.3 Reference Manual

N

neqn(1)	See eqn(1)
newform(1)	AT&T User's Reference Manual
newgrp(1M)	D-NIX 5.3 Reference Manual
news(1)	AT&T User's Reference Manual
nice(1)	D-NIX 5.3 Reference Manual
nice(2)	AT&T Programmer's Reference Manual
nl(1)	AT&T User's Reference Manual
nlist(3C)	AT&T Programmer's Reference Manual
nm(1)	AT&T Programmer's Reference Manual
nohup(1)	See nice(1)
nrand48(3C)	See drand48(3C)
nroff(1)	AT&T User's Reference Manual
ns32000(1)	See machid(1)
null(7)	AT&T Administrator's Reference Manual
nulladm(1M)	See acctsh(1M)

O

od(1)	D-NIX 5.3 Reference Manual
open(2)	AT&T Programmer's Reference Manual
opendir(3X)	See directory(3X)

P

pack(1)	AT&T User's Reference Manual
passwd(1)	D-NIX 5.3 Reference Manual
passwd(4)	AT&T Administrator's Reference Manual
paste(1)	AT&T User's Reference Manual
pause(2)	AT&T Programmer's Reference Manual
pcat(1)	See pack(1)
pclose(3S)	See popen(3S)
pdp11(1)	See machid(1)
perror(3C)	AT&T Programmer's Reference Manual
pg(1)	AT&T User's Reference Manual
pipe(2)	AT&T Programmer's Reference Manual
plock(2)	AT&T Programmer's Reference Manual
pnch(4)	AT&T Administrator's Reference Manual
poll(2)	AT&T Programmer's Reference Manual
popen(3S)	AT&T Programmer's Reference Manual
pow(3M)	See exp(3M)
powerfail(1M)	D-NIX 5.3 Reference Manual
pr(1)	D-NIX 5.3 Reference Manual
prctmp(1M)	See acctsh(1M)
prdaily(1M)	AT&T Administrator's Reference Manual
print(1)	D-NIX 5.3 Reference Manual
printf(3S)	AT&T Programmer's Reference Manual
prof(1)	AT&T Programmer's Reference Manual
prof(5)	AT&T Administrator's Reference Manual
profil(2)	AT&T Programmer's Reference Manual
profile(4)	AT&T Administrator's Reference Manual
prs(1)	AT&T Programmer's Reference Manual

prtacct(1M)	See acctsh(1M)
ps(1)	D-NIX 5.3 Reference Manual
ptrace(2)	AT&T Programmer's Reference Manual
ptx(1)	AT&T User's Reference Manual
putc(3S)	AT&T Programmer's Reference Manual
putchar(3S)	See putc(3S)
putenv(3C)	AT&T Programmer's Reference Manual
putmsg(2)	AT&T Programmer's Reference Manual
putpwent(3C)	AT&T Programmer's Reference Manual
puts(3S)	AT&T Programmer's Reference Manual
pututline(3C)	See getut(3C)
putw(3S)	See putc(3S)
pwck(1M)	AT&T Administrator's Reference Manual
pwd(1)	D-NIX 5.3 Reference Manual

Q

qsort(3C)	AT&T Programmer's Reference Manual
queue(1)	D-NIX 5.3 Reference Manual
quiz(6)	AT&T User's Reference Manual

R

rand(3C)	AT&T Programmer's Reference Manual
rcs(1B)	AT&T User's Reference Manual
rcsdiff(1B)	AT&T User's Reference Manual
rcsfile(5B)	AT&T Administrator's Reference Manual
rcsintro(1B)	AT&T User's Reference Manual
rcsmerge(1B)	AT&T User's Reference Manual
read(2)	AT&T Programmer's Reference Manual
readdir(3X)	See directory(3X)
readonly(1)	D-NIX 5.3 Reference Manual
realloc(3C)	See malloc(3C)
realloc(3X)	See malloc(3X)
red(1)	D-NIX 5.3 Reference Manual
regcmp(1)	AT&T Programmer's Reference Manual
regcmp(3X)	AT&T Programmer's Reference Manual
regex(3X)	See regcmp(3X)
regexp(5)	AT&T Administrator's Reference Manual
reject(1M)	D-NIX 5.3 Reference Manual
reloc(4)	AT&T Administrator's Reference Manual
rewind(3S)	See fseek(3S)
rewinddir(3X)	See directory(3X)
rinstall(1M)	D-NIX 5.3 Reference Manual
rlog(1B)	AT&T User's Reference Manual
rm(1)	D-NIX 5.3 Reference Manual
rmail(1)	D-NIX 5.3 Reference Manual
rmdel(1)	AT&T Programmer's Reference Manual
rmdir(1)	See rm(1)
rmdir(2)	AT&T Programmer's Reference Manual
rmuser(1M)	D-NIX 5.3 Reference Manual
rsh(1)	D-NIX 5.3 Reference Manual
runacct(1M)	AT&T Administrator's Reference Manual

S

sact(1)	AT&T Programmer's Reference Manual
sar(1)	AT&T User's Reference Manual
sar(1M)	AT&T Administrator's Reference Manual
sbrk(2)	See brk(2)
scanf(3S)	AT&T Programmer's Reference Manual
sccsdiff(1)	AT&T Programmer's Reference Manual
sccsfile(4)	AT&T Administrator's Reference Manual
sccstorcs(8B)	AT&T Administrator's Reference Manual
scnhdr(4)	AT&T Administrator's Reference Manual
scr_dump(4)	AT&T Administrator's Reference Manual
scrfile(1)	D-NIX 5.3 Reference Manual
sdb(1)	AT&T Programmer's Reference Manual
sdiff(1)	AT&T User's Reference Manual
sed(1)	D-NIX 5.3 Reference Manual
seed48(3C)	See drand48(3C)
seekdir(3X)	See directory(3X)
semctl(2)	AT&T Programmer's Reference Manual
semget(2)	AT&T Programmer's Reference Manual
semop(2)	AT&T Programmer's Reference Manual
set(1)	D-NIX 5.3 Reference Manual
setbuf(3S)	AT&T Programmer's Reference Manual
setgid(2)	See setuid(2)
setgrent(3C)	See getgrent(3C)
setjmp(3C)	AT&T Programmer's Reference Manual
setkey(3C)	See crypt(3C)
setmnt(1M)	D-NIX 5.3 Reference Manual
setpgrp(2)	AT&T Programmer's Reference Manual
setpwent(3C)	See getpwent(3C)
setspeed(1)	D-NIX 5.3 Reference Manual
setuid(2)	AT&T Programmer's Reference Manual
setutent(3C)	See getut(3C)
setvbuf(3S)	See setbuf(3S)
sh(1)	D-NIX 5.3 Reference Manual
shmctl(2)	AT&T Programmer's Reference Manual
shmget(2)	AT&T Programmer's Reference Manual
shmop(2)	AT&T Programmer's Reference Manual
shutacct(1M)	See acctsh(1M)
shutdown(1M)	D-NIX 5.3 Reference Manual
sighold(2)	See sigset(2)
sigigmore(2)	See sigset(2)
signal(2)	AT&T Programmer's Reference Manual
sigpause(2)	See sigset(2)
sigrelse(2)	See sigset(2)
sigset(2)	AT&T Programmer's Reference Manual
sin(3M)	See trig(3M)
sinh(3M)	AT&T Programmer's Reference Manual
siv(1)	D-NIX 5.3 Reference Manual
size(1)	AT&T Programmer's Reference Manual
sleep(1)	D-NIX 5.3 Reference Manual
sleep(3C)	AT&T Programmer's Reference Manual
sno(1)	AT&T User's Reference Manual

sort(1)	D-NIX 5.3 Reference Manual
spell(1)	AT&T User's Reference Manual
spellin(1)	See spell(1)
spline(1G)	AT&T User's Reference Manual
split(1)	AT&T User's Reference Manual
sprintf(3S)	See printf(3S)
sqrt(3M)	See exp(3M)
srand(3C)	See rand(3C)
srand48(3C)	See drand48(3C)
sscanf(3S)	See scanf(3S)
ssignal(3C)	AT&T Programmer's Reference Manual
starter(1)	AT&T User's Reference Manual
startup(1M)	See acctsh(1M)
stat(2)	AT&T Programmer's Reference Manual
stat(5)	AT&T Administrator's Reference Manual
statfs(2)	AT&T Programmer's Reference Manual
stdio(3S)	AT&T Programmer's Reference Manual
stdipc(3C)	AT&T Programmer's Reference Manual
stime(2)	AT&T Programmer's Reference Manual
strcat(3C)	See string(3C)
str...(3C)	See string(3C)
streamio(7)	AT&T Administrator's Reference Manual
string(3C)	AT&T Programmer's Reference Manual
strip(1)	AT&T Programmer's Reference Manual
strtod(3C)	AT&T Programmer's Reference Manual
strtol(3C)	AT&T Programmer's Reference Manual
stty(1)	D-NIX 5.3 Reference Manual
su(1M)	D-NIX 5.3 Reference Manual
sum(1)	AT&T User's Reference Manual
swab(3C)	AT&T Programmer's Reference Manual
syms(4)	AT&T Administrator's Reference Manual
sync(1M)	D-NIX 5.3 Reference Manual
sync(2)	AT&T Programmer's Reference Manual
sys_errlist(3C)	See perror(3C)
sys_nerr(3C)	See perror(3C)
sysfs(2)	AT&T Programmer's Reference Manual
system(3S)	AT&T Programmer's Reference Manual

T

tabs(1)	AT&T User's Reference Manual
tail(1)	AT&T User's Reference Manual
tan(3M)	See trig(3M)
tanh(3M)	See sinh(3M)
tar(1)	D-NIX 5.3 Reference Manual
tbl(1)	AT&T User's Reference Manual
tc(1M)	D-NIX 5.3 Reference Manual
tdelete(3C)	See tsearch(3C)
tee(1)	AT&T User's Reference Manual
telinit(1M)	D-NIX 5.3 Reference Manual
telldir(3X)	See directory(3X)
tempnam(3S)	See tmpnam(3S)
term(4)	AT&T Administrator's Reference Manual
term(5)	AT&T Administrator's Reference Manual

terminfo(4)	AT&T Administrator's Reference Manual
termio(7)	AT&T Administrator's Reference Manual
test(1)	D-NIX 5.3 Reference Manual
tfind(3C)	See tsearch(3C)
tic(1M)	AT&T Programmer's Reference Manual
time(1)	D-NIX 5.3 Reference Manual
time(2)	AT&T Programmer's Reference Manual
times(1)	D-NIX 5.3 Reference Manual
times(2)	AT&T Programmer's Reference Manual
timezone(4)	AT&T Administrator's Reference Manual
tmpfile(3S)	AT&T Programmer's Reference Manual
tmpnam(3S)	AT&T Programmer's Reference Manual
toascii(3C)	See conv(3C)
tolower(3C)	See conv(3C)
touch(1)	D-NIX 5.3 Reference Manual
toupper(3C)	See conv(3C)
tput(1)	AT&T User's Reference Manual
tr(1)	D-NIX 5.3 Reference Manual
trenter(1M)	AT&T Administrator's Reference Manual
trig(3M)	AT&T Programmer's Reference Manual
true(1)	D-NIX 5.3 Reference Manual
tsearch(3C)	AT&T Programmer's Reference Manual
tsort(1)	AT&T Programmer's Reference Manual
ttt(6)	AT&T User's Reference Manual
tty(1)	D-NIX 5.3 Reference Manual
tty(7)	AT&T Administrator's Reference Manual
ttyname(3C)	AT&T Programmer's Reference Manual
ttyslot(3C)	AT&T Programmer's Reference Manual
turnacct(1M)	See acctsh(1M)
twalk(3C)	See tsearch(3C)
type(1)	D-NIX 5.3 Reference Manual
types(5)	AT&T Administrator's Reference Manual
tzset(3C)	See ctime(3C)

U

u370(1)	See machid(1)
u3b(1)	See machid(1)
u3b2(1)	See machid(1)
u3b5(1)	See machid(1)
u3b10(1)	See machid(1)
ulimit(2)	AT&T Programmer's Reference Manual
umask(1)	D-NIX 5.3 Reference Manual
umask(2)	AT&T Programmer's Reference Manual
umount(1)	D-NIX 5.3 Reference Manual
umount(2)	AT&T Programmer's Reference Manual
uname(1)	D-NIX 5.3 Reference Manual
uname(2)	AT&T Programmer's Reference Manual
unget(1)	AT&T Programmer's Reference Manual
ungetc(3S)	AT&T Programmer's Reference Manual
uniq(1)	D-NIX 5.3 Reference Manual
unistd(4)	AT&T Administrator's Reference Manual
units(1)	AT&T User's Reference Manual
unlink(1M)	See link(1M)

unlink(2)	AT&T Programmer's Reference Manual
unpack(1)	See pack(1)
unset(1)	D-NIX 5.3 Reference Manual
usage(1)	AT&T User's Reference Manual
ustat(2)	AT&T Programmer's Reference Manual
utime(2)	AT&T Programmer's Reference Manual
utmp(4)	AT&T Administrator's Reference Manual
utmpname(3C)	See getut(3C)
uucheck(1M)	AT&T Administrator's Reference Manual
uucico(1M)	AT&T Administrator's Reference Manual
uucleanup(1M)	AT&T Administrator's Reference Manual
uucp(1C)	AT&T User's Reference Manual
uulog(1C)	See uucp(1)
uuname(1C)	See uucp(1)
uupick(1C)	See uuto(1)
uusched(1M)	AT&T Administrator's Reference Manual
uustat(1C)	AT&T User's Reference Manual
uuto(1C)	AT&T User's Reference Manual
Uutry(1M)	AT&T Administrator's Reference Manual
uux(1C)	AT&T User's Reference Manual
uuxqt(1M)	AT&T Administrator's Reference Manual

V

val(1)	AT&T Programmer's Reference Manual
values(5)	AT&T Administrator's Reference Manual
varargs(5)	AT&T Administrator's Reference Manual
vax(1)	See machid(1)
vc(1)	AT&T Programmer's Reference Manual
vi(1)	AT&T User's Reference Manual
vfprintf(3S)	See vprintf(3S)
vprintf(3S)	AT&T Programmer's Reference Manual
vprintf(3X)	AT&T Programmer's Reference Manual
vsar(1)	AT&T User's Reference Manual
vsprintf(3S)	See vprintf(3S)

W

wait(1)	D-NIX 5.3 Reference Manual
wait(2)	AT&T Programmer's Reference Manual
wall(1)	D-NIX 5.3 Reference Manual
wc(1)	D-NIX 5.3 Reference Manual
what(1)	AT&T Programmer's Reference Manual
who(1)	D-NIX 5.3 Reference Manual
whodo(1M)	AT&T Administrator's Reference Manual
write(1)	D-NIX 5.3 Reference Manual
write(2)	AT&T Programmer's Reference Manual
wtmp(4)	See utmp(4)
wtmpfix(1M)	See fwtmp(1M)
wump(6)	AT&T User's Reference Manual

X

xargs(1)	AT&T User's Reference Manual

Y

yacc(1)	AT&T Programmer's Reference Manual
y*n*(3M)	See bessel(3M)