Altos System V™ Series 386 Reference (M)

Document History

EDITION	PART NUMBER	DATE
First Edition	690-22870-001	December 1988
Second Edition	690-22870-002	June 1989

Copyright Notice

Trademarks

Limitations

GUIDE TO YOUR ALTOS SYSTEM V™ SERIES 386 DOCUMENTATION

RUN-TIME SYSTEM

Installation
Part numbers: 690-21170-*nnn*
690-21869-*nnn*

- Installation and upgrade
- Set up Multidrop and UPS

Using the AOM™ Menu System
Part number: 690-18055-*nnn*

- Easy-to-use menus to access programs
- Menu Manager to add, update, remove menus

Operations Guide
Part number: 690-21171-*nnn*

- System administration
- Accounting, file systems
- Backups, port setup
- Communications (UUCP)
- Error messages

Reference (C)
Part number: 690-22869-*nnn*

- Commands (C)

Reference (M)
Part number: 690-22870-*nnn*

- Miscellaneous files (M)

User's Guide
Part number: 690-21178-*nnn*
(Not shipped with the Run-time system)

- Basic concepts and tasks
- Vi, ed, mail, awk, sed
- Shells: sh and csh

TEXT PROCESSING SYSTEM

DOCUMENTER'S WORKBENCH™
Part numbers: 690-15843-*nnn*
690-15844-*nnn*

- Mm macros, reference
- Nroff, troff, tbl, eqn

DEVELOPMENT SYSTEM

Set part number: 690-21585-000

Reference (CP, S, F)

- Programming commands (CP)
- System calls, library routines (S)
- File formats (F)

Programmer's Guide

- Make, SCCS
- Lex, yacc
- Signals, system resources, device drivers
- Adb, sdb
- Shared libraries

C Compiler Library and User's Guide

- I/O functions, pipes
- Curses, terminfo
- Assembly routines
- As, cc, COFF, lint, ld
- Error processing
- Character and string processing

C Compiler Language Reference

- Elements of C
- Program structure
- Declarations, expressions
- Statements, functions
- Preprocessor directives

Macro Assembler User's Guide and Reference

- How to use masm
- Error messages
- Type declarations
- Operands, expressions
- Directives, file control
- Instruction summary

To order the *User's Guide* or any of the above manuals, call 408/434-6688, ext. 3004 and give the manual title and part number.

Permuted Index

The Permuted Index on the following pages contains a listing of programs, utilities, files, etc. in the Altos System V Run-time and Development Systems. These programs are described in the *Altos System V Reference.* Volume 1 of the Reference contains the Run-time system commands (C) and miscellaneous (M) sections. Volume 2 contains the Development system programming commands (CP), system calls and library routines (S), and file formats (F). Entries in each section are in alphabetical order.

NOTE

> These programs, utilities, files, etc. are subject to change.

The table that follows contains a description of each section and its location.

Description	Section	Manual
Run-time commands	C	Reference (C)
Miscellaneous -- programs and system files used for system maintenance and to access devices	M	Reference (M)
Programming commands	CP	Reference (CP, S, F
System calls and library routines for C and assembly language programming	S	Reference (CP, S, F
File formats -- programs and system files not defined in the M section	F	Reference (CP, S, F

About This Manual

USING THIS MANUAL

This reference alphabetically describes the commands and programs that are on the Altos System V™ Run-time System. Altos System V is based on UNIX® System V Release 3 with enhancements from Altos and Microsoft.

ORGANIZATION

This manual contains the miscellaneous utilities and files (M) of the Run-time system.

For commands, programs, and utilities (C), see the *Reference (C)*.

NOTE

The last section of the manual, "Change Information," summarizes the changes that have been made to the manual since the previous version.

MANUAL CONVENTIONS

The documentation conventions used in this manual are explained on the following page.

Symbol	Description
boldface type	What you type. For example: Type **tar tv**
boldface type	Used for command or parameter names that must be typed as shown. **mail** *user*
italic type	Variables (a value that can change), such as *user*. See the previous example. Also for manual titles, such as *Reference (C)* and *Reference (M)*.
Ctrl-d	Keys you press simultaneously (separated by a hyphen and shown in reverse type). For example: Ctrl-d means you press and hold the Ctrl key and then press the **d** key.
Esc **c**	Keys you press sequentially.
[ ]	Optional items in a syntax statement. If you do not use the optional item, the program selects a default action to carry out.
|	Use only one of the separated items.
...	Repeat preceding argument one or more times.
,,,	Repeat the preceding argument one or more times and separate arguments with a comma.
" "	Terms defined in the text. Quotation marks also indicate text from a source code example.

ADDITIONAL REFERENCE MATERIALS

For more information on your operating system, see the following list of manuals. To order a manual, call (408) 434-6688, ext. 3004 and give the manual title and part number.

Owner's Guide (part number 690-21264-*nnn* or 690-20351-*nnn*) describes how to connect computer components and peripherals, turn on power, and use the diagnostic programs.

Using the AOM Menu System (part number 690-18055-*nnn*) describes how to use the Altos Office Manager (AOM) to install software and manage the operating system.

Altos System V User's Guide (part number 690-21178-*nnn*) (not shipped with the Run-time system) explains basic operating system concepts and programs (e.g., vi, ed, sh, csh, mail, sed, and awk).

Altos System V Series 386 Operations Guide (part number 690-21171-*nnn*) tells how to set up the system for users and peripherals, maintain and back up the system, optimize system performance, and use uucp communications programs. This manual also contains system and LP spooler error messages.

Altos System V Series 386 Reference (C) (part number 690-22869-*nnn*) describes the Altos Run-time system commands, programs, and utilities.

Altos System V Series 386 Development System Set (part number 690-21585-000) contains reference and tutorial material.

Manuals in this set include:

Altos System V Series 386 C Compiler Library and User's Guide
Altos System V Series 386 C Compiler Language Reference
Altos System V Series 386 Programmer's Guide
Altos System V Series 386 Macro Assembler User's Guide and Reference
Altos System V Series 386 Reference (CP, S, F)

DOCUMENTER'S WORKBENCH (part numbers 690-15843-*nnn* and 690-15844-*nnn*) describes mm, nroff, troff, and typesetting functions and commands.

Contents
Miscellaneous (M)

getty	Sets terminal mode.
gettydefs	Speed and terminal settings used by getty.
group	Format of the group file.
infocmp	Compares or prints terminfo descriptions.
inir	Cleans the file system and executes init.
init	Process control initialization.
inittab	Script for the init processes.
inode	Format of an inode.
install	Installs commands.
keyboard	Series 500 system console keyboard.
layout	Manages hard disk partitions.
ldunix	Configurable kernel linker.
link, unlink	Links and unlinks files and directories.
log	Interface to STREAMS error logging and event tracing.
lpadmin	Configures the LP spooling system.
lpd	Line printer daemon.
lpinit	Adds new line printers to the system.
lpon, lpoff	Turns on/off lp printer schedulers.
lpsched, lpshut, lpmove	Starts/stops the LP request scheduler and moves requests.
makedevs	Creates special device files.
makekey	Generates an encryption key.
makettys	Creates tty special files.
master	Master configuration database.
mem, kmem	Memory image file.
menus	Format of a Business Shell menu system.
mkboot	Converts an object file to a bootable object file.
mkfs	Constructs a file system.
mkunix	Makes a bootable system file with kernel and driver symbol tables.
mnttab	Mounted file system table.
ncheck	Generates path names from inode numbers.
null	The null file.
options	Floppy disk installation menu.
passwd	The password file.
printers	Print spooler configuration file.
profile	Sets up an environment at login time.
pwck, grpck	Checks password/group file.

(BLANK)

Name

intro - Introduction to miscellaneous features and files.

Description

This section contains miscellaneous information for maintaining the entire system, including descriptions of files, devices, tables, and programs.

Name

acct - Format of per-process accounting file.

Description

Files produced as a result of calling **acct**(S) have records in the form defined by <sys/acct.h>.

In *ac_flag*, the AFORK flag is turned on by each **fork**(S) and turned off by an **exec**(S). 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 the current process size to *ac_mem* computed as follows:

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

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

See Also

acct(C), acct(S)

Notes

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

aliases - Alias file for mail.

Syntax

/usr/lib/mail/aliases

Description

This file describes user ID aliases that are used by the **/usr/lib/sendmail** command. It is fomatted as a series of lines of the form:

name: name_1, name_2, ... name_n

Name is the name to alias, and *name_n* are the aliases for that name. For example,

terry: pubs!terry

Lines beginning with white space are continuation lines. Lines beginning with # are comments.

Aliasing occurs only on local names. Loops cannot occur, since no message will be sent to any person more than once.

Aliases is only the raw data file; the actual aliasing information is placed in binary format in the file **/usr/lib/mail/aliases.hash** by executing the program **aliashash**(M). Each time you change the **aliases** file, run **aliashash** for the changes to take effect.

See Also

aliashash(M), mail(C)

Name

aliashash - Rebuild the data base for the mail alias file.

Syntax

aliashash

Description

Aliashash rebuilds the random access data base for the mail alias file **/usr/lib/mail/aliases.** For the change to take effect, run **aliashash** each time **/usr/lib/mail/aliases** is changed.

See Also

aliases(M)

Name

ascii - Map of the ASCII character set.

Description

Ascii is a map of the ASCII character set. It lists both octal and hexadecimal equivalents of each character. It contains:

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

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a nl	0b vt	0c np	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 $	25 %	26 &	27 '
28 (	29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [	5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c |	7d }	7e ~	7f del

Name

boot - Secondary bootstrap program.

Syntax

/boot

Description

The **boot** program brings up the operating system from a cold start. In addition to bringing an operating system file into memory from disk, **boot** also initializes all I/O subsystems and loads them with their operating software. The **boot** program keeps track of which I/O boards are present in a particular configuration and passes this information to the operating system.

The **boot** program is interactive and will prompt for the name of an operating system file you want to use. This permits the selection of one of several kernel files. Additionally, if the CPU Monitor sets the appropriate flag, **boot** will attempt to automatically boot the kernel file. If it cannot find this file, **boot** will go into interactive mode, and will display:

```
386 System Boot

hd(0,2)unix

Cannot open hd(0,2)unix

Enter bootable program or ? for help
```

For example, if you type **?** Retn, you will see the following display:

```
Boot n.n.nnn of Jan 7, 1987

Type the full name of the program you wish to boot.
The default boot program is: hd(0,2)unix.
Press ESC to start with default name.
Press @ to erase line, or backspace to erase a character.
Press the DEL key to abort the boot once it has started.

Unix's on hd(0,2) are:
hd(0,2)unix
hd(0,2)unix.net

Enter bootable program or '?' for Help.
```

Enter the bootable program in the form:

root_device(disk,partition)kernel_file

where *root_device* is:

fd - floppy disk
hd - main hard disk

The *disk* field selects a disk number which contains a file system in which to find I/O subsystem download code and kernel files. On the floppy, this is always 0. On the hard disk, the disk is usually the root file system, 0 for the root disk. The *partition* field specifies which partition contains the root file system, and is always 0 for the floppy, and normally 2 for hard disk.

The *kernel_file* field is the path name of a kernel file (relative to /).

Typical invocations are:

fd(0,0)unix.fd
hd(0,2)unix

The **boot** program searches the file system in which it found the kernel for code files to load into the I/O subsystem boards. Those files are always kept in the /etc directory and are named:

/etc/dlcode/ioc	generic IOC code (Series 1000 only)
/etc/dlcode/fp.*type*	file processor specific (Series 2000 only)
/etc/dlcode/sio	generic SIO code (Series 2000 only)
/etc/dlcode/sio[0123]	board-specific SIO code (Series 2000 only)
/etc/dlcode/mdc	generic MDC code (Series 2000 only)
/etc/dlcode/mdc[0123]	board-specific MDC code (Series 2000 only)

If **boot** can find a file that corresponds to a particular SIO or MDC board, it will load that file; otherwise, it loads the generic code file.

Files

/boot	Secondary boot
/unix*	Operating system kernel file
/etc/dlcode/fp.esdi	File processor download code for ESDI drive
/etc/ldlcode/fp	File processor download code for ST506 drive
/etc/dlcode/sio	SIO generic code
/etc/dlcode/sio?	SIO specific code
/etc/dlcode/mdc	SIO generic code
/etc/dlcode/mdc?	SIO specific code

See Also

layout(M)

Diagnostics

For an error, **boot** displays an error message, then returns to its prompt. The following is a list of the most common messages.

bad drive specifier *x*
: An invalid drive number was given, only 0-2 are valid.

bad superblock: s_magic *x*
: The partition given doesn't appear to have a filesystem on it.

device error, status [0123]
: An error occurred while trying to read the program. The boot system retries up to 10 times on each error. If all 10 attempts fail, the following message appears: "Fatal disk error (10) retries."

pathname not found
: The supplied pathname does not correspond to an existing file.

Notes

Boot cannot be used to load programs that have not been linked for stand-alone execution.

Name

brc - System initialization procedures.

Syntax

/etc/brc

Description

These shell procedures are executed via entries in **/etc/inittab** by **init**(M) whenever the system is booted (or rebooted).

The **brc** procedure clears the mounted file system table, /etc/mnttab, and puts the entry for the root file system into the mount table.

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

See Also

fsck(C), init(M), rc2(M), shutdown(C)

Name

captoinfo - Converts a termcap description into a terminfo description.

Syntax

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

Description

Captoinfo looks in *file* for **termcap**(M) descriptions. For each one found, an equivalent **terminfo**(M) description is written to standard output, along with any comments found. A description that 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.

Options

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

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

-1 Cause the fields to print 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**(C) 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

tic(C), terminfo(M) and curses(S), in the *Reference (CP, S, F)*

Notes

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

Name

checklist - Lists file systems processed by fsck.

Description

The **/etc/checklist** file contains a list of the file systems to be checked when **fsck**(C) is invoked without arguments. The list contains at most 15 special file names. Each special file name must be on a separate line and must correspond to a file system.

See Also

fsck(C)

Name

ckbupscd - Checks file system backup schedule.

Syntax

/etc/ckbupscd [-m]

Description

Ckbupscd consults the file /etc/bupsched and prints the file system lists from lines with date and time specifications matching the current time. If the **-m** flag is present an introductory message in the output is suppressed so that only the file system lists are printed. Entries in the /etc/bupsched file are printed under the control of **cron**(C).

The file /etc/bupsched should contain lines of 4 or more fields, separated by spaces or tabs. The first 3 fields (the schedule fields) specify a range of dates and times. The rest of the fields constitute a list of names of file systems to be printed if **ckbupscd** is run at some time within the range given by the schedule fields. The general format is:

time[,*time*] *day*[,*day*] *month*[,*month*] *fsyslist*

where:

time Specifies an hour of the day (*0* through *23*), matching any time within that hour, or an exact time of day (*0:00* through *23:59*).

day Specifies a day of the week (*sun* through *sat*) or day of the month (*1* through *31*).

month Specifies the month in which the time and day fields are valid. Legal values are the month numbers (*1* through *12*).

fsyslist The rest of the line is taken to be a file system list to print.

Multiple time, day, and month specifications may be separated by commas, in which case they are evaluated left to right.

An asterisk (*) always matches the current value for that field.

A line beginning with a sharp sign (#) is interpreted as a comment and ignored.

The longest line allowed (including continuations) is 1024 characters.

Examples

The following are examples of lines which could appear in the /etc/bupsched file.

06:00-09:00 fri 1,2,3,4,5,6,7,8,9,10,11 /applic

Prints the file system name */applic* if **ckbupscd** is run between 6:00am and 9:00am any Friday during any month ex cept December.

00:00-06:00,16:00-23:59 1,2,3,4,5,6,7 1,8 /

Prints a reminder to backup the root (/) file system if **ckbupscd** is run between the times of 4:00pm and 6:00am during the first week of August or January.

Files

/etc/bupsched	Specification file containing times and file system to back up

See Also

cron(C), echo(C), sh(C), sysadm(C)

Notes

Ckbupscd will report file systems due for backup if invoked any time in the window. It does not know that bac ups may have just been made.

Name

clock - Provides access to the time-of-day chip.

staticram - Provides 16 bytes of battery-backed-up memory.

Description

The file **/dev/clock** provides access to the time-of-day chip. The current time, date, and year can be read or written as ASCII data. (See for example, the -s option of **date.**) The date is stored in the form:

MMDDhhmmYY

Where *MM* is the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minute, and *YY* is the last 2 digits of the year.

The clock is maintained by a battery, even when the power is off. The clock is normally used to set the system's idea of the date on every power-up.

The file **/dev/staticram** provides 16 bytes of battery-backed-up memory, which is actually part of the time-of-day chip. It may be used for anything the system administrator wishes, such as a system ID code, etc. This memory remains valid until the battery wears out, or until it is rewritten.

Name

clone - Opens 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**(S)) require no further involvement of **clone.**

Notes

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

See Also

log(M), and the *STREAMS Programmer's Guide*

Name

clri - Clears an inode.

Syntax

/etc/clri *special i-number...*

Description

Clri writes nulls on the 64 bytes at offset *i-number* from the start of the inode list. This effectively eliminates the inode at that address. *Special* is the device name on which a file system has been defined. After **clri** is executed, any blocks in the affected file will show up as "not accounted for" when **fsck**(C) is run against the file-system. The inode may be allocated to a new file.

Read and write permission is required on the specified *special* device.

This command is used to remove a file which appears in no directory; that is, to get rid of a file which cannot be removed with the **rm**(C) command.

See Also

fsck(C), fsdb(M), ncheck(M), rm(C), and fs(F) in the *Reference (CP, S, F)*

Notes

If the file is open for writing, **clri** will not work. The file system containing the file should NOT be mounted.

If **clri** is used on the inode number of a file that does appear in a directory, it is imperative to remove the entry in the directory at once, since the inode may be allocated to a new file. The old directory entry, if not removed, continues to point to the same file. This sounds like a link, but does not work like one. Removing the old entry destroys the new file.

Name

crash - Examines system images.

Syntax

/etc/crash [**-d** *dumpfile*] [**-n** *namelist*] [**-o** *offset*]
[**-w** *outputfile*]

Description

The **crash** command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to **crash** are *dumpfile*, *namelist*, *offset*, and *outputfile*.

The *dumpfile* is the file containing the system memory image. The default *dumpfile* is /dev/kmem. The system image can also be /dev/hd0.restart if the system is in a panic state.

The text file *namelist* contains the symbol table information needed for symbolic access to the system memory image to be examined. The default *namelist* is /unix. If a system image from another machine is to be examined, the corresponding text file must be copied from that machine.

The *offset* option offsets from the beginning of *dumpfile* at which data starts. This is useful with /dev/hd0.restart where *offset* is 1024.

When the **crash** command is invoked, a session is initiated. The output from a **crash** session is directed to *outputfile*. The default *outputfile* is the standard output.

Input during a **crash** session is of the form:

function [*argument* ...]

where *function* is one of the **crash** functions described in the Functions section of this command description, and *arguments* are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to **crash** functions wherever they are semantically valid.

-e	Display every entry in a table.
-f	Display the full structure.
-p	Interpret all address arguments in the command line as *physical* addresses.
-s *process*	Specify a process slot other than the default.
-w *file*	Redirect the output of a function to *file*.

Note that if the **-p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions *mode*, *defproc*, and *redirect* correspond to the function options **-p**, -s, and **-w**. The *mode* function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; *defproc* sets the value of the process slot argument for subsequent functions; and *redirect* redirects all subsequent output.

Output from **crash** functions may be piped to another program in the following way:

function [*argument* ...]!*shell_command*

For example,

mount ! grep rw

will write all mount table entries with an **rw** flag to the standard output. The redirection option (-w) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table slot arguments are always decimal. Table slot arguments larger than the size of the function table will not be interpreted correctly. Use the **findslot** function to translate from an address to a table slot number. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by 0x and as octal if it is preceded by 0. Decimal override is designated by 0d, and binary by 0b.

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as **p** for proc, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number or a range. A range of slot numbers may be specified in the form:

a-b

where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, -, *, /, &, or |. An operand that is a number should be preceded by a radix prefix if it is not a decimal number (0 for octal, 0x for hexadecimal, 0b for binary). The expression must be enclosed in parentheses (). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to **crash** functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = table entry|range

start_addr = address|symbol|expression

Functions

?[-w *file*]
List available functions.

!*cmd*
Escape to the shell to execute a command.

< *filename*
Take input from *filename* until end-of-file (EOF). Lines starting with a "#" are comments and are ignored.

adv [-e] [-w *file*] [[-p] *table_entry...*]
Print the advertise table.

base [-w *file*] *number ...*
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other than decimal should be preceded by a prefix that indicates its radix as follows: 0x, hexadecimal; 0, octal; and 0b, binary.

buffer [-w *file*] [-*format*] *bufferslot*

or

buffer [-w *file*] [-*format*] [-p]*start_addr*
Alias: **b**.

Print the contents of a buffer in the designated format, where *format* can be:

-b	byte
-c	character
-d	decimal
-x	hexadecimal
-o	octal
-r	directory
-i	inode

If no format is given, the previous format is used. The default format at the beginning of a **crash** session is hexadecimal.

bufhdr [-**f**] [-w *file*] [[-**p**]*table_entry* ...]
Alias: **buf.**
Print system buffer headers.

callout [-w *file*]
Alias: **c.**
Print the callout table.

dballoc [-w *file*] [*class*...]
Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed.

dbfree [-w *file*] [*class*...]
Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [-**e**] [-w *file*] [*class*...]

or

dblock [-**e**] [-w *file*] [[-**p**] *table_entry*...]
Print allocated streams data block headers. If the class option (-**c**) is used, only data block headers for the class specified will be printed.

defproc [-w *file*] [-**c**]

or

defproc [-w *file*] [*slot*]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (-**c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a **crash** session, the process slot is set to the current process.

dis [-w *file*] [-**a**] *start_address*[*count*]
Disassemble from the start address for *count* instructions. The default count is 1. The absolute option (-**a**) specifies a nonsymbolic disassembly.

ds [-w *file*] *virtual_address* ...
Print the data symbol whose address is closest to, but not greater than, the address entered.

file [-e] [-w *file*] [[-p]*table_entry* ...]
Alias: **f.**
Print the file table.

findaddr [-w *file*] *table slot*
Print the address of *slot* in *table.* Only tables available to the **size** function are available to **findaddr.**

findslot [-w *file*] *virtual_address* ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the **size** function are available to **findslot.**

fs [-w *file*] [[-p] *table_entry*...]
Print the file system information table.

gdp [-e] [-f] [-w *file*] [[-p] *table_entry*...]
Print the gift descriptor protocol table.

gdt [-e] [-w *file*] [[-p] *table_entry*...]
Print the global descriptor table.

help [-w *file*] **-a** *function* ...
Print a description of the named function, including syntax and aliases. The **-a** option lists all functions.

idt [-e] [-w *file*] [[-p] *table_entry* ...]
Print the interrupt descriptor table.

inode [-e] [-f] [-w *file*] [[-p]*table_entry* ...]
Alias: **i.**
Print the inode table, including file system switch information.

kfp [-w *file*] [*value*]
Print the frame pointer for the start of a kernel stack trace. If the *value* argument is entered, the **kfp** is set to that value.

lck [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...
Alias: **l**.
Print record-locking information. If the **-e** option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

ldt [**-e**] [**-w** *file*] [**-s** *process*] [[**-p**] *table_entry* ...]
Print the local descriptor table for the given process, or for the current process if none is given.

linkblk [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the linkblk table.

map [**-w** *file*] *mapname* ...
Print the map structure of the given *mapname*.

mbfree [**-w** *file*]
Print free streams message block headers.

mblock [**-e**] [**-w** *filename*] [[**-p**]*table_entry* ...]
Print allocated streams message block headers.

mode [**-w** *file*] [*mode*]
Set address translation of arguments to virtual (**v**) or physical (**p**) mode. If no mode argument is given, the current mode is printed. At the start of a **crash** session, the mode is virtual.

mount [**-e**] [**-w** *file*] [[**-p**]*table_entry* ...]
Alias: **m**.
Print the mount table.

nm [**-w** *file*] *symbol* ...
Print value and type for the given symbol.

od [**-p**] [**-w** *file*] [*-format*] [*-mode*] [**-s** *process*] *start_addr*[*count*]
Alias: **rd**.
Print *count* values starting at the start address in one of the following formats: character (**-c**), decimal (**-d**), hexadecimal (**-x**), octal (**-o**), ascii (**-a**), or hexadecimal/character (**-h**), and one of the following modes: long (**-l**), short (**-t**), or byte (**-b**). The default mode for character and ascii formats is byte; the default mode for decimal, hexadecimal, and octal

formats is long. The format **-h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When *format* or *mode* is omitted, the previous value is used. At the start of a **crash** session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

pagemode [**-l** *lines*] [**-on**|**-off**]
Toggle **pagemode**. If on, pause after every *lines* (24 by default). Similar to **more**(C).

panic
Print the latest system notices, warnings and panic messages from the limited circular buffer kept in memory.

pcb [-w *file process*]
Print the process control block (TSS) for the given process. If no arguments are given, the active TSS for the current process is printed.

pdt [-e] [-w *file*] [-s *process*] *section segment*

or

pdt [-e] [-w *file*] [-s *process*] [-p] *start_addr*[*count*]
The page descriptor table starting at the start address for *count* entries is printed. If no count is entered, 1 is assumed.

pfdat [-e] [-w *file*] [[-p]*table_entry* ...]
Print the pfdata table.

proc [-f] [-w *file*] [[-p]*table_entry* ... *#procid* ...]

or

proc [-f] [-w *file*] [-r]
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a #. Alternatively, process table information for runnable processes may be specified with the runnable option (**-r**). The full option (**-f**) details most of the information in the process table as well as the pregion table for that process.

qrun [-w *file*]
Print the list of scheduled streams queues.

queue [-e] [-w *file*] [[-p]*table_entry...*]
Print streams queues.

quit
Alias: **q**.
Terminate the **crash** session.

rcvd [-e] [-f] [-w *file*] [[-p]*table_entry...*]
Print the receive descriptor table.

redirect [-w *file*] [-c]

or

redirect [-w *file*] [*file*]
Used with a file name, redirect output of a **crash** session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (-c) closes the previously set file and redirects output to the standard output.

region [-e] [-f] [-w *file*] [[-p]*table_entry* ...]
Print the region table.

search [-p] [-w *file*] [-m *mask*] [-s *process*] *pattern start_addr count*
Print the long words in memory that match *pattern*, beginning at the start address for *count* long words. The *mask* is anded (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

size [-w *file*] [-x] [*structure_name* ...]
Print the size of the designated structure. The -x option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [-e] [-w *file*] [[-p]*table_entry...*]
Print the send descriptor table.

srmount [-e] [-w *file*] [[-p]*table_entry...*]
Print the server mount table.

stack [-w *file*] [*process*]
Alias: **s.**

Dump stack. If no arguments are entered, the kernel stack for the current process is printed. Otherwise, the kernel stack for the given process is printed.

stream [**-e**] [**-f**] [-w *file*] [[-p]*entry_table...*]

Print the streams table.

strstat [-w *file*]

Print streams statistics.

trace [-w *file*] [-r] [*process*]

Alias: **t.**

Print stack trace. The **kfp** value is used with the **-r** option.

ts [-w *file*] *virtual_address* ...

Print closest text symbol to the designated address.

tty [-e] [**-f**] [-w *file*] [-t*type*[[-p]*table_entry* ...]]

Valid types: mdc, sc, kd

Print the tty table. If no arguments are given, the tty table for mdc is printed. If the **-t** option is used, the table for the single tty type specified is printed. If no argument follows the *type* option, all entries in the table are printed. A single tty entry may be specified from the start address.

user [-f] [-w *file*] [*process slot*]

Alias: **u.**

Print the ublock for the designated process.

var [-w *file*]

Alias: **v.**

Print the tunable system parameters.

vtop [-w *file*] [-s *process*] *start_addr* ...

Print the physical address translation of the virtual start address.

Files

/dev/kmem	System image of currently running system
/dev/hd0.restart	Used to access the saved system image on hard disk.

See Also

sh(C), test(C)

Name

default - Default program information directory.

Description

The files in the /etc/default directory contain the default information used by system commands such as **lpd**(M) and **remote**(C). Default information is any information required by the command that is not explicitly given when the command is invoked.

The directory may contain zero or more files. Each file corresponds to one or more commands. A command searches for a file whenever it has been invoked without sufficient information. Each file contains zero or more entries which define the default information. Each entry has one of the following forms:

keyword or *keyword=value*

where *keyword* identifies the type of information available and *value* defines its value. Both *keyword* and *value* consist of letters, digits, and punctuation. The exact spelling of *keyword* and the appropriate *value* depends on the command and are described with the individual commands.

Any line in a file beginning with a number sign (#) is considered a comment and is ignored.

Files

/etc/default/lpd
/etc/default/passwd
/etc/default/quot
/etc/default/su

See Also

lpr(C), quot(C), su(C)

Name

df - Reports number of free disk blocks and inodes.

Syntax

df [-lt] [-f] [*file-system* | *directory* | *mounted-resource*]

Description

The **df** command prints out the number of free blocks and free inodes in mounted file systems, directories, or mounted resources by examining the counts kept in the super-blocks.

File-system may be specified either by device name (e.g., /dev/hd1b) or by mount point directory name (e.g., /usr).

Directory can be a directory name. The report presents information for the device that contains the directory.

Mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

The **df** command uses the following options:

-l Reports on local file systems only.

-t Causes the figures for total allocated blocks and inodes to be reported as well as the free blocks and inodes.

-f An actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free inodes are not reported). This option will not print any information about mounted remote resources.

Note

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

Files

/dev/*
/etc/mnttab

See Also

mount(M) mnttab(M), and fs(F) in the *Reference (CP, S, F)*

Name

dir - Format of a directory.

Syntax

#include <sys/dir.h>

Description

A directory behaves exactly like an ordinary file, except 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 inode entry (see **filesystem**(M)). The structure of a directory is given in the include file /usr/include/sys/dir.h.

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

The first 2 bytes of each entry are the inode numbers, which will be zero if the entry has been removed. The next 14 bytes are the filename. If the name is exactly 14 bytes, there will be no terminating null byte.

See Also

dir(S), filesystem(M)

Name

display - Series 500 system console display.

Description

The system console (and user's terminal) is composed of two separate pieces: the keyboard (see **keyboard**(M)) and the display. Because of their complexity and because there are two possible display interfaces (the monochrome and color/graphics adapters), they are discussed in separate manual entries.

The display normally consists of 25 lines of 80 columns each; 40-column lines are also supported by the color/graphics adapter. Writing characters to the console (**/dev/console**) has an effect that depends on the characters. All characters written to **/dev/console** are first processed by the terminal interface (see **termio**(M)). For example, mapping new-line characters to carriage return plus new-line and expanding tabs to spaces will be done before the following processing:

x	Where *x* is not one of the following, displays *x*.
BEL	Generates a bell (audible tone, no modulation).
CR	Places the cursor at column 1 of the current line.
LF,VT	Places the cursor at the same column of the next line (scrolls if the current line is line 25).
FF	Clears the screen and places the cursor at line 1, column 1.
BS	Depends on the previous character: if a _ (underscore), see below; otherwise, if the cursor is not at column 1, it is moved to the left one position on the same line. If the cursor is at column 1 but not line 1, it is moved to column 79 of the previous line. Finally, if the cursor is at column 1, line 1, it is not moved.

_BSx — Sets the underscore attribute for the character *x* to be displayed. The underscore attribute for the color/graphics adapter is a red background with a white foreground.

ESCx — Where *x* is any of the 256 possible codes (except for c and [), displays that value interpreted. This is useful for using the full set of graphics available on the display. Note again that the characters are processed through the terminal interface prior to this escape sequence. Therefore, to get some of the possible 256 characters, it is necessary that the character not be post processed. The easiest way to accomplish this is to turn off OPOST in the c_oflag field (see **termio**(M)); however, this may have other side effects.

This display can be controlled by means of ANSI X3.64 escape sequences, which are specified sequences of characters, preceded by the ASCII character ESC. The escape sequences, which work on either the monochrome or color/graphics adapter, are the following:

ESCc — Clears the screen and places the cursor at line 1, column 1.

ESC[*n* @ — Insert character - inserts *n* blank places for *n* characters at the current cursor position.

ESC[*n* A — Cursor up - moves the cursor up *n* lines (default: *n*=1).

ESC[*n* B — Cursor down - moves the cursor down *n* lines (default: *n*=1).

ESC[*n* C — Cursor right - moves the cursor right *n* columns (default: *n*=1).

ESC[*n* D — Cursor left - moves the cursor left *n* columns (default: *n*=1).

ESC[*n* E — Cursor next line - moves the cursor to column 1 of the next line, then down *n*-1 lines (default: *n*=1).

ESC[*n* F	Cursor previous line - moves the cursor to column 1 of the current line, then up *n* lines (default: *n*=1).
ESC[*n* G	Cursor horizontal position - moves the cursor to column *n* of the current line (default: *n*=1).
ESC[*n* ; *m* H	Position cursor - moves the cursor to column *m* of line *n* (default: *n*=1).
ESC[*n* J	Erase window - erases from the current cursor position to the end of the window if *n*=0, from the beginning of the window to the current cursor position if *n*=1, and the entire window if *n*=2 (default: *n*=1).
ESC[*n* K	Erase line - erases from the current cursor position to the end of the line if *n*=0, from the beginning of the line to the current cursor position if *n*=1, and the entire line if *n*=2 (default: *n*=1).
ESC[*n* L	Inserts *n* lines at the current cursor position (default: *n*=1).
ESC[*n* M	Deletes *n* lines at the current cursor position (default: *n*=1).
ESC[*n* P	Deletes *n* characters from a line starting at the current cursor position (default: *n*=1).
ESC[*n* S	Scroll up - scrolls the characters in the current window up *n* lines. The bottom *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* T	Scroll down - scrolls the characters in the current window down *n* lines. The top *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* X	Erase character - erases *n* character positions starting at the current cursor position (default: *n*=1).

ESC [*Ps* ; *Ps*; ... *m*

Character attributes - each *Ps* is one of the following characters; multiple characters are separated by semicolons. These parameters apply to successive characters being displayed, in an additive manner (e.g., both bold and underscoring can be selected). Only the parameters through 7 apply to the monochrome adapter; all parameters apply to the color/graphics adapter. (Default: *Ps*=0.)

Ps	Meaning		
0	all attributes off (normal display) (white foreground with black background)		
1	bold intensity		
4	underscore on (white foreground with red background on color)		
5	blink on		
7	reverse video		
10	selects the primary font		
11	selects the first alternate font; lets ASCII characters less than 32 be displayed as ROM characters		
12	selects a second alternate font; toggles high bit of extended ASCII code before displaying as ROM characters		
30	black	(gray)	foreground
31	red	(light red)	foreground
32	green	(light green)	foreground
33	brown	(yellow)	foreground
34	blue	(light blue)	foreground
35	magenta	(light magenta)	foreground
36	cyan	(light cyan)	foreground
37	white	(bright white)	foreground
40	black	(gray)	background
41	red	(light red)	background
42	green	(light green)	background
43	brown	(yellow)	background
44	blue	(light blue)	background
45	magenta	(light magenta)	background
46	cyan	(light cyan)	background
47	white	(bright white)	background

Note that for character attributes 30 through 37, the color selected for foreground will depend on whether the bold intensity attribute (1) is currently on. If not, the first color listed will result; otherwise the second color listed will result.

Similarly, for character attributes 40-47, the color selected for background will depend on whether the blink attribute (5) is currently on. The color selected for background also depends on whether blinking is enabled in color mode byte or no blinking is selected (see the MODE_BLINK and MODE_BG16 bits in the color mode byte defined below). If the blink attribute is not on, then the first color listed will result. If the blink attribute is not on, and blinking is enabled, then the first color listed will result and it will blink. If the blink attribute is on, and no blinking is enabled, then the second color listed will result.

Ioctl calls

The display driver supports **ioctl**(S) calls of the form:

ioctl(*filedes*, *command*, *arg*)

filedes is a valid open file descriptor.
command is one of the commands listed below.
arg is the argument of *command*. The type of *arg* is specific to the command in use.

The following is a list of valid **ioctl** commands for display adapters. These commands and structures are defined in sys/kd.h.

KDDISPTYPE
Returns information about the current display adapter. The argument is the address of a structure (defined in sys/kd.h) of the following type:

```
struct kd_disparam {
        long type;
        char *addr;
        ushort ioaddr[];
}
```

type describes the type of adapter installed, and is one of: KD_MONO, KD_HERCULES, KD_CGA, or KD_EGA.

addr is the physical address of the display memory for this adapter.

ioaddr is a list of I/O addresses valid for this adapter.

KDGETMODE
Returns the current display mode. *Arg* is an integer, whose values are one of the following:

KD_TEXT	Text mode
KD_GRAPHICS	Graphics Mode

KDSETMODE
Sets the current display mode. *Arg* is an integer, whose values are one of those defined above for KDGETMODE. Note, the user is responsible for programming the color/graphics adapter registers for the appropriate graphical state.

KDADDIO
Adds I/O port address to list of valid video adapter addresses. Argument is an unsigned short type which should contain a valid port address for the installed video adaptor.

KDDELIO
Deletes I/O port address from list video adaptor addresses. Argument is an unsigned short type which should contain a valid port address for the installed video adaptor.

KDENABIO
Enables ins and outs to video adaptor ports. No argument.

KDDISABIO
Disables ins and outs to video adaptor ports. No argument.

KDMAPDISP

Maps the display memory for the current adapter in the user's data space. Argument is a pointer to structure type "kd_memloc." Structure definition is:

```
struct kd_memloc {
    char *vaddr;    /* virtual address to map to */
    char *physaddr;/* physical address to map from */
    long length;    /* size in bytes to map */
    long ioflg;     /* enable i/o addresses if set */
}
```

vaddr contains a paged-aligned virtual address in the user's data space. To map the display memory for a monographic adapter requires 4 Kbytes. In order to map the display memory, the user must first use KDSETMODE to place the adapter into graphics mode and also use the VT_SETMODE option (see **vt**(M)) to set the virtual terminal mode to VT_PROCESS. Included in this section is a sample code fragment showing how to correctly map the screen memory into user data space.

KDUNMAPDISP

Unmaps the display adapter memory from user data space.

The following code fragment details how to map the display adapter memory into user data so the screen can be accessed via memory references in user code.

```
#include <sys/types.h>
#include <sys/immu.h>
#include <sys/at_ansi.h>
#include <sys/kd.h>
#include <sys/vt.h>

unsigned char d[0x2000]; /* allocate 2 pages of data */
unsigned char *c;
int fd;
```

```
struct vt_mode vt;
struct kd_memloc mp;
struct screen {
     char ch;
     char attr;
} *scr;
     .
     . .
     . . .

     /* assign a page-aligned address.
      * Starting in the middle of a 2*pagesize array assures
      * it will contain 1 page-aligned address with 1 page of
      * data following.
      */
     c = (unsigned char *)((long)(&d[sizeof(d)/2]) & (NBPP-1));
     if (ioctl(fd,VT_GETMODE,&vt) == -1 )
          exit(1);
     vt.mode  = VT_PROCESS;

     /* set virtual terminal process control mode */
     if (ioctl(fd,VT_SETMODE,&vt) == -1 )
          exit(1);

     /* set adapter in graphics mode */
     if (ioctl(fd,KDSETMODE,KD_GRAPHICS) == -1 )
          exit(1);

     /* virtual address to map to */
     mp.vaddr = (char unsigned *)c;
     /* start of monographic display memory */
     mp.physaddr = (char *)MONO_BASE;
     /* length of monograph display memory */
     mp.length = (long)MONO_SIZE;
     mp.ioflg = (long)0;

     /* map the display memory into user data space */
     if (ioctl(fd,KDMAPDISP,&mp))
          exit(1);

     /* start of screen memory */
     scr = (struct screen *)c;
```

```
/* The layout of screen memory is:
 *      For each character:
 *              1 data byte
 *              1 attribute byte
 */
scr->ch = ...
scr->attr = ....

/* Unmap display and reset modes */
ioctl(fd,KDUNMAPDISP);
ioctl(fd,KD_TEXT);
vt.mode  = VT_AUTO;
ioctl(fd,VT_SETMODE,&vt);
...
..
.
```

VT_OPENQRY
VT_GETMODE
VT_SETMODE
VT_RELDISP
VT_ACTIVATE
: These **ioctl**(S) options are used for controlling virtual terminals. Refer to **vt**(M) for their definitions.

Files

/dev/console

See Also

stty(C), ioctl(S), keyboard(M), termio(M), vt(M)

Name

environ - The user environment.

Description

The user environment (environ) is a collection of information about a user, such as his login directory, mailbox, and terminal type. The environment is stored in special "environment variables," which can be assigned character values, such as names of files, directories, and terminals. These variables are automatically made available to programs and commands that you can invoke. The commands can then use the values to access your files and terminal.

Options

HOME
: Names the user's login directory. Initially, HOME is set to the login directory given in the user's passwd file entry.

PATH
: Defines the search path for the directories containing commands. The system searches these directories whenever a user types a command without giving a full pathname. The search path is one or more directory names separated by colons (:). Initially, PATH is set to :/bin:/usr/bin.

TERM
: Defines the type of terminal being used. This information is used by commands such as **more** which rely on information about the capabilities of the user's terminal. The variable may be set to any valid terminal name (see **terminals**(M)) directly or by using the **tset**(C) command.

TZ
: Defines time zone information. This information used by **date**(C) to display the appropriate time. The variable may have any value of the form *xxxnzzz* where *xxx* is standard local time zone abbreviation, *n* is the difference in hours from GMT, and *zzz* is the daylight-saving local time zone abbreviation (if any). For example, EST5EDT. The difference for a location east of England can be given as a negative number.

The environment can be changed by assigning a new value to a variable. An assignment has the form

name=value

For example, the assignment:

TERM=altos3

sets the TERM variable to an Altos III. When using the standard shell (**sh**(C)), the new value can be "exported" to each subsequent invocation of a shell by exporting the variable with the **export** command (see **sh**(C)) or by using the **env**(C) command. Users of the C-shell (**csh**(C)) can set and export a variable with the **setenv** command (see **csh**(C)).

A user may also add variables to the environment, but must be sure that the new names do not conflict with exported shell variables such as MAIL, PS1, PS2, and IFS. Placing assignments in the **.profile** file is a useful way to change the environment automatically before a session begins.

Note that the environment is made available to all programs as a string of arrays. Each string has the form:

name=value

where the *name* is the name of an exported variable and the *value* is the variable's current value. For programs started with a **exec**(S) call, the environment is available through the external pointer **environ.** For other programs, individual variables in environment are available through **getenv**(S) calls.

See Also

login(C), sh(C), profile(M), and getenv(S) in the *Reference (CP, S, F)*

Name

errprint - Displays error log contents.

Syntax

/etc/errprint [*date*]

Description

Errprint displays the error messages logged by the **strerr**(M) daemon for a particular date. The optional *date* argument may be specified on the command line in any of the following formats:

mm dd
mm-dd
mm/dd
monthname dd

If no date is specified, the current date is used.

For ease of viewing, the only fields displayed for each error message are the time of day and the text of the message. The remaining fields found in the log file are not displayed. The output is automatically piped through **more**(C).

Files

/usr/adm/streams/error.*mm-dd*
/usr/lib/errstrip.awk

See Also

strerr(M)

Name

ff - Fast find: lists file names and statistics for a file system.

Syntax

/etc/ff [*options*] *special*

Description

Ff reads the i-list and directories of the *special* file, assuming it is a file system. Inode data is saved for files which match the selection criteria. Output consists of the path name for each saved inode, plus other file information requested using the print options below. Output fields are positional. The output is produced in inode order; fields are separated by tabs. The default line produced by **ff** is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the option descriptions that follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, -*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

-I Do not print the inode number after each path name.

-l Generate a supplementary list of all path names for multiply-linked files.

-p *prefix* The specified *prefix* will be added to each generated path name. The default is . (dot).

-s Print the file size, in bytes, after each path name.

-u	Print the owner's login name after each path name.
-a *n*	Select if the inode has been accessed in *n* days.
-m *n*	Select if the inode has been modified in *n* days.
-c *n*	Select if the inode has been changed in *n* days.
-n *file*	Select if the inode has been modified more recently than the argument *file*.
-i *inode-list*	Generate names for only those inodes specified in *inode-list*.

See Also

find(C), ncheck(M)

Notes

If the **-l** option is not specified, only a single path name out of all possible ones is generated for a multiply-linked inode. If **-l** is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

Name

filesystem - Format of a system volume.

Syntax

#include <sys/filsys.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/inode.h>
#include <sys/ino.h>

Description

Every file system storage volume (e.g., a hard disk) has a common format for certain vital information. Every such volume is divided into a certain number of 512 byte sectors. Sector 0 is unused and is available to contain a bootstrap program or other information.

Sector 1 is the super-block. The format of a super-block is described in /usr/include/sys/filsys.h. In that include file, *s_isize* is the address of the first data block after the i-list. The i-list starts in logical block 2; thus the i-list is *s_isize*-2 blocks long. *S_fsize* is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers. If an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the console. Moreover, the free array is cleared so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The *s_free* array contains, in *s_free*[1], ..., *s_free*[*s_nfree*-1], up to NICFREE-1 numbers of free blocks *S_free*[0] is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to NICFREE) of free-block numbers listed in the next NICFREE longs of this chain member. The first of these NICFREE blocks is the link to the next member of the chain. To allocate a block: decrement *s_nfree*, and the new block is *s_free*[*s_nfree*]. If the new block number is 0, there are no blocks left, so give an error. If *s_nfree* becomes 0, read in the block named by the new block number, replace *s_nfree* by its first word, and copy the block numbers in the next NICFREE longs into the *s_free* array. To free a block, check if *s_nfree* is 50; if so, copy *s_nfree* and the *s_free* array into it, write it out, and set *s_nfree* to 0. In any event set *s_free*[*s_nfree*] to the freed block's number and increment *s_nfree*.

S_tfree is the total free blocks available in the file system.

S_ninode is the number of free i-numbers in the *s_inode* array. To allocate an inode: if *s_ninode* is greater than 0, decrement it and return *s_inode*[*s_ninode*]. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the *s_inode* array, then try again. To free an inode, provided *s_ninode* is less than NICINOD, place its number into *s_inode*[*s_ninode*] and increment *s_ninode*. If *s_ninode* is already NICINOD, do not bother to enter the freed inode into any table. This list of inodes only speeds up the allocation process. The information about whether the inode is really free is maintained in the inode itself.

S_tinode is the total number of free inodes available in the file system.

S_flock and *s_ilock* are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of *s_fmod* on disk is also immaterial, and is used as a flag to indicate that the superblock has changed and should be copied to the disk during the next periodic update of file system information.

S_ronly is a read-only flag used to indicate write-protection.

S_time is the last time the super-block of the file system was changed, and is a doubleprecision representation of the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the *s_time* of the super-block for the root file system is used to set the system's idea of the time.

I-numbers begin at 1, and the storage for inodes begins in logical block 2. Inodes are 64 bytes long. Inode 1 is reserved for future use. Inode 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each inode represents one file. For the format of an inode and its flags, see *ino.h*.

Files

/usr/include/sys/filsys.h
/usr/include/sys/stat.h
/usr/include/sys/types.h
/usr/include/sys/param.h
/usr/include/sys/inode.h
/usr/include/sys/ino.h

See Also

fsck(C), mkfs(M)

Name

finc - Fast incremental backup.

Syntax

/etc/finc [*selection-criteria*] *file-system raw-tape*

Description

Finc selectively copies the input *file-system* to the output *raw-tape*. The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by **labelit**(C). The selection is controlled by the *selection-criteria*, accepting only those inodes/files for whom the conditions are true.

It is recommended that production of a **finc** tape be preceded by the **ff**(M) command, and the output of **ff** be saved as an index of the tape's contents. Files on a **finc** tape may be recovered with the **frec**(M) command.

The argument *n*, in the *selection-criteria* that follow, is used as a decimal integer (optionally signed), where +*n* means more than *n*, -*n* means less than *n*, and *n* means exactly *n*. A day is defined as 24 hours.

-a *n* True if the file has been accessed in *n* days.

-m *n* True if the file has been modified in *n* days.

-c *n* True if the inode has been changed in *n* days.

-n *file* True for any file which has been modified more recently than the argument *file*.

Examples

To write a tape consisting of all files from file-system / modified in the last 48 hours:

finc -m -2 /dev/root /dev/rct

See Also

ff(M), frec(M), labelit(C) and cpio(C)

Name

frec - Recovers files from a backup tape.

Syntax

/etc/frec [-p *path*] [-f *reqfile*] *raw_tape*
inode_number:name...

Description

Frec recovers files from the specified *raw_tape* backup tape written by **volcopy**(M) or **finc**(M), given their *inode_numbers.* The data for each recovery request will be written into the file given by *name.*

Options

-p *path* Specifies a prefixing *path* (different from your current working directory). This will be prefixed to any *names* that are not fully qualified, i.e., that do not begin with / or ./. If any directories are missing in the paths of recovery *names*, they will be created.

-f *reqfile* Specifies a file that contains recovery requests. The format is *inode_number:name*, one per line.

Examples

To recover a file, inode_number 1216 when backed-up, into a file named *junk* in your current working directory, type:

frec /dev/rct 1216:junk

To recover files with inode_numbers 14156, 1232, and 3141 into files /usr/src/cmd/a, /usr/src/cmd/b and /usr/joe/a.c, enter:

frec -p /usr/src/cmd /dev/rct 14156:a 1232:b 3141:/usr/joe/a.c

See Also

ff(M), finc(M), labelit(M), and cpio(C)

Notes

While paving a path (i.e., creating the intermediate directories contained in a pathname), **frec** can only recover inode fields for those directories contained on the tape and requested for recovery.

Name

fsdb - File system debugger.

Syntax

/etc/fsdb *special* [-]

Description

Fsdb is used to patch up a damaged file system after a crash. It has conversions to translate block and inode numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an inode. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error-checking routines to verify inode and block addresses. These can be disabled if necessary by invoking **fsdb** with the optional - argument or by the use of the **O** symbol. (**Fsdb** reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by **fsdb** are:

#	absolute address
i	convert from inode number to inode address
b	convert to block address
d	directory slot offset
+,-	address arithmetic
q	quit

>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=-	decremental assignment
="	character string assignment
O	error checking toggle
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as inodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current inode. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or inode, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal inode number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for inode examination and refer to the current working inode:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 - 12)
at	access time
ct	creation time
mt	modification time
maj	major device number
min	minor device number

Examples

386i	Prints inode number 386 in an inode format. This now becomes the current working inode.
ln=4	Changes the link count for the working inode to 4.
ln=+1	Increments the link count by 1.
fc	Prints, in ASCII, block zero of the file associated with the working inode.
2i.fd	Prints the first 32 directory entries for the root inode of this file system.
d5i.fc	Changes the current inode to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.

512B.p0o	Prints the superblock of this file system in octal.
2i.a0b.d7=3	Changes the inode number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	Changes the name field in the directory slot to the given string. Quotes are optional when used with nm if the first character is alphabetic.
a2b.p0d	Prints the third block of the current inode as directory entries.

See Also

fsck(C), and dir(S), fs(S) in the *Reference (CP, S, F)*

Name

fsinfo - Reports information about a file system.

Syntax

fsinfo *options file-system*

Description

The **fsinfo** command displays information about the given *filesystem.* All the values returned by **fsinfo** are expressed in 512 byte blocks.

Options

-f Returns the free block count of the *file-system.*

-i Returns the total number of blocks of inodes in a *file-system.*

-l Returns the total number of free blocks in the *file-system.*

-s Performs a sanity check on the *file-system.* The return code will be 0 if the sanity check completes successfully. A positive number is returned on failure.

See Also

df(M)

Name

fsstat - Reports file system status.

Syntax

/etc/fsstat *special_file*

Description

Fsstat reports on the status of the file system on *special_file*. During startup, this command is used to determine if the file system needs checking before it is mounted. **Fsstat** succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

Diagnostics

The command has the following exit codes:

0 - the file system is not mounted and appears okay, (except for root where 0 means mounted and okay).
1 - the file system is not mounted and needs to be checked.
2 - the file system is mounted.
3 - the command failed.

Name

fstab - File system table.

Description

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

column 1	block special file name of file system
column 2	mount-point directory
column 3	"-r" if to be mounted read-only
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/hd1b /usr2
```

Files

/etc/fstab

See Also

mount(C), mountall(C)

Name

fstyp - Determines file system identifier.

Syntax

/etc/fstyp *special*

Description

Fstyp allows the user to determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by **mount**(S) and sometimes by **mount**(M) to mount file systems of different types.

The directory /etc/fstyp.d contains a program for each file system type to be checked; each of these programs applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; otherwise it prints error messages on standard error and exits with a non-zero return code. **Fstyp** runs the programs in /etc/fstyp.d in alphabetical order, passing *special* as an argument; if any program succeeds, its file-system type identifier is printed and **fstyp** exits immediately. If no program succeeds, **fstyp** prints "Unknown_fstyp" to indicate failure.

Notes

The use of heuristics implies that the result of **fstyp** is not guaranteed to be accurate.

See Also

mount(M), and mount(S), sysfs(S) in the *Reference (CP, S, F)*

Name

fuser - Identifies processes using a file or file structure.

Syntax

/etc/fuser [**-ku**] *file...* | *resource...* [-] [[**-ku**]
file... | *resource...*]

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:

c Current directory

p Parent of its current directory (only when the file is being used by the system)

r Root directory

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**(S)).

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.

Any user with permission to read /dev/kmem and /dev/mem can use **fuser.** Only the super-user can terminate another user's process.

Files

/unix	For system namelist
/dev/kmem	For system image
/dev/mem	Also for system image

See Also

mount(C), ps(C)
kill(S), signal(S) in the *Reference (CP, S, F)*

Name

getty - Sets terminal type, modes, speed, and line discipline.

Syntax

/etc/getty [**-h**] [**-t** *timeout*] *line* [*speed* [*type* [*linedisc*]]]
/etc/getty -c *file*

Description

Getty is a program that is invoked by **init**(M). It is the second process in the series, *(init-getty-login-shell)* that ultimately connects a user with the operating system. It can only be executed by the super-user; that is, a process with the user-ID of **root.** Initially **getty** prints the login message field for the entry it is using from /etc/gettydefs. **Getty** reads the user's login name and invokes the **login**(C) command with the user's name as argument. While reading the name, **getty** attempts to adapt the system to the speed and type of terminal being used. It does this by using the options and arguments specified.

Line is the name of a tty line in /dev to which **getty** is to attach itself. **Getty** uses this string as the name of a file in the /dev directory to open for reading and writing. Unless **getty** is invoked with the **-h** flag, **getty** will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **-t** flag plus *timeout* (in seconds), specifies that **getty** should exit if the open on the line succeeds and no one types anything in the specified number of seconds.

Speed, the optional second argument, is a label to a speed and tty definition in the file /etc/gettydefs. This definition tells **getty** at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by pressing Break/Del). The default *speed* is 300 baud.

Type, the optional third argument, is a character string describing to **getty** what type of terminal is connected to the line in question. **Getty** recognizes the following types:

none	default
ds40-1	Dataspeed40/1
tektronix,tek	Tektronix
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition.

Linedisc, the optional fourth argument, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, LDISC0.

When given no optional arguments, **getty** sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pressing Break/Del. This will cause **getty** to attempt the next *speed* in the series. The series that **getty** tries is determined by what it finds in /etc/gettydefs.

After the user's name has been typed in, it is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see **ioctl**(S)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, **login** is executed with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to **login**, which will place them in the environment (see **login**(C)).

A check option is provided. When **getty** is invoked with the **-c** option and *file*, it scans the file as if it were scanning /etc/gettydefs and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See **ioctl**(S) to interpret the values. Note that some values are added to the flags automatically.

Files

/etc/gettydefs
/etc/issue

See Also

ct(C), gettydefs(M), init(M), inittab(M), login(C), tty(M), and ioctl(S) in the *Reference (CP, S, F)*

Notes

While **getty** understands simple single character quoting conventions, it is not possible to quote certain special control characters used by **getty**. Thus, you cannot login via **getty** and type a #, @, /, !, _, backspace, ^U, ^D, or & as part of your login name or arguments. **Getty** uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.

Name

gettydefs - Speed and terminal settings used by getty.

Description

The **/etc/gettydefs** file contains information used by **getty**(M) 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 types a Break character, which indicates the current speed is not correct.

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**(M) 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**(S) 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**(M)). Normally only the speed flag is required in the *inital-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**(C).

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 newline), 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, and is the default entry. It is also used if **getty** cannot 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.

After making or modifying **/etc/gettydefs**, run it through **getty** with the check option to be sure there are no errors.

Files

/etc/gettydefs

See Also

getty(M), termio(M), login(C), uugetty(M)

Name

group - Format of the group file.

Description

The **/etc/group** file contains the following information:

- Group name
- Encrypted password (optional)
- 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 newline. If the password field is empty, then you are not prompted for a password, when using the **newgrp**(C) command.

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

See Also

passwd(M)

Name

haltsys - Closes out the file systems and halts the CPU.

Syntax

/etc/haltsys

Description

You must be the super-user to access this command.

The **haltsys** command immediately terminates the operating system and should only be used if a system problem prevents the running of **shutdown.** Do not run **haltsys** in multiuser mode and when other users are on the system. Since **haltsys** takes effect immediately, user processes should be killed beforehand (see **kill**(C)).

Related Commands

kill(C), ps(C), shutdown(M)

Name

infocmp - Compares or prints out terminfo descriptions.

Syntax

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**(M) entry with other terminfo entries, rewrite a **terminfo**(M) description to take advantage of the **use=** terminfo field, or print out a **terminfo**(M) description from the binary file (**term**(M)) 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**(M) 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**(M) 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**(M), but which are derivable from other **terminfo**(M) variables, will be output. Not all **terminfo**(M) 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**(M) 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**(M) format will not necessarily reproduce the original **terminfo**(M) 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**(M) 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**(C) 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 in which the entries are given. **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**(M) 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**(M) 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**(M) 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(C), curses(S), term(M), terminfo(M), captoinfo(M)

Note

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

Name

inir - Cleans the file system and executes init.

Syntax

/etc/inir

Description

Inir first checks that the console devices (/dev/console, /dev/syscon, /dev/systty) are correct, and if not removes and creates them. **Inir** will then fork a child process that reports the number of users licensed for this system and that cleans the file system by running **fsck**(C).

Inir is called as "c" or "d" to indicate whether the file system is clean or dirty. If **inir** is invoked as anything other than "c," it assumes the file system is dirty.

When the child process returns, **inir** will execute **init**(M).

Files

/dev/console
/dev/syscon
/dev/systty

See Also

init(M), fsck(C)

Name

init, telinit - Process control initialization.

Syntax

/etc/init [0123456SsQq]
/etc/telinit [0123456sSQqabc]

Description

Init is a general process spawner. Its primary role is to create processes from a script stored in the file /etc/inittab (see **inittab**(M)). This file usually has **init** spawn **getty**(M) processes on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a run-level at any given time. A run-level can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by **init** for each of these run-levels are defined in the inittab file.

Init can be in one of eight run-levels, **0-6,** and S or s. The run-level is changed by having a privileged user run /etc/telinit (which is linked to /etc/init). This user-spawned **init** sends appropriate signals to the original **init** spawned by the operating system when the system was booted, telling it which run-level to change to.

Init is invoked as the last step in the **boot**(M) procedure. The first thing it does is to look for /etc/inittab and see if there is an entry of the type *initdefault* (see **inittab**(M)). If there is, **init** uses the run-level specified in that entry as the initial run-level to enter. If this entry is not in inittab or inittab is not found, **init** requests that the user enter a run-level from the virtual system console, /dev/syscon. If an S (s) is entered, **init** goes into the SINGLE USER level. This is the only run-level that doesn't require the existence of a properly formatted inittab file.

If /etc/inittab doesn't exist, then by default the only legal run-level that **init** can enter is the SINGLE USER level. In the SINGLE USER level, the virtual console terminal /dev/syscon is opened for reading and writing and the command /bin/su is invoked immediately. To exit from the SINGLE USER run-level one of two options can be elected. First, if the shell is terminated (via an end-of-file), **init** will reprompt for a new run-level. Second, the **init** or **telinit** command can signal **init** and force it to change the run-level of the system.

When attempting to boot the system, **init** may fail to prompt for a new run-level because the device /dev/syscon is linked to a device other than the physical system terminal (/dev/systty). If this occurs, **init** can be forced to relink /dev/syscon by typing a delete on the system console that is located with the processor.

When **init** prompts for the new run-level, you may enter only one of the digits **0** through **6** or the letters **S** or **s**. If S is entered, **init** operates as previously described in SINGLE USER mode with the additional result that /dev/syscon is linked to your terminal line, thus making it the virtual system console. A message is generated on the physical console, /dev/systty, saying where the virtual terminal has been relocated.

When **init** comes up initially and whenever it switches out of SINGLE USER state to normal run states, it sets the **ioctl**(S) states of the virtual console, /dev/syscon, to those modes saved in the file /etc/ioctl.syscon. This file is written by **init** whenever SINGLE USER mode is entered. If this file does not exist when **init** wants to read it, a warning is printed and default settings are assumed.

If a **0** through **6** is entered, **init** enters the corresponding run-level. Any other input will be rejected and the user will be reprompted. If this is the first time **init** has entered a run-level other than SINGLE USER, **init** first scans inittab for special entries of the type *boot* and *bootwait*. These entries are performed, providing the run-level entered matches that of the entry before any normal processing of inittab. In this way, any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The inittab file is scanned to find all entries that are to be processed for that run-level.

Run-level **2** is usually defined by the system administrator to contain all of the terminal processes and daemons that are spawned in the multiuser environment. Run-level **3** is defined to start up remote file sharing processes and daemons as well as mount and advertise remote resources. So, run-level **3** extends multiuser mode and is known as the Remote File Sharing state.

In a multiuser environment, the inittab file is usually set up so that **init** will create a process for each terminal on the system.

For terminal processes, the shell will ultimately terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When **init** receives a child death signal, telling it that a process it spawned has died, it records the fact and the reason it died in /etc/utmp and /etc/wtmp if it exists (see **who**(C)). A history of the processes spawned is kept in /etc/wtmp if such a file exists.

To spawn each process in the inittab file, **init** reads each entry and for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the inittab file, **init** waits for one of its descendant processes to die, a powerfail signal, or until it is signaled by **init** or **telinit** to change the system's run-level. When one of the above three conditions occurs, **init** re-examines the inittab file. New entries can be added to the inittab file at any time. To provide for an instantaneous response the **telinit Q** or **telinit q** command can wake **init** to reexamine the inittab file.

If **init** receives a powerfail signal (SIGWPR) and is not in SINGLE USER mode, it scans inittab for special powerfail entries. These entries are invoked (if the run-levels permit) before any further processing takes place. In this way **init** can perform various cleanup and recording functions whenever the operating system experiences a power failure. Note that in the single-user state, only powerfail and powerwait entries are executed.

When **init** is requested to change run-levels (via **telinit**), it sends the warning signal (SIGTERM) to all processes that are undefined in the target run-level. **Init** waits 20 seconds before forcibly terminating these processes via the kill signal (SIGKILL).

Telinit

Telinit, which is linked to /etc/init, is used to direct the actions of **init**(S). It takes a one-character argument and signals **init** via the kill system call to perform the appropriate action. You must be the super-user to run **telinit.**

The following arguments serve as directives to **init.**

0-6 Tells **init** to place the system in one of the run-levels **0-6.** Run level 0 is used for shutdown; 1 is single user mode; and 2 is multiuser mode. To switch between single and multiuser modes, use the scripts /etc/singleuser and /etc/multiuser.

a,b,c Tells **init** to process only those /etc/inittab file entries having the **a, b,** or **c** run-level set (see **inittab**(M)).

q,Q Tells **init** to re-examine the /etc/inittab file.

s,S Tells **init** to enter the single user environment. The virtual system teletype, /dev/syscon, is changed to the terminal from which the command was executed.

Init.d

The /etc/init.d directory contains initialization and termination scripts for changing **init** states. These scripts are linked with appropriate files in the rc?.d directories.

File names in rc?.d directories are of the form [S|K]*nn*<**init.d** *filename*> where S means start this job, K means kill this job, and *nn* is the relative sequence number for killing or starting the job. When entering a state (init 0, 2, 3, etc.), the rc[0-6] script executes those scripts in /etc/rc[0-6].d that are prefixed with K followed by those scripts prefixed with S.

For example, when changing to **init** state **2** (default multi-user mode), /etc/rc2 is initiated by the **init** process. The following steps are performed by /etc/rc2:

- In the directory /etc/rc2.d are files used to stop processes that should not be running in state 2. The file names are prefixed with K. Each K file in the directory is executed (by /etc/rc2) in alpha-numeric order when the system enters **init** state 2 (see the following example).

- The rc2.d directory also contains files used to start processes that should be running in state 2. As in the step above, each S file is executed.

  Example:

  The file /etc/netdaemon contains a script that initiates networking daemons when given the argument **start** and terminates the daemons if given the argument **stop**. It is linked to /etc/rc2.d/S68netdaemon, and to /etc/rc0.d/K67netdaemon.

  This script is executed by **/etc/rc2.d/S68netdaemon start** when **init** state 2 is entered and by **/etc/rc0.d/S67netdaemon stop** when shutting the system down.

Files

/etc/inittab
/etc/init.d
/etc/rc0
/etc/rc0.d
/etc/rc2
/etc/rc2.d
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/syscon

See Also

brc(M), getty(M), inittab(M), login(C), rc0(M), rc2(M), sh(C), utmp(M), who(C)

Diagnostics

If **init** finds that it is continuously respawning an entry from /etc/inittab more than 10 times in 2 minutes, it assumes there is an error in the command string, generates an error message on the system console, and refuses to respawn this entry until either 1 minute has elapsed or it receives a signal from a user **init** (**telinit**). This prevents **init** from eating up system resources when someone makes a typographical error in the inittab file or a program is removed that is referenced in the inittab.

Notes

Telinit can be run only by someone who is the super-user or a member of group sys. Attempting to relink /dev/console with /dev/contty by typing Del on the system console does not work.

Name

inittab - Script for the init process.

Description

The **/etc/inittab** file supplies the script to **init**(M)'s role as a general process dispatcher. The process that constitutes the majority of **init's** process dispatching activities is the line process **getty**(M) 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**(C) convention for comments. Comments for lines that spawn **getty**(M) are displayed by the **who**(C) command. They typically 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 that 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 perio(

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 are three other values, **a**, **b**, or **c**, which can appear in the *rstate* field, even though they are not true run-levels. Entries that have these characters in the *rstate* field are processed only when the **telinit** (see **init**(M)) 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. Furthermore, a process started by **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 only at **init**'s boot-time read of the **inittab** file. **Init** is to start the process, wait for its termination, and, when it dies, not restart the process.

powerfail Execute the process associated with this entry only when **init** receives the power fail signal (SIGPWR, see **signal**(S)), which normally occurs wher a UPS detects a power failure.

powerwait Execute the process associated with this entry only when **init** receives the 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 2(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. Also, the **initdefault** entry cannot specify that **init** start in the SINGLE USER state. 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.

restart Entries of this type are executed on a warm restart of the system after a power failure.

sysinit Entries of this type are executed before **init** tries to access the console. 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(C) 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

getty(M), init(M), sh(C), who(C)

Name

inode - Format of an inode.

Syntax

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

Description

An inode for a plain file or directory in a file system has the structure defined by <sys/ino.h>. For the meaning of the defined types *off_t* and *time_t*, see **types**(F).

Files

/usr/include/sys/ino.h

See Also

filesystem(M) and stat(S), types(F) in the *Reference (CP, S, F)*

Name

install - Installs commands.

Syntax

/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**(C)) 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 super-user.

-u *user* The owner of the new file is set to *user*. Only available to the super-user.

-g *group* The group id of the new file is set to *group*. Only available to the super-user.

-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(C)

Name

keyboard - Series 500 system console keyboard.

Description

The system console (and user's terminal) is composed of two separate pieces: the keyboard and the display (see **display**(M)). Because of their complexity they are discussed in separate manual entries.

The actual code sequence delivered to the terminal input routine (see **termio**(M)) is defined by a set of internal tables in the driver. These tables can be modified by software (see **ioctl** calls below). In addition, the driver can be instructed not to do translations, delivering the keyboard up/down scan codes directly.

There are four translation tables: normal keys, shifted keys, alt keys, and shifted alt keys. Each table contains 128 16-bit entries, with an entry being made up of flags in the high-order 8 bits and the character code in the low-order 8 bits. The values that can be set in the flag byte, as defined in **<sys/kd.h>**, are as follows:

```
/* Flag bits */
#define  NUMLCK    0x8000   /* key is affected by num lock */
#define  CAPLCK    0x4000   /* key is affected by caps lock */
#define  CTLKEY    0x2000   /* key is affected by control key */

/* Key types */
#define  NORMKEY   0x0000   /* key is a normal key */
#define  SHIFTKEY  0x0100   /* key is a shift key */
#define  BREAKKEY  0x0200   /* key is a break key */
#define  SS2PFX    0x0300   /* prefix key with <ESC> N */
#define  SS3PFX    0x0400   /* prefix key with <ESC> O */
#define  CSIPFX    0x0500   /* prefix key with <ESC> [ */
#define  NOKEY     0x0f00   /* key sends nothing */
```

The tables are indexed by the keyboard scan code received. The table that is used is determined by the state of the following special keys:

ALT This key essentially chooses an alternate keyboard. If it is not depressed, the normal and shifted tables are used; if it is depressed, the alt and shifted alt tables are used.

SHIFT Depending on the ALT key, this key shifts into either the shifted table or the shifted alt table. The default shifted table is set up such that SHIFT will generate the ASCII uppercase characters.

The character code found in the table may be further modified by the following keys:

CTRL Produces the appropriate ASCII control character if the CTLKEY bit is set in the flag byte. The control character is produced by masking off all but the low-order 5 bits of the character code in the table. If the CTLKEY bit is not set, the normal character (the code in the table) is generated. In the default tables, the CTRL key only modifies keys in the normal and shifted tables; it has no effect in the alt or shifted alt tables.

CAPS LOCK
This is a toggle; it controls whether keys that have the CAPLCK bit set in their flag byte go to the normal or shifted table. If the CAPLCK bit is not set, the normal character is generated regardless of the state of the CAPS LOCK. The SHIFT key inverts whatever state is indicated by the CAPS LOCK. Thus, if CAPS LOCK is off, SHIFT produces uppercase characters; if CAPS LOCK is on, SHIFT produces lowercase characters. In the default tables, the only keys affected by CAPS LOCK are the alphabetic keys.

NUM LOCK
This is a toggle; it controls whether keys that have the NUMLCK bit set in their flag byte go to the normal or shifted table. If the NUMLCK bit is not set, the normal character is generated regardless of the state of the NUM LOCK. The SHIFT key inverts whatever state is indicated by the NUM LOCK. In the default tables, the only keys affected by NUM LOCK are the

keypad keys. Note that CAPS LOCK and NUM LOCK do exactly the same thing; the only difference is the set of keys affected.

SCROLL LOCK
: This key is marked as a BREAKKEY in its flag byte in both the shifted and shifted alt tables. This causes it to send BREAK to the terminal handler.

The remaining values for the key type are discussed below:

SHIFTKEY
: This is used to mark the left and right SHIFT keys, the CTRL key, the ALT key, the CAPS LOCK, and the NUM LOCK in the translation tables. User programs will normally not be concerned with this flag.

SS2PFX, SS3PFX, CSIPFX
: These are used to generate codes for the function keys and for the ALT keys. If one of these flags is specified in the translation table, the driver will prefix the character code in the table with <ESC>N, <ESC>O, or <ESC>[respectively, where <ESC> represents the ASCII escape character (1b hex).

NOKEY
: This is used to mark entries that should not generate any character code. Keystroke combinations that index table entries marked with this flag generate nothing.

The following tables describe the codes generated by the default tables for all the keys. Keycodes are the values delivered at the keyboard interface when the corresponding key is struck (the down scan code). Note that when the key is released, the same code is delivered, but with the high-order bit set. Thus, codes 01-7f are down codes, and 81-ff are up codes. The generated codes are the codes delivered to the terminal driver after translation. All numbers are in hexadecimal.

Shifting Keys

Key	Code	Function
Ctrl	1d	CTRL
Left Shift	2a	SHIFT
Right Shift	36	SHIFT
Alt	38	ALT
Caps Lock	3a	CAPS LOCK
Num Lock	45	NUM LOCK

Special Keys

Keyboard		Generated Codes				SHIFT
Key	Code	Normal	SHIFT	CTRL	ALT	ALT
BACKSPACE	0e	08 bs	08 bs	08 bs	08 bs	08 bs
TAB	0f	09 ht	1d gs	09 ht	09 ht	1d gs
RETURN	1c	0d cr	0d cr	0d cr	0d cr	0d cr
SPACE	39	20 sp	20 sp	00 nul	20 sp	20 sp
ESC	01	1b esc	1b esc	1b esc	1b esc	1b esc

Alphabetic Keys

Keyboard Key	Code	Generated Codes Normal	SHIFT	CTRL	ALT	SHIFT ALT
a	1e	61 a	41 A	01 soh	1b4e61	1b4e41
b	30	62 b	42 B	02 stx	1b4e62	1b4e42
c	2e	63 c	43 C	03 etx	1b4e63	1b4e43
d	20	64 d	44 D	04 eot	1b4e64	1b4e44
e	12	65 e	45 E	05 enq	1b4e65	1b4e45
f	21	66 f	46 F	06 ack	1b4e66	1b4e46
g	22	67 g	47 G	07 bel	1b4e67	1b4e47
h	23	68 h	48 H	08 bs	1b4e68	1b4e48
i	17	69 i	49 I	09 ht	1b4e69	1b4e49
j	24	6a j	4a J	0a lf	1b4e6a	1b4e4a
k	25	6b k	4b K	0b vt	1b4e6b	1b4e4b
l	26	6c l	4c L	0c ff	1b4e6c	1b4e4c
m	32	6d m	4d M	0d cr	1b4e6d	1b4e4d
n	31	6e n	4e N	0e so	1b4e6e	1b4e4e
o	18	6f o	4f O	0f si	1b4e6f	1b4e4f
p	19	70 p	50 P	10 dle	1b4e70	1b4e50
q	10	71 q	51 Q	11 dc1	1b4e71	1b4e51
r	13	72 r	52 R	12 dc2	1b4e72	1b4e52
s	1f	73 s	53 S	13 dc3	1b4e73	1b4e53
t	14	74 t	54 T	14 dc4	1b4e74	1b4e54
u	16	75 u	55 U	15 nak	1b4e75	1b4e55
v	2f	76 v	56 V	16 syn	1b4e76	1b4e56
w	11	77 w	57 W	17 etb	1b4e77	1b4e57
x	2d	78 x	58 X	18 can	1b4e78	1b4e58
y	15	79 y	59 Y	19 em	1b4e79	1b4e59
z	2c	7a z	5a Z	1a sub	1b4e7a	1b4e5a

Numeric and Punctuation Keys

Keyboard		Generated Codes				SHIFT
Key	Code	Normal	SHIFT	CTRL	ALT	ALT
1	02	31 1	21 !	31 1	1b4e31	1b4e21
2	03	32 2	40 @	00 nul	1b4e32	1b4e40
3	04	33 3	23 #	33 3	1b4e33	1b4e23
4	05	34 4	24 $	34 4	1b4e34	1b4e24
5	06	35 5	25 %	35 5	1b4e35	1b4e25
6	07	36 6	5e ^	1e rs	1b4e36	1b4e5e
7	08	37 7	26 &	37 7	1b4e37	1b4e26
8	09	38 8	2a *	38 8	1b4e38	1b4e2a
9	0a	39 9	28 (	39 9	1b4e39	1b4e28
0	0b	30 0	29)	30 0	1b4e30	1b4e29
-	0c	2d -	5f _	1f us	1b4e2d	1b4e5f
=	0d	3d =	2b +	3d =	1b4e3d	1b4e2b
[	1a	5b [	7b {	1b esc	1b4e5b	1b4e7b
]	1b	5d]	7d }	1d gs	1b4e5d	1b4e7d
;	27	3b ;	3a :	3b ;	1b4e3b	1b4e3a
'	28	27 '	22 "	27 '	1b4e27	1b4e22
`	29	60 `	7e ~	1e rs	1b4e60	1b4e7e
\	2b	5c \	7c |	1c fs	1b4e5c	1b4e7c
,	33	2c ,	3c <	2c ,	1b4e2c	1b4e3c
.	34	2e .	3e >	2e .	1b4e2e	1b4e3e
/	35	2f /	3f ?	1f us	1b4e2f	1b4e3f

Keypad Keys

Keyboard Key	Code	Generated Codes Normal	SHIFT	CTRL	ALT	SHIFT ALT
*	37	2a *	2a *	2a *	1b4e2a	1b4e2a
scroll lock	46	1b5b4d	00 break	1b5b4d	1b5b4d	00 break
home	47	1b5b48	37 7	1b5b48	1b5b48	1b4e37
up arrow	48	1b5b41	38 8	1b5b41	1b5b41	1b4e38
page up	49	1b5b49	39 9	1b5b49	1b5b49	1b4e39
minus	4a	2d -	2d -	2d -	1b4e2d	1b4e2d
left arrow	4b	1b5b44	34 4	1b5b44	1b5b44	1b4e34
5	4c	1b5b45	35 5	1b5b45	1b5b45	1b4e35
right arrow	4d	1b5b43	36 6	1b5b43	1b5b43	1b4e36
plus	4e	2b +	2b +	2b +	1b4e2b	1b4e2b
end	4f	1b5b46	31 1	1b5b46	1b5b46	1b4e31
down arrow	50	1b5b42	32 2	1b5b42	1b5b42	1b4e32
page down	51	1b5b47	33 3	1b5b47	1b5b47	1b4e33
insert	52	1b5b4c	30 0	1b5b4c	1b5b4c	1b4e30
del	53	7f	2e .	7f	7f	1b4e2e
sys req	54	00	00	00	00	1b5b35

Function Keys

Keyboard Key	Code	Generated Codes Normal	SHIFT	CTRL	ALT	SHIFT ALT
F1	3b	1b5b4d	1b5b59	1b5b6b	1b4e4d	1b4e59
F2	3c	1b5b4e	1b5b5a	1b5b6c	1b4e4e	1b4e5a
F3	3d	1b5b4f	1b5b61	1b5b6d	1b4e4f	1b4e61
F4	3e	1b5b50	1b5b62	1b5b6e	1b4e50	1b4e62
F5	3f	1b5b51	1b5b63	1b5b6f	1b4e51	1b4e63
F6	40	1b5b52	1b5b64	1b5b70	1b4e52	1b4e64
F7	41	1b5b53	1b5b65	1b5b71	1b4e53	1b4e65
F8	42	1b5b54	1b5b66	1b5b72	1b4e54	1b4e66
F9	43	1b5b55	1b5b67	1b5b73	1b4e55	1b4e67
F10	44	1b5b56	1b5b68	1b5b74	1b4e56	1b4e68
F11	57	1b5b57	1b5b69	1b5b75	1b4e57	1b4e69
F12	58	1b5b58	1b5b6a	1b5b76	1b4e58	1b4e6a

Foreign Character Set Support

The keyboard driver supports input and output mapping for 9 different foreign language keyboards and character sets. The foreign keyboards supported are:

Language	Language Type defined in <sys/kd.h>	Function key
English	US_ENGLISH	F1
U.K./British	UK_ENGLISH	F2
French	FRENCH	F3
German	GERMAN	F4
Spanish	SPANISH	F5
Swedish	SWEDISH	F6
Norwegian	NORWEGIAN	F7
Danish	DANISH	F8
Italian	ITALIAN	F9
7/8 Bit Mode Toggle		F10

The *Series 500 Owner's Guide* describes the keyboard layouts and ASCII character sets for each keyboard.

There are 3 ways to change from one language mapping to another. They are:

1. Ctrl-Alt-Sysreq key combination

2. /etc/language file

3. KDSETLANG and KDGETLANG ioctl commands

At any time when the operating system is running, the user can simultaneously type the keys Ctrl, Alt and Sysreq followed by a Function key, to change keyboard mapping. The function keys for each language are listed in the table above. To change to U.K. English, for example, the user would simultaneously press Ctrl-Alt-Sysreq then type the F2 key. The current language will stay in effect until it is changed via a key sequence, an **ioctl** call, or until the system is rebooted. Note, on some keyboards, the SYSREQ key is labeled as PRINT SCREEN.

The F10 key is used to toggle between 7-bit and 8-bit versions of the language type currently in use. When the system is first booted, 7-bit character sets are used by default. The F10 key does not change the language type.

The system can be configured to boot with a particular language other than English as the default. This is done via the /etc/language file. If this file is present and contains a string matching one of the valid language types from the table above, then that language is mapped in immediately. If the file is not present or does not contain a valid language type, then the default language (US_ENGLISH) is used.

For an explanation of KDSETLANG and KDGETLANG, see the **ioctl** section that follows.

Ioctl Calls

KDGKBTYPE

This call is used to get the current keyboard type. It places one of the following numbers, as defined in **<sys/kd.h>**, at the unsigned char pointed to by the **ioctl** argument:

```
#define  KB_84     1       /* 84-key keyboard */
#define  KB_101    2       /* 101/102-key keyboard */
#define  KB_OTHER  3       /* other type of keyboard */
```

KDGKBMODE

This call is used to get the current keyboard mode. It returns one of the following numbers, as defined in **<sys/kd.h>**:

```
#define  K_RAW    0x00     /* send up/down scan codes */
#define  K_XLATE  0x01     /* translate to ascii */
```

KDSKBMODE

This call is used to set the keyboard mode. The argument to the **ioctl** is either K_RAW or K_XLATE. By using raw mode, the program can see the raw up/down can codes from the keyboard. In translate mode, the translation tables are used to generate the appropriate character code.

KDGKBENT

This call is used to read one of entries in the translation tables. The argument to the **ioctl** is the address of one of the following structures, defined in <**sys/kd.h**>, with the first two fields filled in:

```
struct kbentry {
        unchar  kb_table;       /* which table to use */
        unchar  kb_index;       /* which entry in table */
        ushort  kb_value;       /* value to get/set */
}
/* Table selectors */
#define  K_NORMTAB          0x00     /* normal table */
#define  K_SHIFTTAB         0x01     /* shifted table */
#define  K_ALTTAB           0x02     /* alt table */
#define  K_ALTSHIFTTAB      0x03     /* shifted alt table */
```

The **ioctl** will get the indicated entry from the indicated table and return it in the third field.

KDSKBENT

This call is used to set an entry in one of the translation tables. It uses the same structure as the KDGKBENT **ioctl**, but with the third field filled in with the value that should be placed in the translation table. This can be used to partially or completely remap the keyboard.

KDGETLED

Used to return an unsigned character which may have any or none of the following flags (defined in <**sys/kd.h**>) set:

LED_CAP	The CAP LOCK key is set
LED_SCR	The SCROLL LOCK key is set
LED_NUM	The NUM LOCK key is set

KDSETLED

Used to set the CAP LOCK, SCROLL LOCK, or NUM LOCK keys. The argument should contain one or all of the valid flags shown under KDGETLED.

KDMKTONE

Used to ring the bell at given frequency and for a given duration. The argument is a long integer having the following format:

lower 16 bits	Contains desired frequency
upper 16 bits	Time to ring in milliseconds

The frequency used for the normal system bell character is 1331 (decimal).

KDGETLANG

Used to return the current language in use on the console terminal. The argument returned is an integer which contains one of the valid language types (defined in <sys/kd.h>) listed previously under Foreign Character Set Support.

KDSETLANG

Used to change the language in use on the console terminal. Uses an integer argument which should be set to one of the valid language types (defined in <sys/kd.h>). The change takes effect immediately.

If the argument is 7-bit or 8-bit (defined in <sys/kd.h>), the terminal switches to a 7 or 8-bit version of the language currently in use.

Files

/dev/console

See Also

ioctl(S), display(M), termio(M), vt(M)

Name

killall - Kills all active processes.

Syntax

/etc/killall [*signal*]

Description

Killall terminates all active processes not directly related to the shutdown procedure. **Killall** is used by /etc/shutdown, and can only be run by the super-user.

Killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* (see **kill**(C)). The default signal is 9.

Files

/etc/shutdown

See Also

kill(C), ps(C), shutdown(M)

Name

layout - Manages hard disk partitions.

Syntax

/etc/layout **-c** | **-p** *driveid*
/etc/layout **-la|b|c|d|e|f|g|h|.spares|.restart** *driveid*
/etc/layout [**-f**] [**-r** m|**c**] [**-d**] | [**-e**] | [**-m**] *driveid* *ldevice*

Description

The **layout** command is used to create, alter, and inspect the partition map on a hard disk unit. The hard disk partition map is a fixed-size table of 16 entries, each of which describes the position and size of a logical device on a hard disk. This information, along with the bad-sector map (/dev/hd?.secmap), is used by the file processor subsystem.

CAUTION

Only an experienced system administrator should use this command. Running layout could make all of your files inaccessible.

Several of these devices are informational and have fixed locations (track 0, cylinder 0). Other logical devices are made available for definition by the user.

The logical devices are:

Offset	Device	Use
0	hd0	unmapped drive
1	hd0a	user defined - default swap area on drive 0
2	hd0b	user defined - root file system o drive 0
3	hd0c	user defined
4	hd0d	user defined
5	hd0e	user defined
6	hd0f	user defined
7	hd0g	user defined
8	hd0h	user defined
9	hd0.spares	alternates for unmapped bad sectors
10	hd0.drinfo	drive configuration information (recorded during manufacturing)
11	hd0.badlist	list of bad sectors (recorded during manufacturing)
12	hd0.boot	boot program
13	hd0.restart	restart partition
14	hd0.layout	layout information
15	hd0.secmap	sector sparing map

The second hard disk (hdl) starts at 16 and the third hard disk (hd2) starts at 48.

Layout for the Series 500
UNIX - **hd0**

For the Series 500, if you partition the disk with more than one partition, the *driveid* is a two-digit number. The first digit is the physical disk number (0 or 1). The second digit is the partition number (0, 1, 2, or 3). For example, if you partition your hard disk for both UNIX and DOS, the partitions are hd0 and hd01, respectively, The following lists show the minor device number for the partitions and logical devices on the Series 500 hard disks. The major device number for all of these logical devices is 0. When UNIX is installed, minor devices 0 - 15 are automatically made. If a second hard disk is installed, only minor devices 16 - 31 are made for it.

If you want more than one UNIX partition, run **fdisk**(C) to split up the hard disk. Then run **mknod**(C) to create the logical devices for it. For example, run the following 16 commands to make the devices for the second partition (i.e., partition 1) on drive 0:

```
/etc/mknod /dev/hd01 c 0 32
/etc/mknod /dev/hd01a c 0 33
/etc/mknod /dev/hd01b c 0 34
/etc/mknod /dev/hd01c c 0 35
/etc/mknod /dev/hd01d c 0 36
/etc/mknod /dev/hd01e c 0 37
/etc/mknod /dev/hd01f c 0 38
/etc/mknod /dev/hd01.fsck c 0 39
/etc/mknod /dev/hd01h c 0 40
/etc/mknod /dev/hd01.spares c 0 41
/etc/mknod /dev/hd01.drinfo c 0 42
/etc/mknod /dev/hd01.badlist c 0 43
/etc/mknod /dev/hd01.boot c 0 44
/etc/mknod /dev/hd01.restart c 0 45
/etc/mknod /dev/hd01.layout c 0 46
/etc/mknod /dev/hd01.secmap c 0 47
```

Minor device numbers (offset) for the partitions and logical devices are listed in the following pages.

Drive 0, Partition 0

Offset	Device
0	hd0 (or hd00)
1	hd0a (or hd00a)
2	hd0b (or hd00b)
3	hd0c (or hd00c)
4	hd0d (or hd00d)
5	hd0e (or hd00e)
6	hd0f (or hd00f)
7	hd0.fsck (or hd00.fsck)
8	hd0h (or hd00h)
9	hd0.spares (or hd00.spares)
10	hd0.drinfo (or hd00.drinfo)
11	hd0.badlist (or hd00.badlist)
12	hd0.boot (or hd00.boot)
13	hd0.restart (or hd00.restart)
14	hd0.layout (or hd00.layout)
15	hd0.secmap (or hd00.secmap)

Drive 1, Partition 0

Offset	Device
16	hd1 (or hd10)
17	hd1a (or hd10a)
18	hd1b (or hd10b)
19	hd1c (or hd10c)
20	hd1d (or hd10d)
21	hd1e (or hd10e)
22	hd1f (or hd10f)
23	hd1.fsck (or hd10.fsck)
24	hd1h (or hd10h)
25	hd1.spares (or hd10.spares)

Drive 1, Partition 0 (Cont.)

Offset	Device
26	hd1.drinfo (or hd10.drinfo)
27	hd1.badlist (or hd10.badlist)
28	hd1.boot (or hd10.boot)
29	hd1.restart (or hd10.restart)
30	hd1.layout (or hd10.layout)
31	hd1.secmap (or hd10.secmap)

Drive 0, Partition 1

Offset	Device
32	hd01
33	hd01a
34	hd01b
35	hd01c
36	hd01d
37	hd01e
38	hd01f
39	hd01.fsck
40	hd01h
41	hd01.spares
42	hd01.drinfo
43	hd01.badlist
44	hd01.boot
45	hd01.restart
46	hd01.layout
47	hd01.secmap

Drive 1, Partition 1

Offset	Device
48	hd11
49	hd11a
50	hd11b
51	hd11c
52	hd11d
53	hd11e
54	hd11f
55	hd11.fsck
56	hd11h
57	hd11.spares
58	hd11.drinfo
59	hd11.badlist
60	hd11.boot
61	hd11.restart
62	hd11.layout
63	hd11.secmap

Drive 0, Partition 2

Offset	Device
64	hd02
	.
	.
	.
79	hd02.secmap

Drive 1, Partition 2

Offset	Device
80	hd12
	.
	.
	.
95	hd12.secmap

Drive 0, Partition 3

Offset	Device
96	hd03
	.
	.
	.
111	hd03.secmap

Drive 1, Partition 3

Offset	Device
112	hd13
	.
	.
	.
127	hd13.secmap

There are two additional devices that allow access to the entire hard disk:

128	hd0.entire	All of Drive 0
144	hd1.entire	All of Drive 1

Partition Map Creation

The **layout** command determines the size and positions of userdefinable areas from an ASCII format layout description file. Default layout descriptions are supplied, and may be altered by a knowledgeable user during the hard disk creation process. The install script and the **add.hd**(C) script are used to configure the main hard drive, and an additional drive, respectively.

On some machines, the optional Uninterruptible Power Supply (UPS) is available (for example, the Altos Series 2000). In this case, the install script asks you if a restart partition is desired, and if it is, whether it is to be made the current size of main memory, or the maximum possible memory size. This partition is used by the autorestart mechanism and may only be installed on the main drive. See **shutype**(M) for further details.

Next you are asked whether the default layout is acceptable for this disk. Select the default layout by determining the formatted size of the drive and consulting the /etc/layouts/driveclass file, which contains the names of default layout configuration files for different drive sizes. These files are found in the directory, /etc/layouts/defaults.

If the default layout is not acceptable, as in the case of a system that requires a larger-than-normal swap area, a dialogue is entered with the user (see the "Example" section that follows). As a result of this dialogue, a new layout file is created in the directory, /etc/layouts. The format of a layout description file is a collection of newline-terminated lines of the form:

name_of_partition size_of_partition

The first field is the name of the partition, the second field is the size of the partition in 512-byte blocks. The partition name must be a lowercase character in the range **a** through **h**, or the reserved words **.restart** and **.spares**. The size field is a decimal number. The partition description lines are not required to be in any specific order. The /etc/layouts/config file contains a mapping between the names of various user-configurable partitions and the minor device to which they apply. A sample layout file follows.

```
a         12320
b         6720
c         10080
d         1222280
.spares   152
.restart  4096
```

Any lines in the layout file with # in column 1 are considered comments and are ignored.

The **layout** command uses the following rules for map creation.

Each partition is allocated in the order it is specified in the layout description file. Space is allocated starting from track 2 of cylinder 0. Unlike previous versions of *layout*, partitions are made exactly the size cited in the description file. Likewise, the size of the last partition will not be automatically adjusted to make room for the space required for the maximum number of bad sectors on a drive. This number is calculated at a track per megabyte of unformatted disk. An advisory message will NOT be produced if the last partition spills over in the bad sector reserved area. The command line options for the partition creation invocation of **layout** are:

> **/etc/layout** [**-f**] [**-r m** | **c**] [**-d**] | [**-e**] | [**-m**]
> *driveid ldevice*

The value for *driveid* is a single character that selects the drive in question. The main drive's driveid is 0. The value for *ldevice* is usually the raw layout device for the specified drive. In the case of the main drive, this value is /dev/rhd0.layout.

-f This flag indicates that you want to alter a layout. A dialogue will begin and a new layout description file will be created with the values you specify.

-r This flag indicates that a restart partition is needed. You may choose between a restart partition sized the same as the maximum size of memory (m), or the current size of memory (c).

-d This flag indicates that the default layout description file for this size of disk should be used for all further operations.

-e This flag indicates that an altered layout description file for this size of disk should be used for all further operations.

-m This flag indicates that the partition map already installed on the disk should be used for all further operations.

Layout Viewing

The **-p** option prints (on standard output) a representation of the layout information for a particular drive. This representation consists of the name of the logical device, starting block number, and starting block size in 1/2K blocks. The numbers are in decimal. The following is an example taken from an 80 Mbyte hard disk:

/etc/layout -p 0

produces:

```
/dev/hd0 0 131072
/dev/hd0a 128 12416
/dev/hd0b 12544 6784
/dev/hd0c 19328 10112
/dev/hd0d 29440 100352
/dev/hd0e 0 0
/dev/hd0f 0 0
/dev/hd0g 0 0
/dev/hd0h 129792 416
/dev/hd0.spares 16 112
/dev/hd0.restart 0 0
```

The **-l** option with partition selector is used to supply **mkfs**(C) with the size to make the corresponding file system. For example,

/etc/layout -ld 0

produces:

```
100352
```

and is best used in the following context:

DSIZE = `/etc/layout -ld 0`
/etc/mkfs /dev/hd0d `expr $DSIZE /2` 4 128

Besides **a** through **h**, the **-l** option also takes **.restart** and **.spares** as acceptable arguments.

The **-c** option reads the /dev/hd?.drinfo file and prints the decimal values for size of drive in megabytes, number of cylinders, number of heads, number of sectors per track, numbers of sectors per cylinder, type of drive, and recommended interleave if the drive is a SCSI. The following is the result from an 80 Mbyte ST506-type hard disk:

```
80
1024
8
16
128
ST506
0
```

Other types of drives are SCSI and ESDI. The **-c** option is intended primarily for the benefit of shell scripts used to configure hard disk drives.

Example

For example, to add swap space to an additional drive, type **layout -f 2 /dev/rhd2.layout.** The following menu will be displayed:

```
1:a          #    (extra swap area)
2:b          #    (file system)
3:c          0    (not used)
4:d          0    (not used)
5:e          0    (not used)
6:f          0    (not used)
7:G          0    (not used)
8:h          0    (not used)
9:.spares    #    (spares for grown bad sectors)
10:
11:
12:
13:.restart   #    (restart memory image)
14:.extras    #    (currently unassigned)

command  q)uit  m)ove  d)isplay
```

To increase the swap area size (move blocks to the main swap area), type **m** (for move) and press Retn. A message on the screen prompts:

```
move blocks from partition #:
```

Type 14 (the partition number of currently unassigned blocks). You are asked:

```
move blocks to partition #:
```

Type 1 (for the main swap area). When prompted for the number of blocks, type the number you want to move from partition 14 to 1. Then type **d** to display the new block assignments. Finally, type **q** to quit.

Files

/etc/layouts/config	Device map for configurable partitions
/etc/layouts/defaults/*	Default layout descriptions
/etc/layouts/driveclass	Drive classes file
/dev/hd?.secmap	Bad-sector map

See Also

mknod(C), mkfs(M), shutype(M)

Name

ldunix - Altos configurable kernel linker.

Syntax

ldunix [**-d** *boot_directory*] **-k** *kernel_file*]
[**-s** *system_file*]

Description

Ldunix will link special object file modules produced by **mkboot**(M) creating kernel and symbol table image files. These image files can then be processed by **mkunix**(M) to yield a bootable kernel file.

Ldunix is a utility based on the auto-configuration boot procedure. It allows users to reconfigure a unix kernel file to reflect changes in tuneable parameters, or the addition of special purpose kernel drivers.

To create the image files, **ldunix** uses the KERNEL and system files from the current directory and the special object files from the boot.d directory. The **-d**, **-k**, and -s options can be used to explicitly specify the pathnames for **ldunix** to use for boot.d, KERNEL, and system, respectively.

When **ldunix** links in the modules specified by the master files and by the system file, it checks for functions with specific names in modules that are drivers. The names checked for are formed by concatenating the prefix specified in the master file and the desired suffix. For example, in a driver with the prefix "hd," if **ldunix** is checking for the suffix "intr," it will look for the function "hdintr." In most cases, if the routine is not found, the appropriate table entry gets the entry for the "nodev" routine. In the case of the "rstrt," "shut," and "init" suffixes, if there is no matching routine, no entry is made in the table.

The following suffixes are checked by **ldunix** for each load module of the given type:

block device drivers:

intr	interrupt handler
open	open routine
close	close routine
strategy	strategy routine
print	routine to call to report device errors

character device drivers (including streams drivers):

intr	interrupt handler
open	open routine
close	close routine
read	read routine
write	write routine

all drivers:

rstrt	restart routine to be called when power is restored after a power failure (if UPS is installed)
shut	shutdown routine to be called when power fails (if UPS is installed)
init	routine to be called to initialize the driver (called after all other kernel initialization is completed)

Files

kimage	Kernel image file
ksymbols	Kernel symbol table file

See Also

mkboot(M), mkunix(M)

Name

link, unlink - Links and unlinks files and directories.

Syntax

/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**(C) and **rmdir**(C) commands be used instead of the **unlink** command.

The only difference between **ln**(C) 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**(S) and **unlink**(S) system calls.

See Also

rm(C) and link(S), unlink(S) in the *Reference (CP, S, F)*

Notes

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

Name

log - Interface to STREAMS error logging and event tracing.

Description

Log is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes (**strerr**(M), **strace**(M)). **Log** presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit **log** messages; and a subset of **ioctl**(S) system calls and STREAMS messages for interaction with a user level error logger, a trace logger, or processes that need to submit their own **log** messages.

Kernel Interface

Log messages are generated within the kernel by calls to the function **strlog:**

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
```

Required definitions are contained in **<sys/strlog.h>** and **<sys/log.h>**. *Mid* is the STREAMS module id number for the module or driver submitting the **log** message. *Sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *Level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *Flags* are any combination of SL_ERROR (the message is for the error logger), SL_TRACE (the message is for the tracer), SL_FATAL (advisory notification of a fatal error), and SL_NOTIFY (request that a copy of the message be mailed to the system administrator). *Fmt* is a **printf**(S) style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to NLOGARGS (currently 3) numeric or character arguments can be provided.

User Interface

Log is opened via the **clone** interface, /dev/log. Each open of /dev/log obtains a separate *stream* to **log**. In order to receive **log** messages, a process must first notify **log** whether it is an error logger or trace logger via a STREAMS I_STR **ioctl** call (see below). For the error logger, the I_STR **ioctl** has an ic_cmd field of I_ERRLOG, with no accompanying data. For the trace logger, the **ioctl** has an ic_cmd field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct trace_ids elements. Each trace_ids structure specifies an *mid*, *sid*, and *level* from which messages will be accepted. **Strlog** will accept messages whose *mid* and *sid* exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of -1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

At most one trace logger and one error logger can be active at a time. Once the logger process has identified itself via the **ioctl** call, **log** will begin sending up messages subject to the restrictions noted above. These messages are obtained via the **getmsg**(S) system call. The control part of this message contains a log_ctl structure which specifies the *mid*, *sid*, *level*, *flags*, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, and a sequence number. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of **log** messages can be determined.

Different sequence numbers are maintained for the error and trace logging *streams*, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to **log**, even if it is not an error or trace logger. The only fields of the log_ctl structure in the control part of the message that are accepted are the level and flags fields; all other fields are filled in by **log** before being forwarded to the appropriate logger. The data portion must be packed one word each, on the next word boundary following the end of the format string.

Attempting to issue an I_TRCLOG or I_ERRLOG when a logging process of the given type already exists will result in the error ENXIO being returned. Similarly, ENXIO is returned for I_TRCLOG **ioctls** without any trace_ids structures, or for any unrecognized I_STR **ioctl** calls. Incorrectly formatted **log** messages sent to the driver by a user process are silently ignored (no error results).

Examples

Example of I_ERRLOG notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timeout = 0;     /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I-TRCLOG notification.

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;      /* any sub-id will be allowed */
tid[1].ti_level = -1;    /* any level will be allowed */
```

```
ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struc trace_ids);
ioc.ic_dp = char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a log message (no arguments).

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk \
                 on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERRORSL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

Files

/dev/log
<sys/log.h>
<sys/strlog.h>

See Also

strace(M), strerr(M), clone(M), and intro(S), getmsg(S), putmsg(S) in *Reference (CP, S, F) STREAMS Programmer's Guide*

Name

lpadmin - Configures the LP spooling system.

Syntax

/usr/lib/lpadmin -p*printer* [*options*]
/usr/lib/lpadmin -x*dest*
/usr/lib/lpadmin -d[*dest*]

Description

Lpadmin configures LP spooling systems to describe printers, classes, and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and change the system default destination. **Lpdamin** may not be used when the LP scheduler, **lpsched**(M), is running, except where noted below.

Exactly one of the **-d**, **-p**, or **-x** options must be present for every legal invocation of **lpadmin.**

-d[*dest*] Makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when **lpsched**(M) is running. No other options are allowed with **-d.**

-p*printer* Names a *printer* to which all of the options below refer. If *printer* does not exist then it will be created.

-x*dest* Removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other options are allowed with **-x.**

The following options are only useful with **-p** and may appear in any order. For ease of discussion, the printer will be called *P*.

-c*class*	Inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.
-e*printer*	Names a *printer* to which all of the options below refer. If *printer* does not exist then it will be created.
-h	Indicates that the device associated with *P* is hardwired. This option is assumed when creating a new printer unless the -l option is supplied.
-i*interface*	Establishes a new interface program for *P*. *Interface* is the path name of the new program.
-m*model*	Selects a *model* interface program for *P*. *Model* is one of the model interface names supplied with the LP software (see Models below).
-r*class*	Removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.
-v*device*	Associates a new *device* with printer *P*. *Device* is the path name of a file that is writable by the LP administrator, **lp**. Note that there is nothing to stop an administrator from associating the same device with more than one printer. If only the -p and -v options are supplied, then **lpadmin** may be used while the scheduler is running.

Restrictions

When creating a new printer, the -v option and one of the -e, -i, or -m options must be supplied. Only one of the -e, -i, or -m options may be supplied. The -h and -l keyletters are mutually exclusive. *Printer* and *class* names may be no longer than 14 characters and must consist entirely of the characters A-Z, **a-z**, **0-9**, and _ (underscore).

Models

Model printer interface programs are supplied with the LP software. They are shell procedures that interface between **lpsched**(M) and devices. All models reside in the directory /usr/spool/lp/model and may be used as is with **lpadmin -m.** Models should have 644 permission if owned by lp and bin, or 664 permission if owned by bin and bin. Alternatively, LP administrators may modify copies of models and then use **lpadmin -i** to associate them with printers. The following list describes the models and lists the options which they may be given on the **lp** command line using the **-o** keyletter:

dumb Interface for a line printer without special functions and protocol. Form feeds are assumed. Use this model to copy and modify (for printers that do not have models).

Examples

1. To create a printer named hp2 on port 02, use the commands:

   cd /usr/lib
   lpshut
   xtty disable tty02
   lpadmin -php2 -v/dev/tty02 -mdumb
   accept hp2
   lpenable hp2
   lpsched

2. To print on hp2, use the command:

   lp -dhp2 *files*

Files

/usr/spool/lp/*

See Also

accept(C), lpenable(C), lp(C), lpsched(M), lpstat(C)

Name

lpd - Line printer daemon.

Syntax

lpd *n*

Description

The **lpd** command is the line printer daemon which supports multiple printer spooling. The **lpd** command is executed automatically by the **lpr**(C) command. A single daemon is used per printer device, and daemons are invoked only if there is currently no daemon active. The **lpd** command does not engage in any filtering of the data to the printer, hence printer control codes, escape sequences and other binaries will be reproduced. For serial printers, **lpr**(C) supplies **lpd** with a tty modes setting which is non-destructively used to print individual requests. The **lpd** command restores tty modes between each request, and at exit time.

The **lpr** command decides whether to invoke the **lpd** daemon based on the presence (or absence) of a "lock" file in each spool directory. A daemon will run until there is no more output for its printer. It also removes its lock file so that a new daemon may be started up. If the daemon were to terminate before removing its lock file, the lock file must be removed from its spool directory before printing can be resumed. The **lpd** command prints an optional header (specified in **lpr**), followed by a sequence of files (each followed by a formfeed).

Options

n *N* is a number that selects a spool directory and printer device. If *n* were specified as "2", /usr/spool/lpd2 and /dev/lp2 would be selected. If no number is supplied, then **lpd** assumes /dev/lp and /usr/spool/lpd. The **lpr** command invokes **lpd** with an appropriate printer selector digit.

Related Commands

lpr(C), printers(M)

Files

/usr/spool/lpd?	spool directories
/dev/lp*	printer devices
/usr/spool/lpd?/lock	lock file

Name

lpinit - Adds new lineprinters to the system.

Syntax

/usr/lib/lpinit

Description

Lpinit is a shell script for configuring and adding new lineprinters to a system. It should only be executed by the super user.

Lpinit asks a series of questions for which the default answers are displayed. You can type a response or press Retn for the default answer. If you type a response to the first question, a Help message is displayed. **Lpinit** prompts for the following information:

- The print device pathname (default is /dev/lp).
- The name of the printer (default is linepr).
- The pathname of the printer interface program (default is /usr/spool/lp/model/dumb).

The printer name can be any combination of up to 14 alpha numeric characters or underscores. A printer interface program can be a shell script, C program, or any executable program; or the model interface program, /usr/spool/lp/model/dumb, can be copied and modified.

After you have responded to these questions, **lpinit** stops the print scheduler, lpsched, changes the acceptance status of the new lineprinter to accept, and enables it to print files. **Lpinit** then asks if the new printer will be the default printing destination (default is Yes). All nonspecific print requests are routed to the default destinations (see **lp**(C)).

The steps to configure a new printer can be taken separately (see **lpadmin**(M), **accept**(C), **lpenable**(C), **lpsched**(M) for details).

Files

/usr/lib/lpinit

See Also

accept(C), lpenable(C), lp(C), lpadmin(M), lpsched(M)

Name

lpon, lpoff - Turns on/off line printer scheduling.

Syntax

lpon
lpoff

Description

By default, line printer scheduling is activated in Altos System V, version 5.3d. If there is no line printer attached to the system, this scheduling is superfluous; printer scheduling may be stopped, and boot-time startup of scheduling permanently disabled by using the **lpoff** command. If a printer is added to a system that has printer scheduling disabled, the **lpon** command will start scheduling and enable boot-time scheduling startup.

Files

/etc/init.d/lpsched
/etc/rc0.d/K36lpsched
/etc/rc2.d/S38lpsched
/etc/rc2.d/S02.printers
/etc/rc2.d/s02.printers

See Also

lp(C), lpenable(C), lpdisable(C)

Name

lpsched, lpshut, lpmove - Starts/stops the LP request scheduler and moves requests.

Syntax

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove *request... dest*
/usr/lib/lpmove *dest1 dest2*

Description

Lpsched schedules requests taken by **lp**(C) for printing on line printers.

Lpshut shuts down the line printer scheduler. All printers that are printing at the time **lpshut** is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after **lpsched** is started again. All LP commands perform their functions even when **lpsched** is not running.

Lpmove moves requests that were queued by **lp**(C) between LP destinations. You can use this command only when **lpsched** is not running.

The first form of the command moves the named requests to the LP destination, *dest*. Requests are request ids as returned by **lp**(C). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, **lp**(C) will reject requests for *dest1*.

Note that **lpmove** never checks the acceptance status (see **accept**(C)) for the new destination when moving requests.

Files

/usr/spool/lp/*

See Also

accept(C), lp(C), lpstat(C)

Name

makedevs - Creates special device files.

Syntax

/etc/makedevs *directory*

Description

Makedevs creates all the special device files in the specified *directory* supported by the operating system.

Makedevs is normally run to create the device files for the hard disk at installation time, and to repair the device directory (/dev).

See Also

mknod(C)

Name

makekey - Generates an encryption key.

Syntax

/usr/lib/makekey

Description

Makekey improves the usefulness of encryption schemes by increasing the amount of time required to search the key-space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way that is intended to be difficult to compute (i.e., requires a substantial fraction of a second).

The first eight input bytes (the input key) can be arbitrary ASCII characters. The last two input bytes (the salt) are best chosen from the set of digits, dot (.), slash (/), and uppercase and lowercase letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the output key.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the input key as the key, a constant string is fed into the machine and recirculated. The 64 bits that come out are distributed into the 66 output key bits in the result.

Makekey is intended for use with programs that perform encryption (e.g., **passwd**(M)). Usually its input and output will be pipes.

See Also

ed(C), vi(C), passwd(M)

Name

makettys - Creates tty special files.

Syntax

/etc/makettys [*directory*]

Description

The **makettys** command creates all the special files in the specified *directory* (/dev by default) for all the serial ports (tty special files) supported by the operating system and installed hardware.

Execute this command in single-user mode.

This is done by executing the IOCHOWMANY ioctl to determine how many ports are supported for each type of communications board that is installed. If necessary, it will first remove incorrect entries. It will NOT remove special files that are not supported by the current hardware. (This could happen after a board has been removed.)

All files created have the prefix "tty," and up to three decimal digits appended. (For compatiblity, ports 1-9 become tty01 - tty09.)

Currently, **makettys** supports only the SIO and Multidrop boards; other devices may be supported in the future.

Makettys is normally run from /etc/brc on every system boot to ensure that all tty devices are correct.

Files

/dev default directory

See Also

mknod(C)

Diagnostics

Messages appear if **makettys** can't change to the correct directory, if it is unable to execute the IOCHOWMANY ioctl, or can't create the special files.

Makettys will not make the pseudo file /dev/tty.

Name

master - Master configuration database.

Description

The **master** configuration database is a collection of files. Each file contains configuration information for a device or module that may be included in the system. A file is named with the module name to which it applies. This collection of files is maintained in a directory called /usr/sys/master.d. Each individual file has an identical format. For convenience, this collection of files will be referred to as the **master** file, as though it was a single file. This will allow a reference to the **master** file to be understood to mean the *individual file* in the master.d directory that corresponds to the name of a device or module.

The file is used by the **mkboot**(M) program to obtain device information to generate the device driver and configurable module files. It is also used by the **sysdef**(M) program to obtain the names of supported devices. **Master** consists of two parts; they are separated by a line with a dollar sign ($) in column 1.

- Part 1 contains device information for both hardware and software devices, and loadable modules.
- Part 2 contains parameter declarations used in part 1. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1, Description

Hardware devices, software drivers, and loadable modules are defined with a line containing the following information. Field 1 must begin in the left-most position on the line. Fields are separated by white space (tab or blank).

Field 1:	Element characteristics:	
	o	Specify only once
	r	Required device
	b	Block device
	c	Character device
	a	Generate segment descriptor array
	t	Initialize cdevsw[].d_ttys
	s	Software driver
	f	STREAMS driver
	m	STREAMS module
	x	Not a driver; a loadable module
	number	The first interrupt vector for a device
Field 2:	Number of interrupt vectors required by a hardware device; "-" if none	
Field 3:	Handler prefix (4 chars. maximum)	
Field 4:	Software driver external major number; "-" if not a software driver, or to be assigned during execution of **ldunix**(M)	
Field 5:	Number of sub-devices per device; "-" if none	
Field 6:	Mask of which CPU's driver can run on; "-" if driver doesn't have multiprocessor knowledge	
Field 7:	Dependency list (optional); this is a comma separated list of other driver or modules that must be present in the configuration if this module is to be included.	

For each module, two classes of information are required by **mkboot**(M):

- External routine references
- Variable definitions

Routine and variable definition lines begin with white space and immediately follow the initial module specification line. These lines are free form; thus they may be continued arbitrarily between non-blank tokens as long as the first character of a line is white space.

Part 1, Routine Reference Lines

If the system kernel or other dependent module contains external references to a module, but the module is not configured, then these external references would be undefined. Therefore, the routine reference lines are used to provide the information necessary to generate appropriate dummy functions at boot time when the driver is not loaded. Routine references are defined as follows:

Field 1: Routine name ()

Field 2: The routine type: one of

{}	routine_name(){}
{nosys}	routine_name(){return nosys();}
{nodev}	routine_name(){return nodev();}
{false}	routine_name(){return 0;}
{true}	routine_name(){return 1;}
{pass}	routine_name(){return first_argument;}

Part 1, Variable Definition Lines

Variable definition lines are used to generate all variables required by the module. The variable generated may be an arbitrary size, initialized or not, or arrays containing an arbitrary number of elements. These variables are defined as follows:

Field 1: *Variable_name*

Field 2: [*expr*] - optional field used to indicate array size

Field 3: (*length*) - required field indicating the size of the variable (see below)

Field 4: ={ *expr*,...} - optional field used to initialize individual elements of a variable

The *length* field is mandatory. It is an arbitrary sequence of length specifiers, each of which may be one of the following:

%i	Integer
%l	Long integer
os	Short integer
%c	Single character
%*number*	Field which is *number* bytes long
%*number* c	Character string which is *number* bytes long
%*vname*	*Length* is the value that variable *name* was initialized with in the corresponding **boot.d** module

For example, the length field

(%8c%l%0x58%1%c%c)

could be used to identify a variable consisting of a character string 8-bytes long, a long integer, a 0x58 byte structure of any type, another long integer, and two characters. Appropriate alignment of each % specification is performed (%*number* is word aligned) and the variable length is rounded up to the next word boundary during processing.

The expressions for the optional array size and initialization are infix expressions consisting of the usual operators for addition, subtraction, multiplication, and division: +, -, *, and /. Multiplication and division have the higher precedence, but parentheses may be used to override the default order. The built-in functions **min** and **max** accept a pair of expressions, and return the appropriate value. The operands of the expression may be any mixture of the following:

&name Address of name where *name* is any symbol defined by the kernel, any module loaded or any variable definition line of any module loaded

#name Size of name where *name* is any variable name defined by a variable definition for any module loaded; the size is that of the individual variable, not of an entire array

#C	Number of controllers present; this number is determined by the EDT for hardware devices, or by the number provided in the system file for non-hardware driver or modules
#C(*name*)	Number of controllers present for the module *name*; this number is determined by the EDT for hardware devices, or by the number provided in the system file for nonhardware driver or modules
#D	Number of devices per controllers taken directly from the current master file entry
#D(*name*)	Number of devices per controller taken directly from the master file entry for the module *name*
#M	Internal major number assigned to the current module if it is a device driver; zero if this module is not a device driver
#M(*name*)	Internal major number assigned to the module *name* if it is a device driver: zero if that module is not a device driver
name	Value of a parameter as defined in the second part of **master**
number	Arbitrary number (octal, decimal, or hex allowed)
string	Character string enclosed within double quotes (all of the character string conventions supported by the C language are allowed); this operand has a value which is the address of a character array containing the specified *string*

When initializing a variable, provide one initialization expression for each %i, %l, %s, or %c of the length field. The only initializers allowed for a '%*number* c' are either a character string (the string may not be longer than *number*), or an explicit zero. Initialization expressions must be separated by commas, and variable initialization will proceed element by element. Note that %*number* specifications cannot be initialized -- they are set to zero.

Only the first element of an array can be initialized, the other elements are set to zero. If there are more initializers than size specifications, it is an error and execution of the **mkboot**(M) program will be aborted. If there are fewer initializations than size specifications, zeros will be used to pad the variable. For example:

={ "V2.L1", #C*#D, max(10,#D), #C(OTHER), #M(OTHER)}

would be a possible initialization of the variable whose length field was given in the preceding example.

Part 2, Description

Parameter declarations may be used to define a value symbolically. Values can be associated with identifiers and these identifiers may be used in the variable definition lines.

Parameters are defined as follows:

Field 1: *Identifier* (8 characters maximum)

Field 2: =

Field 3: *Value* - the value may be a number (decimal, octal, or hex allowed), or a string

Example

A sample **master** file for a tty device driver would be named **atty** if the device appeared in the EDT as ATTY. The driver is a character device, the driver prefix is **at**, two interrupt vectors are used, and the interrupt priority is 6. In addition, another driver named ATLOG is necessary for the correct operation of the software associated with this device.

```
* FLAG #VEC PREFIX SOFT #DEV CPU DEPENDENCIES/ VARIABLES

tca     2     at      -     2       ATLOG
                                      atpoint(){false}
                                      at_tty[#C*#D] (%0x58)
                                      at_cnt(%i) ={ #C*#D}
                                      at_logmaj (%i) ={#M(ATLOG)}
                                      at_id(%8c) ={ ATID}
                                      at_table(%i%1%31%s)
                                      ={ max(#C ATMAX),
                                      &at_tty,
                                      #C }
$
ATID="fred"
ATMAX=6
```

This **master** file will cause a routine named **atpoint** to be generated by the **mkboot**(M) program if the ATTY driver is not loaded, and there is a reference to this routine from any other module loaded. When the driver is loaded, the variables *at_tty*, *at_cnt*, *at_logmaj*, *at_id*, and *at_table* will be allocated and initialized as specified. Due to the **t** flag, the *d_ttys* field in the character device switch table will be initialized to point to *at_tty* (the first variable definition line contains the variable whose address will be stored in *d_ttys*). The ATTY driver would reference these variables by coding:

```
extern struct tty at_tty[];
extern int at_cnt;
extern int at_logmaj;
extern char at_id[8];
extern struct




{
  int member1;
  struct tty *member2;
  char junk[31];
  short member3;
} at_table;
```

Files

/usr/sys/master.d/*

See Also

ldunix(M), mkboot(M), sysdef(M)

Name

mem, kmem - Memory image file.

Description

The **mem** file provides access to the computer's physical memory. All byte addresses in the file are interpreted as memory addresses. Thus, memory locations can be examined in the same way as individual bytes in a file. Note that accessing a nonexistent location causes an error.

The **kmem** file is the same as **mem**, except that it corresponds to kernel virtual memory rather than physical memory.

In rare cases, the **mem** and **kmem** files may be used to write to memory and memory-mapped devices. Such patching is not intended for the naive user and may lead to a system crash if not conducted properly. Patching device registers is likely to lead to unexpected results if the device has read-only or write-only bits.

Files

/dev/mem
/dev/kmem

Notes

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

Name

menus - Format of a Business Shell menu system.

Description

A menu system is defined as a collection of menus, each of which is an ASCII text file. It is relatively easy to create a new customized Business Shell (**bsh**(C)) menu system or to modify the default menu system. The procedure to create a menu system follows.

To create a text file containing the source menu, use the following format:

```
&Menuidentifier
        . . .  the substance of the menu . .
        . . .  not over 24 lines length

&Actions
        . . .  zero or more sequences of . . .

~ prompt size

        . . .  sequences of actions . . .
        for this prompt . . .
```

This sequence may be repeated as often as desired. The ampersand (&) and tilde (~) must appear in the first column. &Actions must appear, even if there are no actions.

The substances of each menu is composed of text which will be reproduced exactly as it appears in the location where it appears. There are five exceptions where characters have special meanings:

"~*string*" denotes a valid "prompt" *string* (the text of the actual prompt).

"!date" inserts the current *date* and *time*.

"!user" inserts the current *user id*.

"!pwd" inserts the current directory.

"!@" indicates where to leave the cursor.

The "!" may appear as a suffix, in which case the string will be right-justified instead of left-justified.

The prompts must be reproduced as they are expected to be typed in the Actions chapter. The actions may be composed of **bsh** commands or commands which are executed by the standard shell (/bin/sh). The actions should all be indented one tab stop.

Size rows will be reserved at the bottom of the screen for output. If *size* is omitted, a value of 5 will be used. If *size* is 0, the entire screen will be used. After executing the actions, the message

```
[Type return to continue]
```

will appear at the bottom of the screen. If *size* is -1 the entire screen is used, but no message is issued; and **bsh** resumes without pause after all the actions have been executed.

Transfer to another menu is specified by writing the name of the destination menu in the semantics field.

Commands to be executed by the **bsh** interpreter must be typed one-per-line.

Commands to be executed by the operating system follow the usual conventions.

For example, the menu for Electronic Mail can be created as follows:

```
&Mail

!date          \ELECTRONIC~MAIL~SERVICES

               ~a - Receive~mail
               ~b - Send~mail
               ~c - Return~to~starting~menu

&Actions
~a      0
        mail

~b      -1

        echo -n "To whom do you wish to send mail?"
        read x
        echo "Now type the message."
        echo "Terminate it by typing a control -d."
        mail $x

~c
        Start
```

See Also

bsh(C), termcap(M)

Name

mkboot - Converts an object file to a bootable object file.

Syntax

/etc/mkboot [**-m** *master*] [**-d** *directory*] [**-k** *kernel*.**o**] *driver*.**o** ...

Description

The **mkboot** command is used to create a bootable object file in a format compatible with the self-configuration program. It can only be used by the super-user. The object file specified as an argument must have a corresponding **master**(M) file in the /usr/sys/etc/master.d directory. The master file name for the UNIX system kernel object file is always kernel. The other master file names derive from their associated object file names in lowercase letters minus any optional path prefix or "**.o**" suffix.

To create the new bootable object file, the applicable master file is read and the configuration information is extracted. Then, the new bootable file is created containing this configuration information and written to the /usr/sys/boot.d directory. It is given the same name, in uppercase letters and without the "**.o**" suffix, as the object file. Note that if the current working directory is /usr/sys/boot.d when **mkboot** is executed, then the object file used is the previous bootable object file residing in this directory. This means that you do not have to keep separate "**.o**" files.

The options are:

-m *master* This option specifies the directory containing the master files to be used for the object file. The default *master* directory is /usr/sys/master.d.

-d *directory* This option specifies the directory to be used for storing the new bootable object file. The default output *directory* is /usr/sys/boot.d.

-k *kernel*.**o** This option specifies the name of the object file for the operating system. The master file name used for this object file is always named kernel.

The name of the object file for a module or driver is specified by the *driver*.**o** argument.

Example

mkboot -m newmaster gentty.o

This will read the file name *gentty* from the directory *newmaster* for the *gentty* device configuration data, take the file *gentty.o* from the current directory and create the formatted file /usr/sys/boot.d/GENTTY containing the configuration information for the *gentty*.

See Also

mkunix(M), master(M)

Diagnostics

Most messages are self-explanatory.

name.**o: not processed; cannot open /etc/master.d/***name*

The file *name*.**o** was specified on the command line but there was no master file in the master.d directory for *name*.**o**.

name.**o: not processed**

An error has aborted processing for the named object file.

Name

mkfs - Constructs a file system.

Syntax

/etc/mkfs *special blocks*[*:inodes*] [*gap blocks/cyl*]
/etc/mkfs *special proto* [*gap blocks/cyl*]

Description

Mkfs constructs a file system by writing on the *special* file using the values found in the remaining arguments of the command line. The command waits 10 seconds before starting to construct the file system. During this 10-second pause the command can be aborted by entering a delete (Break/Del).

If the second argument is a string of digits, the size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* (512 byte) disk blocks the file system will occupy. If the number of inodes is not given, the default is the number of *logical* (1024 byte) blocks divided by 4. **Mkfs** builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file that can be opened, **mkfs** assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1.   /stand/diskboot
2.   4872 110
3.   d--777 3  1
4.   usr       d--777 3  1
5.             sh        ---755 3 1 /bin/sh
6.             ken       d--755 6 1
7.                       $
8.             b0        b--644 3 1 0 0
9.             c0        c--644 3 1 0 0
10.            $
11.  $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program.

Line 2 specifies the number of *physical* (512 byte) blocks the file system is to occupy and the number of inodes in the file system. Lines 3-9 tell **mkfs** about files and directories to be included in this file system.

Line 3 specifies the root directory.

Lines 4-6 and 8-9 specifies other directories and files.

The $ on line 7 tells **mkfs** to end the branch of the file system it is on, and continue from the next higher directory. The $ on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **-bcd** to specify regular, block special, character special and directory files respectively. The second character of the mode is either **u** or - to specify set-user-id mode or not. The third is **g** or - for the set-group-id mode. The rest of the mode is a 3 digit octal number giving the owner, group, and other read, write, execute permissions (see **chmod**(C)).

Two decimal number tokens come after the mode; they spec fy the user and group IDs of the owner of the file.

If the file is a regular file, the next token of the specification may be a path name whence the contents and siz are copied. If the file is a block or character special file, two decimal numbers follow which give the major and minor device numbers. If the file is a directory, **mkfs** makes the entries . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token **$**.

The final argument in both forms of the command specifies the rotational *gap* and the number of *blocks/cyl*. The following values are recommended:

Device	Gap Size	Blks/Cyl
30M Hard Disk	8	90
72M Hard Disk	8	162 (CDC Wren II)
72aM Hard Disk	8	144 (Micropolis)
72bM Hard Disk	8	198 (Priam)
72cM Hard Disk	8	198 (Fujitsu)
Floppy Disk	4	18

Mkfs uses a gap size in multiples of 4. If the *gap* and *blocks/cyl* are not specified or are considered illegal values a default value of gap size 4 and 400 blocks/cyl is used.

See Also

chmod(C), dir(F), and fs(F) in the *Reference (CP, S, F)*

Notes

With a prototype file, it is not possible to copy in a file larger than 64K bytes, nor is there a way to specify links. The maximum number of inodes configurable is 65500.

Name

mkunix - Makes a bootable system file with kernel and driver symbol tables.

Syntax

/etc/mkunix [*-i kernel_file*] [**-o** *unix_file*]

Description

The **mkunix** command will create an absolute, bootable system file (*new_namelist*) from the UNIX system kernel file and the object files created by **mkboot**(M). This procedure completes the generation of a new /unix. It can only be used by the super-user.

The resulting *unix_file* can be used as the *kernel_file* for ps(C), etc. In addition, this file may be booted directly, bypassing the self-configuration phase of the boot process. This will save on the order of 30 to 60 seconds at boot time.

Kernel_file (defaults to the path name specified as the **BOOT** program in the /usr/sys/system file) is read to obtain the object, data, and symbol table for the basic kernel. This name, if specified, must be the same as that used in /usr/sys/system for the boot line; if not, a warning diagnostic is issued since the resulting namelist file will not be accurate.

The argument **-o** *unix_file* (defaults to **a.out**) is the new file - a bootable image of the current operating system with the composite symbol table.

See Also

mkboot(M), ps(C), and nm(CP) in the *Reference (CP, S, F)*

Name

mnttab - Mounted file system table.

Syntax

#include <mnttab.h>

Description

The **/etc/mnttab** file contains a table of devices mounted by the **mount**(C) command.

Each table entry contains the pathname of the directory on which the device is mounted, the name of the device special file, the read/write permissions of the special file, and the date on which the device was mounted.

The maximum number of entries in **mnttab** is based on the system parameter NMOUNT located in /usr/include/mnttab.n, which defines the number of allowable mounted special files.

See Also

mount(C)

Name

multiuser, singleuser - Causes the system to enter multi-user or single-user mode.

Syntax

/etc/multiuser
/etc/singleuser

Description

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

Multiuser changes the system mode of operation from single-user to multi-user. **Multiuser** performs system startup functions such as mounting file systems and starting various daemons and spoolers. The **/etc/telinit 2** command is executed to tell **init**(M) to enter multi-user mode (run level 2).

Singleuser causes the system to kill all currently running processes and enter system maintenance mode (run level 1).

See Also

init(M), shutdown(M), who(C)

Name

ncheck - Generates path names from inode numbers.

Syntax

/etc/ncheck [**-i** *inode...*] [**-a**] [**-s**] [*file-system*]

Description

Ncheck with no arguments generates a path-name vs. inode list of all files on a set of default file systems (see /etc/checklist). Names of directory files are followed by /..

The options are as follows:

-i Limits the report to only those files whose inode numbers follow.

-a Allows printing of the names . and .., which are ordinarily suppressed.

-s Limits the report to special files and files with set-user-ID mode. This option may be used to detect violations of security policy.

File-system must be specified by the file system's special file. The report should be sorted so that it is more useful.

See Also

fsck(C), sort(C)

Diagnostics

If the file system structure is not consistent, ?? denotes the "parent" of a parentless file and a path-name beginning with ... denotes a loop.

Name

null - The null file.

Description

Data written on a null special file is discarded. Reads from a null special file always return 0 bytes.

Files

/dev/null

Name

options - Floppy disk installation menu.

Syntax

options

Description

The **options** command displays the installation menu on the operating system Root diskette.

To display this menu, first go to system maintenance mode. Then boot the system from the Root File System floppy disk. Type **options** to display the menu.

Use this menu to initially install or upgrade the operating system, restore data from a cartridge tape, shut down the system, or exit to the shell.

Name

passwd - The password file.

Description

The **/etc/passwd** file contains the following information for each user:

- Login name
- Encrypted password
- Numerical user ID
- Numerical group ID
- Comment
- 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 comment can contain any desired information; it typically contains the user's real name. Each user is separated from the next by a newline. If the password field is null, no password is demanded; if the shell field is null, the **sh**(C) command is used.

This file resides in the directory /etc. Because the passwords are encrypted, the file has 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 in effect for a particular user if his encrypted password in the password file is followed by a comma and a nonnull string of characters from the above alphabet. (Such a string must be introduced by the super-user.) The first character of the age denotes the maximum number of weeks for which a pa word is valid.

A user who attempts to log in after his password has expired will be forced to supply a new one. The next character 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.) The first and second characters must have numerical value in the range 0-63, where the dot (.) is equal to 0 and lowercase **z** is equal to 63. If the numerical value of both characters is 0, the user will be forced to change his password the next time he logs in. If the second character is greater than the first, only the super-user will be able to change the password.

Files

/etc/passwd

See Also

group(M), login(C), passwd(C)

Name

printers - Printer spooler configuration file.

Description

Using the printer spooler facility **lpr**(C), you can print a specified list of files on one or several line printers. Additionally, a printer on a machine connected to WorkNet can be shared by other machines on the same net. Such printers may need to have an arbitrary set of terminal modes set for tab expansion, baud rate, etc.

The system printer configuration file (/etc/printers) consists of lines of printer configuration information. These include WorkNet machine names, tty types, device names, and tty modes. Each line in the /etc/printers file is of the form:

lp[*p*]:*name*:*ttytype*:[*netname*]:[*ttymodes*]

Fields are separated by colons (:) and may not contain spaces between the colon separators and field values. The length of each line may not exceed 128 characters. Comments are permitted in the configuration file. A comment line begins with "#" in the first column. Any fields surrounded by "[]" are optional, although their colon separators are not. That is, if a field position is to be empty, its place must be marked by two colons (::).

The fields are:

lp[*p*] The printer device selected. Allowable values for *p* are null or 0 - 255. This value is used to specify one of several printers.

name A tag by which a particular configuration line can be selected. Allowable values are alphanumeric strings, which do not contain the ":" character.

ttytype Exists for the convenience of word processing programs that derive printer control sequences from /etc/termcap (or similar database). (Not used by the printer spooler.)

netname May be null, which indicates that spooling is to take place on the requestor's machine. Other values are network machine names. The print spooler uses this name to do remote printing.

ttymodes A list of whitespace-delimited tty mode specifications, such as would be supplied to stty.

Example

The following example shows the contents of a printer configuration file (the contents of /etc/printers):

```
# a printer configuration file
lp:calcite:NEC3510:gateway:
lp:galena:Oki93::-tabs 1200 nl
lp0:obsidian:I9:Marketing:tabs 9600 nl
lp1:feldspar:epson::nl tabs 9600
lp2:mica:TI810:Finance:9600 -tabs
```

In this example:

The first line uses the /dev/lp printer on the machine named "gateway." The printer type is "NEC3510" and no tty modes are set on that printer. This line may be selected by specifying "calcite" to **lpr.**

The next line specifies the /dev/lp printer on the user's local machine (note the null *netname* field), is type Oki93 and sets tab expansion (-tabs), 1200 baud operation, and no linefeed to cr-lf expansion. This line is selected with the name "galena."

The third line requests /dev/lp0, is on the Marketing machine, runs the printer at 9600 baud, etc., is type I9, and is selected by the name "obsidian."

The last two lines use /dev/lp1 on the local machine, and /dev/lp2 on the Finance machine.

Files

/etc/printers	Printer mode control file

Related Commands

lpr(C), lpd(M), tty(M), lp(C)

Name

profile - Sets up an environment at login time.

Description

The optional file .profile permits automatic execution of commands when a user logs into /bin/sh and other shells (except /bin/csh). Use this file to personalize a user's work environment by setting exported environment variables and terminal mode (see **environ**(M)).

When a user logs in, the user's login shell looks for .profile in the login directory. If found, the shell executes the commands in the file before beginning the session. The commands in the file must match the command as if typed at the keyboard. Any line beginning with the number sign (#) is considered a comment and is ignored. The following is an example of a typical file:

```
# Tell me when new mail comes in
MAIL=/usr/mail/myname
# Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 22
```

The file /etc/profile is a system-wide profile that, if it exists, is executed for every user before the user's .profile is executed.

Files

$HOME/.profile
/etc/profile

See Also

env(C), mail(C), sh(C), stty(C), su(C), login(M), environ(M)

Name

pwck, grpck - Checks password/group file.

Syntax

/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(M), passwd(M)

Diagnostics

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

Name

rc0 - Runs commands performed to stop the operating system.

Syntax

/etc/rc0

Description

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown." This command can be used only by the superuser.

The system state that requires this procedure is:

state 0 - system halt state

Whenever a change to one of these states occurs, the **/etc/rc0** procedure is run. The entry in **/etc/inittab** might read:

```
hlt0:0:once:/etc/rc0 </dev/console >/dev/console 2>&1
```

Some of the actions performed by **/etc/rc0** are carried out by files beginning with K in **/etc/rc0.d.** These files are executed in ASCII order (see files below for more information), terminating some system service. The combination of commands in **/etc/rc0** and files in **/etc/rc0.d** determines how the system is shut down.

The recommended sequence for **/etc/rc0** is:

1. Stop System Services and Daemons.

   Various system services (such as a local area network or LP spooler) are gracefully terminated.

   When new services are added that should be terminated when the system is shut down, the appropriate files are installed in **/etc/rc0.d.**

2. Terminate Processes

   SIGTERM signals are sent to all running processes by **killall**(C). Processes stop themselves cleanly if sent SIGTERM.

3. Kill Processes

   SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.

   At this point the only processes left are those associated with **/etc/rc0** and processes 5 and 1, which are special to the operating system.

4. Unmount All File Systems

   Only the root file system (/) remains mounted.

   Depending on which system state the system ends up in (0 or 6), the entries in **/etc/inittab** will direct what happens next. If the **/etc/inittab** has not defined any other actions to be performed as in the case of system state 0, then the operating system will have nothing to do. It should not be possible to get the system's attention. The only thing that can be done is to turn off the power or possibly get the attention of a firmware monitor.

Files

The execution by **/bin/sh** of any files in **/etc/rc0.d** occurs in ASCII sort-sequence order. See **rc2**(M) for more information.

See Also

killall(C), rc2(M), shutdown(M)

Name

rc2 - Runs commands performed for multi-user environment.

Syntax

/etc/rc2

Description

This file is executed via an entry in **/etc/inittab** and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multi-user" state. This command can be used only by the super-user.

The actions performed by **/etc/rc2** are found in files in the directory /etc/rc.d and files beginning with S in /etc/rc2.d. These files are executed by /bin/sh in ASCII sort-sequence order (see "Files" for more information). When functions are added that need to be initialized when the system goes multi-user, an appropriate file should be added in /etc/rc2.d.

The functions done by **/etc/rc2** command and associated /etc/rc2.d files include:

- Setting and exporting the TZ variable.
- Setting-up and mounting the user (/usr) file system.
- Cleaning up (remaking) the /tmp and /usr/tmp directories.
- Loading the network interface and ports cards with program data and starting the associated processes.
- Starting the cron daemon by executing /etc/cron.
- Cleaning up (deleting) uucp lock, status, and temporary files in the /usr/spool/uucp directory.

Other functions can be added, as required, to support the addition of hardware and software features.

Examples

The following are prototypical files found in /etc/rc2.d. These files are prefixed by an S and a number indicating the execution order of the files.

```
MOUNTFILESYS
     #    Set up and mount file systems

     cd /
     /etc/mountall /etc/fstab
RMTMPFILES
     # clean up /tmp
     rm -rf /tmp
     mkdir /tmp
     chmod 777 /tmp
     chgrp sys /tmp
     chown sys /tmp
uucp
     #    clean-up uucp locks, status, and temporary files

     rm -rf /usr/spool/locks/*
```

The file /etc/TIMEZONE is included early in **/etc/rc2**, thus establishing the default time zone for all commands that follow.

Files

Here are some hints about files in /etc/rc.d:

The order in which files are executed is important. Since they are executed in ASCII sort-sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9] very early
[A-Z] early
[a-n] later
[o-z] last

Files in /etc/rc.d that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them.

Files in **/etc/rc2.d** must begin with an **S** or a **K** followed by a number and the rest of the file name. Upon entering run level 2, files beginning with **S** are executed with the **start** option; files beginning with **K**, are executed with the **stop** option. Files beginning with other characters are ignored.

See Also

rc0(M), shutdown(M)

Name

sadcon, sadcoff - Turns on/off system activity data collector.

Syntax

sadcon
sadcoff

Description

By default, the system activity data collector is deactivated in Altos System V, version 5.3d. The data collector may be started, and boot-time startup of the collector enabled by using the **sadcon** command. A subsequent **sadcoi** command will disable boot-time data collector startup.

Files

/etc/init.d/sadc
/etc/rc2.d/S34sadc
/usr/spool/cron/crontabs/sys
/usr/spool/cron/crontabs/adm

See Also

sar(C), cron(C)

Name

sar: sa1, sa2, sadc - System activity report package.

Syntax

/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**(C)) 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/sar*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 /usr/spool/cron/crontabs/sys (see **cron**(C)) entries:

```
0 * * * 0-6 /usr/lib/sa/sal
20,40 8-17 * * 1-5 /usr/lib/sa/sal
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script **sa2** writes a daily report in file /usr/adm/sa/sar*dd*. 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. The structure of the binary daily data file is:

```
struct sa
{
  struct sysinfo si;          /* see/usr/include/sys/sysinfo.h */
  struct minfo mi;            /* defined in sys/sysinfo.h */
  struck dinfo di;            /* RFS info defined in sys/sysinfo.h */
  int minserve, maxserve;     /* RFS server low and high water
                              * marks */
  int szinode;        /* current size of inode table */
  int szfile;         /* current size of file table */
  int szproc;         /* current size of proc table */
  int  szlckf;        /* current size of file record header table */
  int  szlckr;        /* current size of file record lock table */
  int mszinode;       /* size of inode table */
  int mszfile;        /* size of file table */
  int mszproc;        /* size of proc table */
  int  mszlckf;       /* maximum size of file record header table */
  int  mszlckr;       /* maximum size of file record lock table */
  long inodeovf;      /* cumulative overflows of inode table */
  long fileovf;       /* cumulative overflows of file table */
  long procovf;       /* cumulative overflows of proc table */
  time_t ts;          /* time stamp, seconds */
  long devio[NDEVS][4];       /* device unit information */
  #define IO_OPS      0       /* cumulative I/O requests */
  #define IO_BCNT     1       /* cumulative blocks transferred */
  #define IO_ACT      2       /* cumulative drive busy time in ticks */
  #define IO_RESP     3       /* cumulative I/O resp time in ticks */
};
```

Files

/usr/adm/sa/sa*dd*	Daily data file
/usr/adm/sa/sar*dd1*	Daily report file
/tmp/sa.adrfl	Address file

See Also

cron(C), sar(C)

Name

shutdown - Brings a system to single-user mode or to shutdown.

Syntax

/etc/shutdown [**-y**] [**-g***grace_period*] [**-i***init_state*]

Description

This command is executed by the super-user to change the state of the machine. By default, it brings the system to a state where only the console has access to the system. This state is traditionally called "single-user."

The command sends a warning message (via **wall**(C)) and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are as follows:

-y Pre-answers the confirmation question so the command can be run without user intervention. A default of 60 seconds is allowed between the warning message and the final message. Another 60 seconds is allowed between the final message and the confirmation.

-g*grace_period* Allows the super-user to change the numbe of seconds from the 60-second default. Yc can specify a number from 0 to 999 to delay shutdown for that amount of time following notification to the users. If 0 is entered, **shutdown** will be immediate, and i no parameter is given, 60 seconds is assumed.

-i*init_state* Specifies the state that **init**(M) is to be put in following the warnings, if any. By default, system state "s" is used (the same as states "1" and "S").

Other recommended system state definitions are:

state 0 Shut the machine down so it is safe to remove the power. Have the machine remove power if it can. The **/etc/rc0** procedure is called to so this work.

state 1, s, S
Bring the machine to the state traditionally called single-user. The **/etc/rc0** procedure is called to do this work. (Though **s** and **1** are both used to go to single-user state, **s** only kills processes spawned by **init** and does not unmount file systems. State 1 unmounts everything except root and kills all user processes, except those that relate to the console.

state5 Stop the system and go to the firmware monitor.

state 6 Stop the system and reboot to the state defined by the initdefault entry in **/etc/inittab.**

See Also

wall(C), init(M), rc0(M), rc2(M)

Name

shutype - UPS shutdown configuration utility.

Syntax

shutype [**-p**] [**-t***type*] [**-f***failtime*] [**-c***pwrcnt*] [**-u***upstime*] [**w***pwrtime*] [**-e***termtime*]

Description

The **shutype** command allows the alteration of the current configurable settings for a UPS power failure condition. The six configuration settings that can be changed are:

-t*type*

The type of shutdown that is to be initiated for a power failure condition. This option causes the following to occur: the **shutkill** command issues a SIGPWR signal t all processes, and then posts SIGTERM and SIGKILL signals to the processes; a **sync**(S command is then executed to maintain the integrity of the file system; a **shutsave** command delivers the SIGPWR signal to all processes, but saves memory to disk so a later restart can be attempted.

-f*failtime*

The time in ticks to wait to check for a power failure condition after the first power failure condition was detected. Thi is used to check if a power glitch only ha occurred.

-c*pwrcnt*

The maximum number of power failure inte rupts that can occur within the above FAILTIME time interval before the power source is considered to be unreliable.

-u*upstime*

The time in seconds that the UPS battery backup unit can operate reliably after power has been turned off.

-w*pwrtime*

The time in seconds for the system to wait after posting the SIGPWR signal to all pro cesses before initiating shutdown procedures.

-e*termtime* The time in seconds for the system to wait after posting the SIGTERM signal to all processes before posting the SIGKILL signal to all processes. This is only used when the **shutkill** option is in effect.

If no options are given, **shutype** will prompt you for each of the above parameters. A null response followed by a carriage return will leave the current configuration value the same.

The **-p** option will print out the current settings of the above mentioned configurable parameters. No other options are allowed to be given with the **-p** option.

A sanity check will be done on any and all of the values entered. If the shutdown type is **shutkill,** the total times of *termtime*, *pwrtime*, and *failtime* cannot exceed the value of *upstime*.

If the shutdown type is **shutsave,** the total times of *pwrtime* and *failtime* plus the estimated disk output time cannot exceed *upstime*. The estimated disk output time will be printed if no options are given, or the **-p** option is given. If there are any inconsistencies, appropriate error message will be output.

Only the super-user is allowed to change any of the above mentioned configurable parameters.

See Also

shuttype(S)

Name

strace - Prints STREAMS trace messages.

Syntax

strace [*mid sid level*]...

Description

Strace without arguments writes all STREAMS event trace messages from all drivers and modules to its standard output. These messages are obtained from the STREAMS log driver (**log**(M)). If arguments are provided they must be in triplets of the form *mid, sid, level,* where *mid* is a STREAMS module id number, *sid* is a sub-id number, and *level* is a tracing priority level. Each triplet indicates that tracing messages are to be received from the given module/ driver, sub-id (usually indicating minor device), and priority level equal to or less than the given level. The token *all* may be used for any member to indicate no restriction for that attribute.

The format of each trace message output is:

<seq> <time> <ticks> <level> <flags> <mid> <sid> <text>

where:

<seq>	trace sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot
<level>	tracing priority level
<flags>	E: message is also in the error log F: indicates a fatal error N: mail was sent to the system administrator
<mid>	module id number of source
<sid>	sub-id number of source
<text>	formatted text of the trace message

Once initiated, **strace** will continue to execute until terminated by the user.

Examples

Output all trace messages from the module or driver whose module id is 41:

```
strace 41 all all
```

Output those trace messages from driver/module id 41 with sub-ids 0, 1, or 2:

```
strace 41 0 1 41 1 1 41 2 0
```

Messages from sub-ids 0 and 1 must have a tracing level less than or equal to 1. Those from sub-id 2 must have a tracing level of 0.

Notes

Due to performance considerations, only one **strace** process is permitted to open the STREAMS log driver at a time. The log driver has a list of the triplets specified in the command invocation, and compares each potential trace message against this list to decide if it should be formatted and sent up to the **strace** process. Hence, long lists of triplets will have a greater impact on overall STREAMS performance. Running **strace** will have the most impact on the timing of the modules and drivers generating the trace messages that are sent to the **strace** process. If trace messages are generated faster than the **strace** process can handle them, then some of the messages will be lost. This last case can be determined by examining the sequence numbers on the trace messages output.

See Also

log(M), and *STREAMS Programmer's Guide*

Name

strclean - STREAMS error logger cleanup program.

Syntax

strclean [**-d** *logdir*] [**-a** *age*]

Description

Strclean is used to clean up the STREAMS error logger directory on a regular basis (for example, by using **cron**(M)). By default, all files with names matching **error.*** in **/usr/adm/streams** that have not been modified in the last 3 days are removed. A directory other than **/usr/adm/streams** can be specified using the **-d** option. The maximum age in days for a log file can be changed using the **-a** option.

Example

strclean -d/usr/adm/streams -a 3

has the same result as running **strclean** with no arguments.

Notes

strclean is typically run from **cron**(M) on a daily or weekly basis.

Files

/usr/adm/streams/error.*

See Also

cron(C), strerr(M), and *STREAMS Programmer's Guide*

Name

strerr - STREAMS error logger daemon.

Syntax

strerr

Description

Strerr receives error log messages from the STREAMS log driver (**log**(M)) and appends them to a log file. The error log files produced reside in the directory /usr/adm/streams, and are named error.*mm-dd*, where *mm* is the month and *dd* is the day of the messages contained in each log file.

The format of an error log message is:

<seq> <time> <ticks> <flags> <mid> <sid> <text>

where:

<seq>	error sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot
<flags>	T: message was also sent to a tracing process F: indicates a fatal error N: send mail to the system administrator
<mid>	module id number of source
<sid>	sub-id number of source
<text>	formatted text of the error message

Messages that appear in the error log are intended to report exceptional conditions that require the attention of the system administrator. Those messages which indicate the total failure of a STREAMS driver or module should have the F flag set. Those messages requiring the immediate attention of the administrator will have the N flag set, which causes the error logger to send the message to the system administrator via **mail**(C). Messages with a module id of 0 are generated by the kernel.

Notes

Only one **strerr** process at a time is permitted to open the STREAMS log driver. If a module or driver is generating a large number of error messages, running the error logger will cause a degradation in STREAMS performance. If a large burst of messages are generated in a short time, the log driver may not be able to deliver some of the messages. This situation is indicated by gaps in the sequence numbering of the messages in the log files.

Files

/usr/adm/streams/error.*mm-dd*

See Also

log(M), and *STREAMS Programmer's Guide*

Name

sulogin - Special login program invoked by init (via /etc/inittab) to bring the machine up in single-user or multi-user mode.

Syntax

sulogin

Description

Sulogin prompts you for system maintenance (single-user) mode or multi-user mode.

If you select single-user mode by typing a valid root password, the system is brought up in system maintenance (single-user) mode by executing the shell script /etc/singleuser. If you select multiuser mode by typing Ctrl-d, or there is no reponse for 5 seconds, **sulogin** will execute the shell script file /etc/multiuser, which will bring the system up in multi-user mode.

Files

/etc/multiuser
/etc/singleuser
/etc/inittab

See Also

init(M)

Name

sysdef - Outputs system definition.

Syntax

/etc/sysdef [*system_namelist* [*master.d*]]

Description

Sysdef outputs the current system definition in tabular form. It lists all hardware devices, their local bus addresses, and unit count, as well as pseudo devices, system devices, loadable modules and the values of all tunable parameters. It generates the output by analyzing the named operating system file (*system_namelist*) and extracting the configuration information from the name list itself. The operating system file must be an "absolute" boot file (see **mkunix**(M)).

Files

/unix	Default operating system file (where the system namelist is)
/usr/sys/master.d/*	Default directory containing master files

See Also

mkunix(M), master(M), and nlist(S) in the *Reference (CP, S, F)*

Diagnostics

internal name list overflow
: if the master table contains more than an internally specified number of entries for use by **nlist**(S).

Name

term - Compiled term file.

Description

Compiled terminfo descriptions are placed under the directory /usr/lib/terminfo. To avoid a linear search of a huge system directory, a two-level scheme is used: /usr/lib/terminfo/*c*/*name* where *name* is the name of the terminal, and *c* is the first character of name. Thus, act4 can be found in the file /usr/lib/terminfo/a/act4. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An eight (or more) bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the terminfo compiler (**tic**(C)) program, and read by the routine **setupterm**(S). Both of these pieces of software are part of **curses**(S). The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The headers section begins the file. This section contains six short integers in the following format.

- The magic number (octal 0432).
- The size, in bytes, of the names section.
- The number of bytes in the boolean section.
- The number of short integers in the numbers section.
- The number of offsets (short integers) in strings section.
- The size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value -1 is represented by 0377,0377; other negative values are illegal. The -1 generally means that a capability is missing from this terminal. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the '|' character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. 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 flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is -1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as short integer, in the format above. A value of -1 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.

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Files

/usr/lib/terminfo/*/*	Compiled terminal capability data base

See Also

terminfo(M)

Name

termcap - Terminal capability database.

Description

The file /etc/termcap is a data base describing terminals. Terminals are described in **termcap** by a set of capabilities and how operations are performed. Padding requirements and initialization sequences are included in **termcap.** Note that the use of **term**(M) is preferred.

Entries in **termcap** consist of a number of ':' separated fields. The first entry for each terminal gives the names known for the terminal, separated by vertical bar (|) characters. The first name is always 2 characters long for compatibility with older systems. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

Capabilities

The following is a list of the capabilities that can be defined for a given terminal. In this list (P) indicates padding may be specified, and (P*) indicates that padding may be based on the number of lines affected.

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not ^H
BE	str		Bell character
bs	bool		Terminal can backspace with ^H
BS	str		Sent by BACKSPACE key (if not bc)
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column

Name	Type	Pad?	Description
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
CF	str		Cursor off
ch	str	(P)	Like cm but horizontal motion only, line stays same
CL	str		Sent by CHAR LEFT key
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
CN	str		Sent by CANCEL key
co	num		Number of columns in a line
CO	str		Sent by CHAR RIGHT key
cr	str	(P*)	Carriage return, (default ^M)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only
CW	str		Sent by CHANGE WINDOW key
da	bool		Display may be retained above
db	bool		Display may be retained below
dB	num		Number of millisec of bs delay
dC	num		Number of millisec of cr delay
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay
DK	str		Sent by down arrow key (if not kd)
DL	str		Sent by DELETE key
DL	str		Sent by destructive character delete key
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	number		Number of millisec of nl delay needed
do	str		Down one line
ed	str		End delete mode
EE	str		Edit mode end
EG	num		Number of chars taken by ES and EE
ei	str		End insert mode; give ':ei=:'
EN	str		Sent by END key
eo	str		Erase overstrikes with a blank
ES	str		Edit mode start
ff	str	(P*)	Hardcopy terminal page eject (default ^L)
G1	str		Upper-right (1st quadrant) corner character

Name	Type	Pad?	Description
G2	str		Upper-left (2nd quadrant) corner character
G3	str		Lower-left (3rd quadrant) corner character
G4	str		Lower-right (4th quadrant) corner character
GD	str		Down-tick character
GE	str		Graphics mode end
GG	num		Number of chars taken by GS and G
GH	str		Horizontal bar character
GS	str		Graphics mode start
GU	str		Up-tick character
GV	str		Vertical bar character
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed
hz	str		Hazeltine; can't print 's
ic	str	(P)	Insert character
if	str		Name of file containing is
im	bool		Insert mode (enter); give ':im=:q' if ic
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by 'other' function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of 'keypad transmit' mode
KF	str		Key-clock off
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of 'other' keys
KO	str		Key-clock on
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in 'keypad transmit' mode
dT	num		Number of millisec of tab delay needed
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on 'other' function keys
LD	str		Sent by line delete key

Name	Type	Pad?	Description
LF	str		Sent by line feed key
li	num		Number of lines on screen or page
LK	str		Sent by left arrow key (if not kl)
ll	str		Last line, first column (if no cm)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor
MN	str		Sent by minus sign key
MP	str		Multiplan initialization string
MR	str		Multiplan reset string
mu	str		Memory unlock (turn off memory lock)
nc	bool		No correctly working carriage return (DM25000,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default /n)
ns	bool		Terminal is a CRT but doesn't scroll
NU	str		Sent by NEXT UNLOCKED CELL key
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null
PD	str		Sent by PAGE DOWN key
PL	str		Sent by PAGE LEFT key
PR	str		Sent by PAGE RIGHT key
PS	str		Sent by plus sign key
pt	bool		Has hardware tabs (may need to be set with is)
PU	str		Sent by PAGE UP key
RC	str		Sent by RECALC key
RF	str		Sent by TOGGLE REFERENCE key
RK	str		Sent by right arrow key (if not kr)
RT	str		Sent by RETURN key
RT	str		Sent by return key
se	str		End stand out mode
sf	str	(P)	Scroll forward
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
TB	str		Sent by TAB key
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use cm

Name	Type	Pad?	Description
ti	str		String to begin programs that use cm
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by use or ue
UK	str		Sent by up arrow key (if not ku)
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
WR	str		Sent by WORD RIGHT key
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like ce /r /n (Delta Data)
xs	bool		Standard out not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in **termcap** are of three types:

- Boolean capabilities which indicate that the terminal has some particular feature.
- Numeric capabilities giving the size of the terminal or the size of particular delays
- String capabilities, which give a sequence which can be used to perform particular terminal operations.

The following entry describes the Altos II terminal.

```
a2|altos2|alt2|altos 2|Altos II:\
     :cd=\E[J:ce=\E[K:cl=\E:cl=\E[;H\E[2J:\
     :up=\E[1A:do=\E[1B:nd=\E[1C:bc=\E[1D:cm=\E[%i%d;%dH:ho=\E[H:\
     :al=\E[L:dl=\E[M:ic=\E[@:dc=\E[P:im=:ei=:\
     :co#80:li#24:ug#0:sg#0:bs:pt:sr:\
     :so=\E[7m:se=\E[m:us=\E[4m:ue=\E[m:\
     :is=\E>\E[?31\E[?41\E[?51\E[?7h\E[?8H:if=/usr/lib/tabset/vt100:\
     :ku=\E[A:kd=\E[B:kr=\E[C:kl=\E[D:kh=\E[f:kb=^H:cr=^M:\
     :XU=^Aq\r:XD=^Ar\r:XR=^As\r:XL=^At\r:\
     :YU=^AQ\r:YD=^AR\r:YR=^AS\r:YL=^AT\r:\
     :HL=^AP\r:\
     :IS=\E[@:DE=\E[P:IL=\E[L:DL=\E[M:NS=\E[S:PS=\E[T:\
     :LO=\E[0q:LC=\E[5q:LL=\E[6q:\
     :k0=^A@\r:k1=^AA\r:k2=^AB\r:k3=^AC\r:\
     :K4=^AD\r:k5=^AE\r:k6=^AF\r:k7=^AG\r:\
     :k8=^AH\r:k9=^AI\r:kA=^AJ\r:kB=^AK\r:\
     :kC=^AL\r:kD=^AM\r:kE=^AN\r:kF=^AO\r:\
     :c0=^A`\r:c1=^Aa\r:c2=^Ab\r:c3=^Ac\r:\
     :c4=^Ad\r:c5=^Ae\r:c6=^Af\r:c7=^Ag\r:\
     :c8=^Ah\r:c9=^Ai\r:cA=^Aj\r:cB=^Ak\r:\
     :cC=^Al\r:cD=^Am\r:cE=^An\r:cF=^Ao\r:
```

Type of Capabilities

All capabilities have two letter codes. 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 value. Thus **co**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either an integer, e.g., '20', or an integer followed by an '*', i.e., '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required.

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.

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there. A \E maps to an ESCAPE character, ^x maps to a control-x for any appropriate x, and the sequence **\n** **\r** **\t** **\b** **\f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a : in a capability it must be escaped in octal as **\072.** If it is necessary to place a null character in a string capability it must be encoded as **\200.** The routines that deal with **termcap** use C strings, and strip the high bits of the output very late so that a **\200** comes out as **\000** would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **termcap** and to build up a description gradually, using partial descriptions. Be aware that a very unusual terminal may expose deficiencies in the ability of the **termcap** file to describe it.

Basic Capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have an **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than ^H in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in **termcap** are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the **termcap** file usually assumes that this is on, i.e., **am**.

These capabilities suffice to describe hardcopy and 'glass-tty' terminals. Thus the model 33 teletype is described as:

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as:

```
cl|adm3|3|lsi adm3:am:bs:cl=^:li#24:co#80
```

Cursor Addressing

Cursor addressing in the terminal is described by a **cm** string capability, with **printf**(S)-like escapes (%x) in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the % encodings have the following meanings:

%d	as in **printf**, 0 origin
%2	like %2d
%3	like %3d
%	like %c
%+x	adds x to value, then %
%>xy	if value > x adds y, no output
%r	reverses order of line and column, no output
%i	increments lines/column (for 1 origin)
%%	gives a single %
%n	exclusive-or (xor) row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x mod 10), no output
%d	Reverse coding (x-2*(x mod 16)), no output (Delta Data)

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is 'cm=6\E&%r%2c%2Y'. The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, 'cm=^T%%'. Terminals which use '%' need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus 'cm=\E=%+ %+ '.

Cursor Motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly, a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

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 **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; 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 **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. 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 let the editor understand that deleting a line on the screen may bring non-blank lines up from below, or that scrolling back with **sb** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using **termcap**. The most common insert/delete character options affect only the characters on the current line and shift characters off the end of the line. 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 find out which 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'. If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. No known terminals have an insert mode not falling into one of these two classes.

The editor can handle both terminals that have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as **ei** the sequence to leave insert mode (give an empty value also if you gave **im** an empty value). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**; terminals that send a sequence to open a screen position should give it here. (Insert mode is preferable if a terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

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 **mi** because of the way its insert mode works.

Finally, you can specify delete mode by entering **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode, these can be given as **so** and **se** respectively. If there are several types of standout mode, (such as inverse video, blinking, or underlining), the preferred mode is inverse video by itself. 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, this is acceptable, and although it may confuse some programs slightly, it can't be helped.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of **ex**, this can be given as **vs** and **ve**, sent at the start and end of visual mode respectively. These can be used to change from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. 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.

If your terminal correctly generates underlined characters (with no special codes needed), even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

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 mode (this applies for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. 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 **kl**, **kr**, **ku**, **kd**, and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0**, **k1**, ..., **k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9.** If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the **termcap** 2 letter codes can be given in the **ko** capability, for example, ':ko=cl,ll,sf,sb:', which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2.0 of **vi**, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh.** It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl, j** for **kd, k** for **ku, l** for **kr,** and H for **kh.** For example, the mime would be :ma=^Kj^k^Xl: indicating arrow keys left (^H), down (^K), up (^), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc.**

If tabs on the terminal require padding, or if the terminal uses a character other than ^I to tab, then this can be given as **ta.**

Hazeltine terminals, which don't allow '~' characters to be printed should indicate **hz.** Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc.** Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn.** If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt.** Other specific terminal problems may be corrected by adding more capabilities of the form *xz*. Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is **/usr/lib/tabset/atd,** but **is** clears the tabs first.

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 **tc** can be given with the name of the similar terminal. This capability must be **last** and the combined length of the two entries must not exceed 1024. Since **termlib** routines search the entry from left to right, and since the **tc** capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with *xx@*, where *xx* is the capability. For example:

hh|2621nl:ks@:ke@:tc=2621:

This defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Files

/etc/termcap File containing terminal descriptions.

Related Commands

ex(C), tset(C), more(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

Use of **term**(M) is preferred.

Ex(C) allows only 256 characters for string capabilities, and the routines in **termcap** do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the **vi**(C) program.

Not all programs support all entries. There are entries that are not supported by any program.

Name

terminals - Supported terminals.

Description

The **/etc/termcap** file and the **/usr/lib/terminfo** directory contain two types of descriptions: terminals that have been tested and are supported by Altos, and terminals that are supplied for information only. The corresponding names can be used to assign the terminal type to TERM (see **environ**(M)).

If you wish to add a terminal from the "information only" section of one of the terminfo files, choose a description that closely resembles the terminal you are adding. Put the terminal description in a file, and edit it to suit your needs. Use **tic**(C) to compile the file by typing the following:

tic *filename*

Files

/etc/termcap
/usr/lib/terminfo/*/*

Name

terminfo - Terminal capability database.

Syntax

/usr/lib/terminfo/*/*

Description

Terminfo is a database describing terminals, used, for example, by **curses**(S). Terminals are described in **terminfo** by giving a set of capabilities that they have, and by describing how operations are performed. Padding requirements and initialization sequences are included.

Entries in **terminfo** consist of a number of comma-separated fields. White space after each comma (,) is ignored. The first entry for each terminal gives the names which are known for the terminal, separated by vertical bar (|) characters. The first name given is the most common abbreviation for the terminal, 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 be in lower case and contain no blanks, the last name may well contain upper case and 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, thus alt3 for the Altos III terminal. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, a vt100 in 132 column mode would be vt100-w. The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-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
-*n*p	Number of pages of memory	c100-4p
-rv	Reverse video	c100-rv

Capabilities

The variable is the name by which the programmer (at the terminfo level) accesses the capability. The capname is the short name used in the text of the database, and is used by a person updating the database. The i.code is the two letter internal code used in the compiled database, and always corresponds to the old **termcap**(M) capability name.

Capability names have no hard length limit, but an informal limit of five characters has been adopted to keep them short and to allow the tabs in the source file caps to line up nicely. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

(P) indicates that padding may be specified

(G) indicates that the string is passed through tparm with parms as given (*#i*)

(*) indicates that padding may be based on the number of lines affected

(*#i*) indicates the *i*th parameter

Variable	Cap-name	Termcap Code	Description
Booleans:			
auto_left_margin,	bw	bw	cubl wraps from column 0 to last column
auto_right_margin,	am	am	Terminal has automatic margins
beehive_glitch,	xsb	xb	Beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch,	xhp	xs	Standout (not erased by overwriting (hp)
eat_newline_glitch,	xenl	xn	newline ignored after 80 cols (Concept)
erase_overstrike,	eo	eo	Can erase overstrikes with a blank
generic_type,	gn	gn	Generic line type (e.g., dialup, switch)
hard_copy,	hc	hc	Hardcopy terminal
has_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 modes
move_standout_mode,	msgr	ms	Safe to move in standout modes
over_strike,	os	os	Terminal overstrikes
status_line_esc_ok,	eslok	es	Escape can be used on the status line
teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch,	hz	hz	Hazeltine; cannot print ~'s
transparent_underline,	ul	ul	Underline character overstrikes
xon_xoff,	xon	xo	Terminal uses xon/xoff handshaking
Numbers:			
column,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
lines,	lines	li	Number of lines on screen or page
lines_of_memory,	lm	lm	Lines of memory if > lines. 0 means varies
magic_cookie_glitch,	xmc	sg	Number of blank chars left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed
virtual_terminal,	vt	vt	Virtual terminal number (UNIX system)
width_status_line,	wsl	ws	No. columns in status line

Variable	Cap-name	Termcap Code	Description
Strings:			
acs_chars	acsc	ac	Graphic char set pairs aAbBcC - def = vt100+
back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	Change to lines #1 through #2 (vt100) (PG)
clear_all_tabs,	tbc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Term.settable cmd char in prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
cursor_down,	cud1	do	Down one line
cursor_home	home	ho	Home cursor (if no cup)
cursor_invisible,	civis	vi	Make cursor invisible
cursor_left,	cub1	le	Move cursor left one space
cursor_mem_address,	mrcup	CM	Memory relative cursor addressing
cursor_normal,	cnorm	ve	Make cursor appear normal (undo vs/vi)
cursor_right,	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll,	ll	ll	Last line, first column (if no cup)
cursor_up,	cuu1	up	Upline (cursor up)
cursor_visible,	cvvis	vs	Make cursor very visible
delete_character,	dch1	dc	Delete character (P*)
delete_line,	dl1	dl	Delete line (P*)
dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
enter_blink_mode,	blink	mb	Turn on blinking
enter_bold_mode,	bold	md	Turn on bold (extra bright) mode
enter_ca_mode,	smcup	ti	String to begin programs that use cup
enter_delete_mode,	smdc	dm	Delete mode (enter)
enter_dim_mode,	dim	mh	Turn on half-bright mode
enter_insert_mode,	smir	im	Insert mode (enter)
enter_protected_mode,	prot	mp	Turn on protected mode
enter_reverse_mode,	rev	mr	Turn on reverse video mode
enter_secure_mode,	invis	mk	Turn on blank mode (chars invisible)
enter_standout_mode,	smso	so	Begin stand out mode
enter_underline_mode,	smul	us	Start underscore mode

Variable	Cap-name	Termcap Code	Description
erase_chars	ech	ec	Erase #1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
exit_attribute_mode,	sgr0	me	Turn off all attributes
exit_ca_mode,	rmcup	te	String to end programs that use cup
exit_delete_mode,	rmdc	ed	End delete mode
exit_insert_mode,	rmir	ei	End insert mode
exit_standout_mode,	rmso	se	End stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ichl	ic	Insert character (P)
insert_line,	ill	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (P*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC	Sent by clear screen or erase key
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdchl	kD	Sent by delete character key
key_dl,	kdll	kL	Sent by delete line key
key_down,	kcudl	kd	Sent by terminal down arrow key
key_eic,	krmir	kM	Sent by rmir or smir in insert mode
key_eol,	kel	kE	Sent by clear-to-end-of-line key
key_eos,	ked	kS	Sent by clear-to-end-of-screen key
key_f0	kf0	k0	Sent by function key f0
key_f1,	kf1	k1	Sent by function key f1
key_f10,	kf10	ka	Sent by function key f10
key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9

Variable	Cap-name	Termcap Code	Description
key_home,	khome	kh	Sent by home key
key_ic,	kichl	kI	Sent by ins char/enter ins mode key
key_il,	kill	kA	Sent by insert line
key_left,	kcubl	kl	Sent by terminal left arrow key
key_ll,	kll	kH	Sent by home-down key
key npage,	knp	kN	Sent by next-page key
key_ppage,	kpp	kP	Sent by previous-page key
key_right,	kcufl	kr	Sent by terminal right arrow key
key_sf,	kind	kF	Sent by scroll-forward/down key
key_sr,	kri	kR	Sent by scroll-backward/up key
key_stab,	khts	kT	Sent by set-tab key
key_up,	kcuul	ku	Sent by terminal up arrow key
keypad_local,	rmkx	ke	Out of "keypad transmit" mode
keypad_xmit,	smkx	ks	Put terminal in "keypad transmit" mode
lab_f0,	lf0	l0	Labels on function key f0 if not f0
lab_f1,	lf1	l1	Labels on function key f1 if not f1
lab_f10,	lf10	la	Labels on function key f10 if not f10
lab_f2,	lf2	l2	Labels on function key f2 if not f2
lab_f3,	lf3	l3	Labels on function key f3 if not f3
lab_f4,	lf4	l4	Labels on function key f4 if not f4
lab_f5,	lf5	l5	Labels on function key f5 if not f5
lab_f6,	lf6	l6	Labels on function key f6 if not f6
lab_f7,	lf7	l7	Labels on function key f7 if not f7
lab_f8,	lf8	l8	Labels on function key f8 if not f8
lab_f9,	lf9	l9	Labels on function key f9 if not f9
meta_on,	smm	mm	Turn on "meta mode" (8th bit)
meta_off,	rmm	mo	Turn off "meta mode"
newline,	nel	nw	Newline (behaves like cr followed by lf)
pad_char,	pad	pc	Pad character (rather than null)
parm_dch,	dch	DC	Delete #1 chars (PG*)
parm_delete_line,	dl	DL	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC	Insert #1 blank chars (PG*)
parm_index,	indn	SF	Scroll forward #1 lines (PG)
parm_insert_line	il	AL	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP	Move cursor up #1 lines (PG*)

Variable	Cap-name	Termcap Code	Description
pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute string #2
pkey_xmit,	pfx	px	Prog funct key #1 to xmit string #2
print_screen,	mc0	ps	Print contents of the screen
prtr_off,	mc4	pf	Turn off the printer
prtr_on,	mc5	po	turn on the printer
repeat_char,	rep	rp	Repeat char #1 #2 times (PG*)
reset_1string,	rs1	r1	Reset terminal completely to sane modes
reset_2string,	rs2	r2	Reset terminal completely to sane modes
reset_3string,	rs3	r3	Reset terminal completely to sane modes
reset_file,	rf	rf	Name of file containing reset string
restor_cursor,	rc	rc	Restore cursor to position of last sc
row_address,	vpa	cv	Vertical position absolute (set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
set_tab,	hts	st	Set a tab in all rows, current column
set_window,	wind	wi	Current window is lines #1-#2 cols #3-#4
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column #1
underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1	Upper left of keypad
key_a3,	ka3	K3	Upper right of keypad
key_b2,	kb2	K2	Center of keypad
key_c1,	kc1	K4	Lower left of keypad
key_c3,	kc3	K5	Lower right of keypad
prtr_non	mc5p	p0	Turn on the printer for #1 bytes

A Sample Entry

The following is a complex example that describes a Concept-100.

```
concept100|c100|c104|c100-4p|concept 100,
     am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
     cols#80, cr=^M$<9>, cubl=^H, cudl=^J, cufl=\E=,
     cup=\Ea%pl%' '%+%c%p2%' '%+%c,
     cuul=\E;, cvvis=\EW, db, dchl=\E^A$<16*>, dim=\EE;, dll=E^B$<3*>,
     ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\EK$<20>\EK, ht=\t$<8>,
     ill=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
     is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
     kbs=^h, kcubl=\E>, kcudl=\E<, kcufl=\E=, kcuul=\E;,
     kfl=\E5, kf2=\E6, kf3=\E7, khome=\E?,
     lines#24, mir, pb#9600, prot=\EI, rep=\Er%pl%c%p2%' '%+%c$<.2*>,
     rev=\ED, rmcup=\EV  $<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
     rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\200,
     smcup=\EU\Ev 8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
     smul=\EG, tabs, ul, vt#8, xenl,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments may be included on lines beginning with a **#**.

Capabilities in **terminfo** are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence that can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the Concept-100 has automatic margins (i.e., an automatic return and linefeed when the end of a line is reached) which is indicated by the capability **am.** Numeric capabilities are followed by the character **#** and then the value. Thus, **cols,** which indicates the number of columns the terminal has, gives the value 80 for the Concept.

Finally, string valued capabilities, such as **el** (clear to end of line sequence) are given by the two-character code, an **=**, and then a string ending at the next following **,**. A delay in milliseconds may appear anywhere in such a capability, enclosed in $<..> brackets, as in el=\EK$<3>, and padding characters are supplied by tputs to provide this delay. The delay can be either a number, e.g., **20**, or a number followed by an *, i.e., **3***. A * indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the peraffected-unit padding required. (In the case of insert character, the factor is still the number of lines affected. This is always one unless the terminal has xenl and the software uses it.) When a * is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string valued capabilities for each encoding of characters there. Both \E and \e map to an ESCAPE character, ^x maps to a control-x for any appropriate x, and the sequences \n \l \r \t \b \f \s give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include \^ for ^, \\ for \, \, for comma, \: for :, and \0 for null. (\0 will produce \200, which does not terminate a string but behaves as a null character on most terminals.)

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name.

Preparing Terminal 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**(C) to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **terminfo** file to describe it or bugs in **vi**. To easily test a new terminal description you can set the environment variable TERMINFO to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in /usr/lib/terminfo. To get the padding for insert line right (if the terminal manufacturer did not document it) a severe test is to edit a test file at 9600 baud, delete 16 or so lines (i.e., **d16d**) from the middle of the screen, then press the **u** key several times quickly. If the terminal messes up, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as TEKTRONIX 4010 series, as well as hardcopy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, control M.) If there is a code to produce an audible signal (bell, beep, etc.) give this as **bel**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over, for example, you would not normally use '**cuf1=**' because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example.

If the terminal has switch selectable automatic margins, the **terminfo** file usually assumes that this is on; i.e., **am.** If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the model 33 teletype is described as:

```
33|tty33|model 33 teletype,
bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM-3 is described as:

```
adm3|3|lsi adm3,
am, bel=^G, clear=^, 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 **prinf**(S) like escapes **%x** in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup.**

The parameter mechanism uses a stack and special % codes to manipulate it. Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary.

The % encodings have the following meanings:

```
%%              outputs '%'
%d              print pop() as in printf
%2d             print pop() like %2d
%3d             print pop() like %3d
%02d
%03d            as in printf
%c              print pop() as %c
%s              print pop() as %s

%p[1-9]         push ith parm
%P[a-z]         set variable [a-z] to pop()
%g[a-z]         get variable [a-z] and push it
%'c'            char constant c
%{nn}           integer constant nn

%+%-%*%/%m
arithmetic      (%m is mod): push(pop() op pop())
%&%|%^          bit operations:  push(pop() op pop())
%=%>%<          logical operations:  push(pop() op
                 pop())
%!%~            unary operations push(op pop())
%i              add 1 to first two parms (for ANSI
                 terminals)

%? expr %t thenpart %e elsepart %;
                if-then-else, %e elsepart is optional.
                else-if's are possible ala Algol 68:
                %? c1 %tb1 %ec2 %tb2 %ec3 %tb3 %ec4
                 %tb4 %e%;
                ci are conditions, bi are bodies.
```

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use %gx%{5}%-.

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E%&a12c03Y padded for six milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is cup=6\E&%p2%2dc%p1%2dY.

The Microterm ACT-IV needs the current row and column sent preceded by a ^T, with the row and column simply encoded in binary, cup=^T%p1%c%p2%c. Terminals which use %c need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n** ^D and **\r**, as the system may change or discard them. (The library routines dealing with terminfo set tty modes so that tabs are never expanded, so \t is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus cup=\E=%p1%' '%+%c%p2%' '%+%c. After sending '\E=', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

If the terminal has row or column absolute cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two parameter sequence (as with the hp2645) and can be used in preference to **cup.** If there are parameterized local motions (e.g., move n spaces to the right) these can be given as **cud, cub, cuf,** and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the TEKTRONIX 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to the very upper left corner of screen) then this can be given as **home;** similarly a fast way of getting to the lower left-hand corner can be given as **ll;** this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the \EH sequence on HP terminals cannot be used for **home.**)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **Ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed**(C) is not available.)

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**. If the terminal has a settable scrolling region (like the vt100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command -- the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using **terminfo.** The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks.

You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type **abc def** using local cursor motions (not spaces) between the **abc** and the **def.** Then position the cursor before the **abc** and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the **abc** shifts over to the **def** which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in,** which stands for insert null. While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces), no known terminals have an insert mode that 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 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 **ichl** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ichl**; 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 **ichl.** Do not give both unless the terminal actually requires both to be used in combination.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option).

Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an insert mode and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as standout mode, representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **smul** and **armul** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include:

blink	blinking
bold	bold or extra bright
dim	dim or half-bright
invis	blanking or invisible text
prot	protected
rev	reverse video
sgr0	turn off all attribute modes
smacs	enter alternate character set mode
rmacs	exit alternate character set mode

Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 9 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.

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 HP2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **flash**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis.** If there is a way to make the cursor completely invisible, give that as **cvis.** The capability **cnorm** should be given, which undoes the effects of both of these models.

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 the one used by **terminfo.**

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul.** If overstrikes are erasable with a blank, then indicate this by giving **eo.**

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 unshifed HP2621 keys). If the keypad can be set to transmit or not to 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, ..., f10, the codes they send can be given as **kf0, kf1, ..., kf10.** If these 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), **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.

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 (move left 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 as spaces the tabs are set to. This is normally used by the **tset**(C) command to determine whether to set the mode for hardwar 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.

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 are normally sent to the terminal by the **tset**(C) program each time the user logs in. They will be printed in the following order: **is1; is2;** setting tabs using **tbc** and **hts; if;** running the program **iprog;** and finally **is3.** 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.** A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1, rs2,** and **rs3,** analogous to **is2** and **if.** These strings are output by the **reset** program, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs2** and **rf** only if they produced annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of **is2**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80 column mode.

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

Delays

Certain capabilities control padding in the teletype driver. These are primarily needed by hardcopy terminals, and are used by the **tset** program to set teletype modes appropriately. Delays embedded in the capabilities **cr, ind, cub1, ff,** and **tab** will cause the appropriate delay bits to be set in the teletype driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb.**

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 modifications from the AT&T 4410v1 terminal. These characters and their corresponding glyphs are shown in the following table:

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 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. An example is on the following page:

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	.

Specify the characters defining the new tty character set in a left-to-right order, as shown in the following example (taken from the example above):

```
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 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 to go to the beginning of the status line and to 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**.

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: The environment is to be searched for a **CC** variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

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 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.

If the terminal has a "meta key" that acts as a shift key, setting the eighth bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the eighth 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.

If the terminal is one of those supported by the UNIX virtual terminal protocol, the terminal number can be given as **vt**. 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. It is undefined whether the text is also displayed on the terminal screen when the printer is on. 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. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. 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; **floc** causes the string to be executed by the terminal in local; and **pfx** causes the string to be transmitted to the computer.

Glitches and Braindamage

Hazeltine terminals, which do not allow ~~ characters to be displayed should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept and vt100, should indicate **xenl**.

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

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

The Beehive Superbee, which is unable to correctly transmit the escape or Control-C characters, has **xsb**, indicating that the f1 key is used for escape and f2 for control-C. (Only certain Superbees have this problem, depending on the ROM.) Other specific terminal problems may be corrected by adding more capabilities of the form *xx*.

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 cancelled by placing **xx@** to the left of the capability definition, where xx is the capability. For example, the entry

```
2621-nl, smkx@, rmkx@, use=2621,
```

defines a 2621-nl that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Files

/usr/lib/terminfo/?/*	Files containing terminal descriptions
/usr/lib/terminfo/altos.src	File containing descriptions of terminals supported by Altos
/usr/lib/terminfo/terminfo.src	File containing descriptions of other terminals not supported by Altos

See Also

term(M)

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 these files; they are opened by **getty**(M) 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**(S). A process can break this association by changing its process group using **setpgrp**(S).

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

Normally, terminal input is processed in units of lines. A line is delimited by a newline (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 keystroke 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**(S).

QUIT (Control-\ or ASCII FS) generates a quit signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called core) will be created in the current working directory.

ERASE (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, 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 cannot be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

STOP (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters cannot be changed or escaped.

The character values for INTR, QUIT, SWITCH, 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**(S) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#define NCC      8
struct  termio {
        unsigned  short  c_iflag;   /*input modes*/
        unsigned  short  c_oflag;   /*output modes*/
        unsigned  short  c_cflag;   /*control modes*/
        unsigned  short  c_lflag;   /*local modes*/
        char             c_line;    /*line discipline*/
        unsigned  char   c_cc[NCC]; /*control chars*/
 };
```

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

0	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 uppercase to lowercase 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 disable. 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 uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a receive(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 cha acters 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 lowercase 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	Selector 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 verical-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 lowercase alphabetic character is transmitted as the corresponding uppercase 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 dealy. 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 two 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	134.5 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
B38400	0000017	38400 baud (not supported)
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		Block layer output 0040000

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the dataterminal-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 stop 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 operating system disconnects ("hangs up") the line when the last process (or file) for that port closes. That is, the dataterminal-ready signal becomes false (is not asserted). If HUPCL is false, the data-terminal-ready signal remains true even after the last close on the port.

If CLOCAL is set when a port is opened, the operating system assumes the line is a local, directly connected port (i.e., there is no modem control) and the open completes without waiting for carrier. The dataterminal-ready and request-to-send signals are asserted, and incoming modem signals are ignored. If CLOCAL is false for a port when opening it, the operating system assumes there is modem control, and the open waits for the carrier-detect to be true (if the O_NDELAY flag is not set on the file (see fcntl.h)). The data-terminal-ready and request-to-send signals are asserted.

The operating system also checks CLOCAL when a modem in terrupt occurs, usually when the data-terminal-ready signal changes. The operating system assumes that the data-terminal-ready signal reflects the carrier sense of the modem and will kill the process group for the port if data-terminal-ready goes from true to false. If data-terminal-ready goes from false to true, the operating system wakes up any open requests waiting for carrier-detect to go true. If CLOCAL is true, the operating system disables modem interrupts.

Finally, CLOCAL also affects hardware flow control. If CLOCAL is false, the operating system does not enable any hardware flow control, regardless of the setting of hardware flow control (see SETFLOW below) flags.

The initial hardware control value after open is B9600, CS8, CREAD, HUPCL for modem ports, and B9600, CS8, CREAD, CLOCAL for local ports.

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/ower 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. 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 uppercase 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, 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 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**(S) system calls have the form:

> **ioctl** (*filedes*, *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.

Another group of **ioctl** system calls have the form:

ioctl (*filedes, command, arg*)
char **arg*;

The commands using this form are:

SETFLOW
: Sets the hardware flow control bits, defined as TXHARD and RXHARD for the terminal. If the TXHARD bit is set, hardware output flow control is enabled. If the RXHARD bit is set, hardware input flow control is enabled. The argument is a pointer to a byte with these bits set (or not). The software flow control bits (TXSOFT and RXSOFT) are ignored.

GETFLOW
: Returns the hardware flow control bits. The argument is a pointer to a byte with these bits set (or not).

Additional **ioct** calls have the form:

ioctl (*filedes, 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.

SETMODEM
: Sets the modem mode to USER, ON, or OFF for the terminal. *Arg* should be either MDM_ON, MDM_OFF, or MDM_USER.

GETMODEM
: Returns the current modem setting, either MDM_ON, MDM_OFF, MDM_USER. Arg is ignored.

See **ioctl**(S) for details on how to use this system call.

Files

/dev/tty
/dev/tty*
/dev/console

See Also

ioctl(S), stty(C), xtty(C)

Name

timezone - Sets default system time zone.

Syntax

/etc/TIMEZONE

Description

This file sets and exports the time zone environmental variable TZ. This file is included into other files that must know the time zone.

Examples

/etc/TIMEZONE for the East coast:

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

See Also

rc2(M), profile(M), and ctime(S) in the *Reference (CP, S, F)*

Name

ttys - Login terminals file.

Syntax

/etc/ttys

Description

The /etc/ttys file contains a list of the device special files associated with possible login terminals.

The file contains one or more entries of the form:

state mode name

The *name* must be the filename of a device special file. Only the filename may be supplied, the path is assumed to be /dev. If *state* is "1", the device is enabled for logins; if "0", the device is disabled. The *mode* is used as an argument to the **getty** program. It defines the line speed and type of device associated with the terminal. A list of arguments is provided in **getty.**

For example, the entry "16tty02" means the serial line tty02 is to be enabled for logging in at 9600 baud.

Files

/etc/ttys

See Also

getty(M), pconfig(C)

Notes

Edit the /etc/ttys file only when in system maintenance mode. This file is obsolete, and is maintained only for the convenience of old programs. **Init**(M) no longer examines this file.

Name

utmp, wtmp - Utmp and wtmp entry formats.

Syntax

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

Description

These files, which hold user and accounting information for such commands as **who**(C), **write**(C), and **login**(M), 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 */
};

/* Definitioins 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 accounting */
/* 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(S), login(C), who(C), write(C)

Name

uucheck - Checks the uucp directories and permissions file.

Syntax

/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_level* 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(M), uusched(M), uucp(C), uustat(C), uux(C)

Notes

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

Name

uucico - File transport program for the uucp system.

Syntax

/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** option 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**(C). **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 for debugging. For example, the script **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(C), uusched(M), uutry(M), uucp(C), uustat(C), uux(C)

Name

uucleanup - Uucp spool directory cleanup.

Syntax

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

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 cannot 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 (default 1 day). The message includes the *JOBID*, and in the case of mail, telling whom to call to check the problem (**-m** option).

-X*time* Any **X.** files greater or equal to *time* days old will be removed (default 2 days). 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.

-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**(C).

Files

/usr/lib/uucp Directory with commands used by **uucleanup** internally

/usr/spool/uucp Spool directory

See Also

cron(C), uucp(C), uux(C)

Name

uugetty - Sets terminal type, modes, speed, and line discipline.

Syntax

/usr/lib/uucp/uugetty [**-h**] [**-t** *timeout*] [**-r**] *line* [*speed* [*type* [*linedisc*]]]
/usr/lib/uucp/uugetty -c *file*

Description

Uugetty is identical to **getty**(M) but changes have been made to support using the line for **uucico, cu,** and **ct;** that is, the line can be used in both directions. **Uugetty** will allow users to log in, but if the line is free, **uucico, cu,** or **ct** can use it for dialing out. The implementation depends on the fact that **uucico, cu,** and **ct** create lock files when devices are used. When the **open**(S) returns (or the first character is read when **-r** option is used), the status of the lock file indicates whether the line is being used by **uucico, cu, ct,** or someone trying to log in. Note that in the **-r** case, several carriage-return characters may be required before the login message is output. The users will be able to handle this slight inconvenience. **Uucico** trying to log in will have to be told by using the following login script:

    "" \r\d\r\d\r\d\r in:--in:...

where the **...** is whatever would normally be used for the login sequence.

An entry for an intelligent modem or direct line that has a **uugetty** on each end must use the **-r** option. (This causes **uugetty** to wait to read a character before it puts out the login message, thus preventing two **uugettys** from looping.) If there is a **uugetty** on one end of a direct line, there must be a **uugetty** on the other end as well. Here is an **/etc/inittab** entry using **uugetty** on an intelligent modem or direct line:

tt12:2:respawn:env - TERM=altos5
/usr/lib/uucp/uugetty -r -t 60 tty12 1200

For an explanation of **uugetty** options, see **getty**(M).

Files

/etc/gettydefs
/etc/issue

See Also

uucico(M), getty(M), init(M), tty(M), cu(C), login(M) gettydefs(M), inittab(M), and ioctl(S) in the *Reference (CP, S, F)*

Notes

Uugetty does not support linking of device files.

Name

uusched - Scheduler for the uucp file transport program.

Syntax

/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**(C) 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(C), uucico(M), uucp(C), uustat(C), uux(C)

Name

Uutry - Tries to contact remote system with debugging on.

Syntax

/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(M), uucp(C), uux(C)

Name

uuxqt - Executes remote command requests.

Syntax

/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(M), uucp(C), uustat(C), uux(C), mail(C)

Name

volcopy, labelit - Copies file systems with label checking.

Syntax

/etc/**volcopy** [*options*] *fsname special1 volname1 special2 volname2*
/etc/**labelit** *special* [*fsname volume* [-*n*]]

Description

The **volcopy** command makes a literal copy of the file system using a blocksize matched to the device.

The **labelit** command creates a label for an unmounted disk file system or a **volcopy** archive device. The **-n** option provides for initial labeling on tapes only (this destroys previous contents). Otherwise, a label must already exist and only the *fsname* and *volume* arguments are modified. If all optional arguments are omitted, **labelit** prints the current label values of the special device.

Options

-a Invokes a verification sequence requiring a positive operator response instead of the standard 10-second delay before the copy is made.

-s Prompts the user before the copy is made. The copy is aborted if the user presses Break/Del within 10 seconds (default).

-y Assumes a "yes" response to all questions.

The following additional options are used only with tapes:

-reel*num* Specifies the beginning reel number for a restarted copy.

-buf Uses double-buffered I/O.

-feet*num*	Specifies the tape length, only valid when using reel tape.
-bpi*num*	Specifies the tape density (bits/inch), only valid when using reel tape.
-tr	Specifies reel tape.
-tc	Specifies cartridge tape.
-scsi	Assumes tape drive is of scsi type, only valid when using cartridge tape.
-nonscsi	Assumes tape drive is not scsi type, only valid when using cartridge tape.
-type*LABEL*	Specifies the type of cartridge tape being used, only valid when using cartridge tape.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. Reel or cartridge tapes may be used. If the file system is too large to fit on one reel, **volcopy** will prompt for additional reels. Labels of all reels are checked.

If **volcopy** is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations, such as **labelit**, and return to **volcopy** by exiting the new shell.

The *fsname* argument represents the mounted name (for example, root or usr) of the file system being copied. The *special* argument should be the physical disk section or tape, for example, /dev/rhd0b or /dev/rct.

The *volname* argument is the physical volume name (for example, rhd0b), and should match the external label sticker. Such label names are limited to six or fewer characters. To use the existing volume name, specify -- for the *volname* argument.

The arguments *special1* and *volname1* are the device and volume from which the copy of the file system is being extracted. The arguments *special2* and *volname2* are the target device and volume.

Neither the source or target device should have a file system mounted while running **volcopy**, or while creating a label with **labelit**. The exception is for the / file system, where you should be in single-user mode. (You can read the label of a mounted file system with **labelit**.)

The values for *fsname* and *volname* are recorded in the last 12 characters of the superblock (char fsname[6], volname[6];).

Examples

To label a tape for the / file system, with volume label v001, go to single-user mode and enter:

/etc/labelit /dev/rct / v001

To archive the / file system on a tape, labeled as in the above example, enter:

/etc/volcopy / /dev/rhd0b hd0b /dev/rct v001

To restore a tape (archived as above) of the / file system to disk, enter:

/etc/volcopy / /dev/rct v001 /dev/rhd0b hd0b

Note that when using **volcopy** for the / file system, go to single-user mode.

Files

/etc/log/filesave.log	Record of file systems/volume copied

See Also

sh(C)

Notes

Only device names beginning with /dev/rct are treated as tapes.

Name

vt - Virtual terminal management (Series 500 only).

Description

The virtual terminal (VT) device driver is a layer of management functions that provides the facilities to support and switch between up to eight screen faces on each physical device. Terminal or display device drivers that have been written to take advantage of this facility can therefore present multiple VTs on a single physical device. The correspondence between physical and virtual terminals is determined using the minor device number of the physical device, with the bottom five bits selecting the physical device and the top three bits selecting the virtual terminal.

Virtual terminals are accessed in exactly the same way as any other device. The **open**(S) system call is used to open the virtual terminal, and **read**(S), **write**(S), and **ioctl**(S) are used in the normal way and support all the functionality of the underlying device. In addition, some VT-specific **ioctl** calls are provided as described below.

Virtual terminals provide the link between different screen faces and the device. The virtual terminal that corresponds to the currently visible screen face is called the active virtual terminal. The active VT is the one that input from the device will be directed to, and any device-specific modes that can change on a per-VT basis will be set to the characteristics associated with the active VT.

Open virtual terminals on a device are placed on a "ring," with the active VT always being the VT on the top of the ring. The ring can be cycled through via a "hot key" that is specific to the underlying device driver. The first open of a VT causes it to be placed at the top of the ring and become the active VT. The last close on a VT causes it to be removed from the ring, and if this was the active VT, the previous VT on the ring becomes the active one.

Virtual terminal switching can be done in two different modes: automatically on receipt of a hot key, or under control of the process owning the VT. In the first case, the process associated with the VT knows nothing about the switch and it is handled entirely by the underlying device driver and the virtual terminal manager. In process-controlled switch mode, when a switch hot key is sent, the process owning the VT is sent a signal (see **sigset**(S)) that it has specified to the VT manager. This signal requests the process to release the physical device. The VT manager then awaits an **ioctl** from the process indicating that the VT either has released the physical device (in which case a switch occurs), or refuses to release the device (in which case the switch does not occur). If a predefined time limit expires before the **ioctl** is received from the process owning the VT, the VT manager behaves as if an **ioctl** indicating refusal was received. The ring of active VTs can contain intermixed auto mode and process control mode VTs. Process control mode VTs will be sent a signal that they have specified when they become the active VT. Some device drivers may support a forced switch mode, in which case an alternate hotkey sequence will cause the driver to force a switch to the next VT even if a normal switch is refused. The driver does the forced switch and the VT manager signals the VT that it has been forced out.

Ioctl Calls

The following **ioctl** calls apply to any device that supports VTs.

VT_OPENQRY
: This call is used to find an available VT. The argument to the **ioctl** is a pointer to a long. The long will be filled in with the number of the first available VT that no other process has open (this may be the one currently opened). If there are no available VTs then -1 will be filled in.

VT_GETMODE
: This call is used to determine what mode the VT is currently in, either VT_AUTO or VT_PROCESS. The argument to the **ioctl** is the address of the following structure, as defined in **<sys/vt.h>**.

```
struct vt_mode {
    char     mode;      /* VT mode */
    char     waitv;     /* if non-zero, hang on writes when
                        not active */
    short    relsig;    /* signal to use for release request */
    short    acqsig;    /* signal to use for display acquired */
    short    frsig;     /* signal to use for forced release */
}
/* Virtual Terminal Modes */
#define      VT_AUTO        0/* automatic VT switching */
#define      VT_PROCESS     1/* process controls switching */
```

The structure will be filled in with the current value for each field.

VT_SETMODE
: This call is used to set the VT mode. The argument to the **ioctl** is a pointer to a vt_mode structure, as defined above. The structure should be filled in with the desired VT mode and whether or not to block on writes when not active. If process-control mode is specified then the signals that should be used to communicate with the process should be specified. If any of the signals are not specified (value is zero), then the default for that signal will be used (SIGUSR1 for *relsig* and *acqsig* and SIGUSR2 for *frsig*).

VT_RELDISP
: This call is used to tell the VT manager if the display has been released or if the process has refused to release the display. A non-zero argument signals release and zero indicates refusal to release.

VT_ACTIVATE
: This call has the effect of making the VT specified in the argument the active VT. The VT manager will cause a switch to occur in the same manner as if a hotkey had initiated the switch. If the specified VT is not open or does not exist, the call will fail and *errno* will be set to ENXIO.

Files

/dev/vtxxn

See Also

ioctl(S), sighold(S), signal(S), sigrelse(S), sigset(S)

Warnings

There is a potential for a race condition on a heavily loaded system. When a process-control mode VT is sent the release requested signal, it is possible that it may not reply with a release **ioctl** before the internal timer expires and refusal to switch is assumed. The switch request will then be canceled and the VT will not switch screen faces. This can be detected by the process attempting to release the display. If the release **ioctl** fails and *errno* is EINVAL, then the releasing process can assume that the switch request was canceled.

Name

whodo - Shows who is doing what.

Syntax

/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(C), who(C)

Name

xpd - Transparent printer daemon.

Syntax

xpd *tty lp type*

Description

The **xpd** daemon directs any output sent to the *lp* device to the printer attached to the *tty* device printer port.

tty	is the name of the terminal device to which the printer is attached. It must be invoked as /dev/*tty*.
lp	is the name of a FIFO special device to be used by the printer. It must be invoked as /dev/*lp*.
type	is the name of the terminal type. The altos2, altos3, altos4, altos5, and Wyse 30 terminals are supported.

Files

/dev/tty??
/dev/lp?

Also See

mknod(C)

hange Information

This is a summary of the changes that have been made to the previous version of this manual. The chapters, page numbers, and/or paragraphs mentioned in this summary reference the previous manual.

tle: Altos System V Series 386 Reference (M)

vised Part Number: 690-22870-002

evious Part Number: 690-22870-001

te: June 1989

anges:

Updated the Permuted Index and Table of Contents.

Added **aliases**(M) and **aliashash**(M).

Changed **rc5**(M) to **rc0**(M).

Changed the following pages:

Page	Command	Description
8	**crash**(M)	Corrected **pagemode** on/off toggle option to read **-on** or **-off.**
4	**init**(M)	Run level 0 is now used for shutdowns (formerly run level 5).
7	**keyboard**(M)	Modified Keyboard Keys table and added Other Keys table.
4, 20	**terminfo**(M)	Added information about line graphics character set mapping with **acsc.**

Page	Command	Description
26	**terminfo**(M)	Added two more files containing terminal descriptions: **altos.src** and **terminfo.src.**
3	**volcopy**(M)	Removed references to backing up root file system with **volcopy**(M).

READER'S COMMENTS

Manual Title: Altos System V Series 386 Reference (M)

Part Number: 690-22870-002

Altos Computer Systems' Publications Department wants to provide documents that meet the needs of all our customers. Your comments help us produce better manuals.

Please Rate This Manual:	Excellent	Good	Average	Fair	Poor
Completeness of information	☐	☐	☐	☐	☐
Organization of manual	☐	☐	☐	☐	☐
Adequate illustrations	☐	☐	☐	☐	☐
Overall manual	☐	☐	☐	☐	☐

Do you find any of the chapters confusing or difficult to use? If so, which ones and why?

____

____

What could we do to improve the manual for you?

____

____

If you find errors or other problems when using this manual, please write them below. Do include page numbers or section titles.

____

____

Name: ______________________ Title: ______________________

Company: ______________________ Type of system: ____________

Phone: (_____) __________ - ____________ ext. ____________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

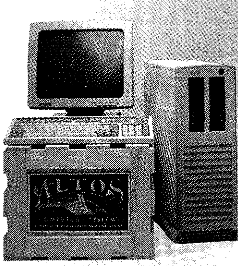
FIRST CLASS PERMIT NO. 7399 SAN JOSE, CA 95134

POSTAGE WILL BE PAID BY ADDRESSEE

Altos Computer Systems
ATTN: PUBLICATIONS DEPARTMENT
2641 Orchard Parkway
San Jose, CA 95134-9987
USA

Fold Here

P/N 690-22870-002
Printed in U.S.A.
9/89



Altos Computer Systems
2641 Orchard Parkway, San Jose, CA 95134
408/946-6700, FAX 408/433-9335