NAME

dir − directory layout

SYNOPSIS

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

DESCRIPTION

The directories of the V1 and V2 file systems are arrays of the following structure defined in <sys/dir.h>:

```
struct direct {
        ino_t           d_ino;          /* I-node number */
        char            d_name[14];     /* Name of up to 14 characters */
};
```

The **d_ino** field is the inode number of the file named by **d_name**. **D_ino** is zero if the directory slot isn't allocated. This number is the same as **st_ino** returned by **stat**(2) unless the entry is mounted on. **D_name** is the name of up to 14 characters, null-terminated only if less then 14 in length. Any character other than null or '/' is allowed.

See **directory**(3) for a portable way to access directories, Minix is probably the last system with these old V7 format directories.

SEE ALSO

directory(3).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

ethers − ethernet address to hostname database

SYNOPSIS

/etc/ethers

DESCRIPTION

The ethers database translates ethernet addresses to hostnames for use by the RARP daemon (see **rarpd**(8).) **/etc/ethers** may look like this:

```
0:0:c0:a:77:23      darask
0:0:c0:a:68:ce      burask
```

The six octet ethernet numbers must be entered as shown above, the hex constants must use lowercase letters and may not have leading zeros. Comments are marked with '#'.

See **rarpd**(8) on why you shouldn't list Sun hosts in this file.

FILES

/etc/ethers Ethernet addresses database.

SEE ALSO

hosts(5), **rarpd**(8), **boot**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

fstab, mtab − list of file systems to mount, mounted file system table.

SYNOPSIS

/etc/fstab

/etc/mtab

DESCRIPTION

/etc/fstab is a table of file system to mount at boot time, **/etc/mtab** is a table of currently mounted file systems as maintained by **mount** and **umount**.

/etc/fstab is not read by **mount** as it should be. It is instead a simple shell script listing the three devices that Minix needs to operate: The device names of the root file system, the temporary (scratch) file system, and the file system for **/usr**. Of these only the /usr file system is mounted in **/etc/rc**, the scratch file system is there for the system administrator to test new kernels, or as a temporary file system.

/etc/mtab contains lines of four fields. The layout is:

device **directory** **type** **options**

These fields may be explained as follows:

device
A block special device.

directory
Mount point.

type
Either **1**, or **2**, indicating a V1 or V2 file system.

options
Either **ro**, or **rw**, indicating a read-only or read-write mounted file system.

FILES

/etc/fstab Shell script naming three important file systems.

/etc/mtab List of mounted file systems.

SEE ALSO

printroot(8), **mount**(1), **fsck**(1), **mkfs**(1).

BUGS

/etc/fstab is a joke.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

hosts − hostname to IP address database

SYNOPSIS

/etc/hosts

DESCRIPTION

The hosts database lists the IP addresses and the hostnames that translate to these IP addresses. It is used by **nonamed**(8) in a network without name servers. A simple **/etc/hosts** may look like this:

```
127.0.0.1       localhost
192.9.200.1     darask
192.9.200.2     burask
```

The localhost entry lists a special address that refers to the local host itself (a kind of /dev/tty for hosts.) You should only list it if **nonamed** needs it! The other entries are actual machines. The file may contain comments marked with '#'.

You can have aliases (more hostnames on the same line), but it is not recommended, because **nonamed** can't present them to the system as CNAME records. An often seen form like

```
192.9.200.1     darask.home.cs.vu.nl    darask
```

is harmless though, and has the small advantage that you can use the short name in **/etc/ethers** so **rarpd** can match it at boot time.

FILES

/etc/hosts Hosts database.

SEE ALSO

ethers(5), **nonamed**(8), **rarpd**(8), **boot**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

keymap − keyboard maps

SYNOPSIS

/etc/keymap

DESCRIPTION

/etc/keymap is the compressed mapping from keyboard scan codes to ASCII. It is made from a keymap source file consisting of MAP_COLS columns (MINIX assigns the value 6 to MAX_COLS, corresponding to key pressed, key+SHIFT, key+LEFT_ALT, key+RIGHT_ALT, key+ALT+SHIFT and key+CTRL) and NR_SCAN_CODES rows (MINIX assigns the value 0x80 to NR_SCAN_CODES, corresponding to the number of scan codes to be provided by the keyboard), and each element is 2 bytes in length (see u16_t in type definitions). The low order byte corresponds to the character represented by the scan code, and the high order byte corresponds to the special meaning (when CAPS LOCK has effect, if it is a function key, etc.), which is converted to binary keymap format using the **genmap** utility.

Types (general): <sys/types.h>

<sys/types.h> defines the **u8_t** and **u16_t** types, corresponding to 8 and 16 bit values.

Macros: <minix/keymap.h>

C(*c*) - Control
Maps to control code

A(*c*) - Alt
Sets the eight bit

CA(*c*) - Control-Alt
Short for **A(C(*c*))**

L(*c*) - Caps Lock
Adds Caps Lock effect

These macros are used in a keymap source file to help define keys. So instead of writing **032** to put a CTRL-Z in the map you write **C('Z')**. The **L(*c*)** macro is used in column 0 to tell that the Caps Lock key is active for this key. (Caps Lock should only have effect on letters.)

Definitions: <minix/keymap.h>

<minix/keymap.h> contains a large number of definitions for special keys, like function keys, and keys on the numeric keypad. They are:

Escape key and modifiers: **EXT**, **CTRL**, **SHIFT**, **ALT**.

Numeric keypad keys: **HOME**, **END**, **UP**, **DOWN**, **LEFT**, **RIGHT**, **PGUP**, **PGDN**, **MID** (numeric '5'), **PLUS**, **INSRT**.

ALT + numpad key: **AHOME**, **AEND**, ..., **AINSRT**.

CTRL + numpad: **CHOME**, **CEND**, ..., **CINSRT**.

Lock keys: **CALOCK** (Caps Lock), **NLOCK** (Num Lock), **SLOCK** (Scroll Lock).

Function keys: **F1**, ..., **F12**.

ALT - function key: **AF1**, ..., **AF12**.

CTRL - function key: **CF1**, ..., **CF12**.

SHIFT - function key: **SF1**, ..., **SF12**.

ALT - SHIFT - function key: **ASF1**, ..., **ASF12**.

There is one key definition that isn't a key at all: **EXTKEY**. This keycode is sent by the keyboard as an indicator that the next keycode is special. For instance both ALT keys have the same keycode, but the right ALT key is sent by the keyboard preceded by the EXTKEY keycode. The same is true for the '/' key on the numeric pad versus the other '/' key on the US keyboard. (On other keyboards this key may have a different symbol.) The keyboard driver knows that a different key is presses if it is preceded by EXTKEY.

Creating/changing keyboard mapping

You can create your own keyboard mapping by copying one of the existing keymap source files (Standard Minix: **kernel/keymaps/*.src**, Minix-vmd: **kernel/ibm/keymaps/*.src**) and modifying the desired keys. Once this has been done, you need to recompile the genmap.c file, either by adding a new entry to the Makefile, or by running the following commands:

cc -DKEYSRC=\"*keymap***.src\" genmap.c**

After this, the **keymap** file can be generated by running:

a.out > *keymap***.map**

The keymap can be loaded in the keyboard driver by:

loadkeys *keymap***.map**

It is wise to first run **loadkeys** on one of the maps in **/usr/lib/keymaps** so that you can easily revert back to a known keymap with a few taps on the up-arrow key and pressing return. You will otherwise have to fix the keymap with a faulty keymap loaded into the keyboard driver, which is no fun.

When the keymap is to your satisfaction you can copy it to **/etc/keymap** to have it loaded automatically at reboot.

FILES

/etc/keymap Default keymap file

SEE ALSO

loadkeys (1).

AUTHOR

Victor A. Rodriguez - El bit Fantasma (Bit-Man@Tasa.Com.AR)

NAME

passwd, group, shadow − user and group databases, shadow passwords

SYNOPSIS

/etc/passwd
/etc/group
/etc/shadow

DESCRIPTION

/etc/passwd lists all the users of the system, and **/etc/group** lists all the groups the users may belong to. Both files also contain encrypted passwords, numeric ID's etc. Encrypted passwords may be hidden in the file **/etc/shadow** if extra protection is warranted.

Each file is an text file containing one line per user or group. The data fields on a line are separated by colons. Each line in the password file has the following form:

name:passwd:uid:gid:gecos:dir:shell

The *name* field is the login name of a user, it is up to 8 letters or numbers long starting with a letter. The login name must be unique. The *password* field is either empty (no password), a 13 character encrypted password as returned by **crypt**(3), or a login name preceded by two number signs (#) to index the shadow password file. Anything else (usually ∗) is invalid. The *uid* and *gid* fields are two numbers indicating the users user-id and group-id. These id's do not have to be unique, there may be more than one name with the same id's. The *gecos* field can be set by the user. It is expected to be a comma separated list of personal data where the first item is the full name of the user. The *dir* field is the path name of the users home directory. Lastly the *shell* field is the path name of the users login shell, it may be empty to indicate **/bin/sh**. A Minix specific extension allows the shell field to contain extra space separated arguments for the shell.

Lines in the group file consist of four fields:

name:passwd:gid:mem

The *name* field is the name of the group, same restrictions as a login name. The *passwd* field may be used to let users change groups. The *gid* field is a number telling the group-id. The group-id is unique for a group. The *mem* field is a comma separated list of login names that are special members of the group. If a system supports supplementary group id's then a user's set of supplementary group id's is set to all the groups they are a member of. If a system allows one to change groups then one can change to a group one is a member of without using the group's password.

The shadow password file has precisely the same form as the password file, except that only the *name* or *passwd* fields are used as yet. The other fields are zero or empty. A password in the password file may have the form **##***user* to indicate the entry *user* in the shadow password file. The password in this entry is then used for authentication of the user. The shadow file can only be read by the privileged utility **pwdauth**(8), so that the encrypted passwords in the shadow file are kept secret, and thus safe from a dictionary attack.

Special password and group file entries

There are several entries in the password and group files that are preallocated for current or future use. All id's less than 10 are reserved. The special password file entries are:

```
root:##root:0:0:Big Brother:/usr/src:
daemon:*:1:1:The Deuce:/etc:
bin:##root:2:0:Binaries:/usr/src:
uucp:*:5:5:UNIX to UNIX copy:/usr/spool/uucp:/usr/sbin/uucico
news:*:6:6:Usenet news:/usr/spool/news:
ftp:*:7:7:Anonymous FTP:/usr/ftp:
nobody:*:9999:99::/tmp:
ast:*:8:3:Andrew S. Tanenbaum:/usr/ast:
```

The **root** id is of course the super user. The **daemon** id is used by some daemons. Some devices are protected so that only those daemons can access them. The **bin** id owns all sources and most binaries. The **uucp**, **news** and **ftp** id's are for serial line data transfer, usenet news, or ftp if so needed. The

nobody id is used in those cases that a program may not have any privileges at all. The **ast** id is the honorary home directory for Andrew S. Tanenbaum, the creator of Minix. You can also find the initial contents for a new home directory there.

The special group file entries are:

```
operator:*:0:
daemon:*:1:
bin:*:2:
other:*:3:
tty:*:4:
uucp:*:5:
news:*:6:
ftp:*:7:
kmem:*:8:
nogroup:*:99:
```

Groups with the same name as special user id are used with those id's. The **operator** group is for the administrators of the system. Users in this group are granted special privileges. The **other** group is for ordinary users. The **tty** group is for terminal devices, and associated set-gid commands. Same thing with the **kmem** group and memory devices.

FILES

/etc/passwd The user database.

/etc/group The group database.

/etc/shadow The shadow password file.

SEE ALSO

login(1), **passwd**(1), **su**(1), **crypt**(3), **getpwent**(3), **getgrent**(3), **pwdauth**(8).

NOTES

The **nobody** and **nogroup** id's are likely to be renumbered to the highest possible id's once it is figured out what they are.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

resolver − resolver configuration file

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The *resolver* is a set of routines in the C library (*resolv*(3)) that provide access to the Internet Domain Name System. The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of keywords with values that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine, the domain name is determined from the host name, and the domain search path is constructed from the domain name.

The different configuration options are:

nameserver
Internet address (in dot notation) of a name server that the resolver should query. Up to MAXNS (currently 3) name servers may be listed, one per keyword. If there are multiple servers, the resolver library queries them in the order listed. If no **nameserver** entries are present, the default is to use the name server on the local machine. (The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made).

domain Local domain name. Most queries for names within this domain can use short names relative to the local domain. If no **domain** entry is present, the domain is determined from the local host name returned by *gethostname*(2); the domain part is taken to be everything after the first '.'. Finally, if the host name does not contain a domain part, the root domain is assumed.

search Search list for host-name lookup. The search list is normally determined from the local domain name; by default, it begins with the local domain name, then successive parent domains that have at least two components in their names. This may be changed by listing the desired domain search path following the *search* keyword with spaces or tabs separating the names. Most resolver queries will be attempted using each component of the search path in turn until a match is found. Note that this process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local, and that queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains with a total of 256 characters.

The *domain* and *search* keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance will override.

The keyword and value must appear on a single line, and the keyword (e.g. **nameserver**) must start the line. The value follows the keyword, separated by white space.

FILES

/etc/resolv.conf

SEE ALSO

gethostbyname(3N), resolver(3), hostname(7), named(8)
Name Server Operations Guide for BIND

NAME

resolv.conf − Domain Name System resolver configuration

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The **/etc/resolv.conf** is used to configure how the host will use the Domain Name System to resolve hostnames to IP addresses. It may contain these two lines:

```
nameserver    IP-address
domain        domain-name
```

The nameserver entry tells the IP address of the host to use for DNS queries. If it is set to 127.0.0.1 (which is the default) then the local name daemon is used that may use the **/etc/hosts** database to translate host names. You normally only need a nameserver entry if the name server is at the other side of a router. The default **nonamed** name server can't look beyond the local network.

The domain entry tells the default domain to use for unqualified hostnames. This entry is usually not given in which case the domain of the local host is used.

The long version of this story can be found in **resolver**(5).

FILES

/etc/resolv.conf	DNS resolver configuration file.

SEE ALSO

resolver(5), **hosts**(5), **nonamed**(8), **boot**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

rhosts, hosts.equiv − trusted remote users or hosts

SYNOPSIS

˜*user*/.rhosts
/etc/hosts.equiv

DESCRIPTION

The per user **.rhosts** and the per system **hosts.equiv** files can be used to allow users to use **rlogin** or **rsh** without a password. The remote login services first check the system wide **/etc/hosts.equiv** file and then the **˜*user*/.rhosts** of the intended user. Both files contain lines of one of two forms:

host
host ruser

The first form tells that any user from *host* is allowed to login to this system under the same name. The second form allows *ruser* from *host* to login.

The **.rhosts** file is checked to be owned by *user* or root, and only its owner may be able to write it.

SEE ALSO

rlogin(1), **rsh**(1).

NOTES

Minix has no restrictions on becoming super-user remotely. Other systems usually do.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

termcap − terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals, used, *e.g.*, by **vi**(1) and **curses**(3). Terminals are described in **termcap** by giving a set of capabilities that they have and by describing how operations are performed. Padding requirements and initialization sequences are included in **termcap**.

Entries in **termcap** consist of a number of ':'-separated fields. The first entry for each terminal gives the names that are known for the terminal, separated by '|' characters. The first name is always two characters long and is used by older systems which store the terminal type in a 16-bit word in a system-wide data base. The second 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 first and 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 hp2621. This name should not contain hyphens. Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Therefore, 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 automatic 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)	concept100-na
-*n*p	Number of pages of memory	concept100-4p
-rv	Reverse video	concept100-rv

CAPABILITIES

The characters in the **Notes** field in the table have the following meanings (more than one may apply to a capability):

N indicates numeric parameter(s)
P indicates that padding may be specified
∗ indicates that padding may be based on the number of lines affected
o indicates capability is obsolete

Obsolete capabilities have no **terminfo** equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them at all.

Name	**Type**	**Notes**	**Description**
ae	str	(P)	End alternate character set
AL	str	(NP∗)	Add *n* new blank lines
al	str	(P∗)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str	(o)	Backspace if not **ˆH**
bl	str	(P)	Audible signal (bell)
bs	bool	(o)	Terminal can backspace with **ˆH**
bt	str	(P)	Back tab
bw	bool		**le** (backspace) wraps from column 0 to last column
CC	str		Terminal settable command character in prototype
cd	str	(P∗)	Clear to end of display
ce	str	(P)	Clear to end of line

ch	str	(NP)	Set cursor column (horizontal position)
cl	str	(P*)	Clear screen and home cursor
CM	str	(NP)	Memory-relative cursor addressing
cm	str	(NP)	Screen-relative cursor motion
co	num		Number of columns in a line (See BUGS section below)
cr	str	(P)	Carriage return
cs	str	(NP)	Change scrolling region (VT100)
ct	str	(P)	Clear all tab stops
cv	str	(NP)	Set cursor row (vertical position)
da	bool		Display may be retained above the screen
dB	num	(o)	Milliseconds of **bs** delay needed (default 0)
db	bool		Display may be retained below the screen
DC	str	(NP*)	Delete *n* characters
dC	num	(o)	Milliseconds of **cr** delay needed (default 0)
dc	str	(P*)	Delete character
dF	num	(o)	Milliseconds of **ff** delay needed (default 0)
DL	str	(NP*)	Delete *n* lines
dl	str	(P*)	Delete line
dm	str		Enter delete mode
dN	num	(o)	Milliseconds of **nl** delay needed (default 0)
DO	str	(NP*)	Move cursor down *n* lines
do	str		Down one line
ds	str		Disable status line
dT	num	(o)	Milliseconds of horizontal tab delay needed (default 0)
dV	num	(o)	Milliseconds of vertical tab delay needed (default 0)
ec	str	(NP)	Erase *n* characters
ed	str		End delete mode
ei	str		End insert mode
eo	bool		Can erase overstrikes with a blank
EP	bool	(o)	Even parity
es	bool		Escape can be used on the status line
ff	str	(P*)	Hardcopy terminal page eject
fs	str		Return from status line
gn	bool		Generic line type (*e.g.* dialup, switch)
hc	bool		Hardcopy terminal
HD	bool	(o)	Half-duplex
hd	str		Half-line down (forward 1/2 linefeed)
ho	str	(P)	Home cursor
hs	bool		Has extra status line
hu	str		Half-line up (reverse 1/2 linefeed)
hz	bool		Cannot print ˜s (Hazeltine)
i1-i3	str		Terminal initialization strings (**terminfo** only)
IC	str	(NP*)	Insert *n* blank characters
ic	str	(P*)	Insert character
if	str		Name of file containing initialization string
im	str		Enter insert mode
in	bool		Insert mode distinguishes nulls
iP	str		Pathname of program for initialization (**terminfo** only)
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string (**termcap** only)
it	num		Tabs initially every *n* positions
K1	str		Sent by keypad upper left
K2	str		Sent by keypad upper right
K3	str		Sent by keypad center
K4	str		Sent by keypad lower left

K5	str		Sent by keypad lower right
k0-k9	str		Sent by function keys 0-9
kA	str		Sent by insert-line key
ka	str		Sent by clear-all-tabs key
kb	str		Sent by backspace key
kC	str		Sent by clear-screen or erase key
kD	str		Sent by delete-character key
kd	str		Sent by down-arrow key
kE	str		Sent by clear-to-end-of-line key
ke	str		Out of keypad transmit mode
kF	str		Sent by scroll-forward/down key
kH	str		Sent by home-down key
kh	str		Sent by home key
kI	str		Sent by insert-character or enter-insert-mode key
kL	str		Sent by delete-line key
kl	str		Sent by left-arrow key
kM	str		Sent by insert key while in insert mode
km	bool		Has a meta key (shift, sets parity bit)
kN	str		Sent by next-page key
kn	num	(o)	Number of function (**k0**−**k9**) keys (default 0)
ko	str	(o)	Termcap entries for other non-function keys
kP	str		Sent by previous-page key
kR	str		Sent by scroll-backward/up key
kr	str		Sent by right-arrow key
kS	str		Sent by clear-to-end-of-screen key
ks	str		Put terminal in keypad transmit mode
kT	str		Sent by set-tab key
kt	str		Sent by clear-tab key
ku	str		Sent by up-arrow key
l0-l9	str		Labels on function keys if not f*n*
LC	bool	(o)	Lower-case only
LE	str	(NP)	Move cursor left *n* positions
le	str	(P)	Move cursor left one position
li	num		Number of lines on screen or page (See BUGS section below)
ll	str		Last line, first column
lm	num		Lines of memory if > **li** (0 means varies)
ma	str	(o)	Arrow key map (used by **vi** version 2 only)
mb	str		Turn on blinking attribute
md	str		Turn on bold (extra bright) attribute
me	str		Turn off all attributes
mh	str		Turn on half-bright attribute
mi	bool		Safe to move while in insert mode
mk	str		Turn on blank attribute (characters invisible)
ml	str	(o)	Memory lock on above cursor
mm	str		Turn on meta mode (8th bit)
mo	str		Turn off meta mode
mp	str		Turn on protected attribute
mr	str		Turn on reverse-video attibute
ms	bool		Safe to move in standout modes
mu	str	(o)	Memory unlock (turn off memory lock)
nc	bool	(o)	No correctly-working **cr** (Datamedia 2500, Hazeltine 2000)
nd	str		Non-destructive space (cursor right)
NL	bool	(o)	**\n** is newline, not line feed
nl	str	(o)	Newline character if not **\n**
ns	bool	(o)	Terminal is a CRT but doesn't scroll

nw	str	(P)	Newline (behaves like **cr** followed by **do**)
OP	bool	(o)	Odd parity
os	bool		Terminal overstrikes
pb	num		Lowest baud where delays are required
pc	str		Pad character (default NUL)
pf	str		Turn off the printer
pk	str		Program function key *n* to type string *s* (**terminfo** only)
pl	str		Program function key *n* to execute string *s* (**terminfo** only)
pO	str	(N)	Turn on the printer for *n* bytes
po	str		Turn on the printer
ps	str		Print contents of the screen
pt	bool	(o)	Has hardware tabs (may need to be set with **is**)
px	str		Program function key *n* to transmit string *s* (**terminfo** only)
r1-r3	str		Reset terminal completely to sane modes (**terminfo** only)
rc	str	(P)	Restore cursor to position of last **sc**
rf	str		Name of file containing reset codes
RI	str	(NP)	Move cursor right *n* positions
rp	str	(NP∗)	Repeat character *c* *n* times
rs	str		Reset terminal completely to sane modes (**termcap** only)
sa	str	(NP)	Define the video attributes
sc	str	(P)	Save cursor position
se	str		End standout mode
SF	str	(NP∗)	Scroll forward *n* lines
sf	str	(P)	Scroll text up
sg	num		Number of garbage chars left by **so** or **se** (default 0)
so	str		Begin standout mode
SR	str	(NP∗)	Scroll backward *n* lines
sr	str	(P)	Scroll text down
st	str		Set a tab in all rows, current column
ta	str	(P)	Tab to next 8-position hardware tab stop
tc	str		Entry of similar terminal − must be last
te	str		String to end programs that use **termcap**
ti	str		String to begin programs that use **termcap**
ts	str	(N)	Go to status line, column *n*
UC	bool	(o)	Upper-case only
uc	str		Underscore one character and move past it
ue	str		End underscore mode
ug	num		Number of garbage chars left by **us** or **ue** (default 0)
ul	bool		Underline character overstrikes
UP	str	(NP∗)	Move cursor up *n* lines
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (must not move cursor)
ve	str		Make cursor appear normal (undo **vs**/**vi**)
vi	str		Make cursor invisible
vs	str		Make cursor very visible
vt	num		Virtual terminal number (not supported on all systems)
wi	str	(N)	Set current window
ws	num		Number of columns in status line
xb	bool		Beehive (f1=ESC, f2=ˆC)
xn	bool		Newline ignored after 80 cols (Concept)
xo	bool		Terminal uses xoff/xon (DC3/DC1) handshaking
xr	bool	(o)	Return acts like **ce cr nl** (Delta Data)
xs	bool		Standout not erased by overwriting (Hewlett-Packard)
xt	bool		Tabs ruin, magic **so** char (Teleray 1061)

xx	bool	(o)	Tektronix 4025 insert-line

A Sample Entry

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

```
ca|concept100|c100|concept|c104|concept100-4p|HDS Concept−100:\
        :al=3*\E^R:am:bl=^G:cd=16*\E^C:ce=16\E^U:cl=2*^L:cm=\Ea%+ %+ :\
        :co#80:.cr=9^M:db:dc=16\E^A:dl=3*\E^B:do=^J:ei=\E\200:eo:im=\E^P:in:\
        :ip=16*:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E:k1=\E5:\
        :k2=\E6:k3=\E7:kb=^h:kd=\E<:ke=\Ex:kh=\E?:kl=\E>:kr=\E=:ks=\EX:\
        :ku=\E;:le=^H:li#24:mb=\EC:me=\EN\200:mh=\EE:mi:mk=\EH:mp=\EI:\
        :mr=\ED:nd=\E=:pb#9600:rp=0.2*\Er%.%+ :se=\Ed\Ee:sf=^J:so=\EE\ED:\
        :.ta=8\t:te=\Ev    \200\200\200\200\200\200\Ep\r\n:\
        :ti=\EU\Ev  8p\Ep\r:ue=\Eg:ul:up=\E;:us=\EG:\
        :vb=\Ek\200\200\200\200\200\200\200\200\200\200\200\200\200\200\EK:\
        :ve=\Ew:vs=\EW:vt#8:xn:\
        :bs:cr=^M:dC#9:dT#8:nl=^J:ta=^I:pt:
```

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). Comments may be included on lines beginning with #.

Types of Capabilities

Capabilities in **termcap** are of three types: Boolean capabilities, which indicate particular features that the terminal has; numeric capabilities, giving the size of the display or the size of other attributes; and string capabilities, which give character sequences that can be used to perform particular terminal operations. 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 Boolean capability **am**. Hence the description of the Concept includes **am**.

Numeric capabilities are followed by the character '#' then the value. In the example above **co**, which indicates the number of columns the display 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-letter code, an '=', then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, which causes padding characters to be supplied by **tputs** after the remainder of the string is sent to provide this delay. The delay can be either a number, *e.g.* '20', or a number followed by an '∗', *i.e.*, '3∗'. An '∗' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-line padding required. (In the case of insert-character, the factor is still the number of *lines* affected; this is always 1 unless the terminal has **in** and the software uses it.) When an '∗' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of control characters there. **\E** maps to an ESC character, **^X** maps to a control-X for any appropriate X, and the sequences **\n \r \t \b \f** map to linefeed, return, tab, backspace, and formfeed, respectively. 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 NUL 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 a **\000** would.)

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the first **cr** and **ta** in the example above.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **termcap** and to build up a description gradually, using partial descriptions with **vi** to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **termcap** file to describe it or bugs in **vi**. To easily test a new terminal description you can set the environment variable **TERMCAP** to the absolute pathname of a file containing the description you are working on and programs will look there rather than in **/etc/termcap**. **TERMCAP** can also be set to the **termcap** entry itself to avoid reading the file when starting up a program.

To get the padding for insert-line right (if the terminal manufacturer did not document it), a severe test is to use **vi** to edit **/etc/passwd** at 9600 baud, delete roughly 16 lines from the middle of the screen, then hit the 'u' key several times quickly. If the display 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 of the display is given by the **co** numeric capability. If the display is a CRT, then the number of lines on the screen is given by the **li** capability. If the display wraps around to the beginning of the next line when the cursor reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, the code to do this is given by the **cl** string capability. If the terminal overstrikes (rather than clearing the position when a character is overwritten), 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 the Tektronix 4010 series, as well as to hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage-return, **^M**.) If there is a code to produce an audible signal (bell, beep, *etc.*), give this as **bl**.

If there is a code (such as backspace) to move the cursor one position to the left, that capability should be given as **le**. Similarly, codes to move to the right, up, and down should be given as **nd**, **up**, and **do**, respectively. These *local cursor motions* should not alter the text they pass over; for example, you would not normally use nd= unless the terminal has the **os** capability, because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **termcap** have undefined behavior at the left and top edges of a CRT display. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up off the top using local cursor motions.

In order to scroll text up, a program goes to the bottom left corner of the screen and sends the **sf** (index) string. To scroll text down, a program goes to the top left corner of the screen and sends the **sr** (reverse index) string. The strings **sf** and **sr** have undefined behavior when not on their respective corners of the screen. Parameterized versions of the scrolling sequences are **SF** and **SR**, which have the same semantics as **sf** and **sr** except that they take one parameter and scroll that many lines. They also have undefined behavior except at the appropriate corner of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output there, but this does not necessarily apply to **nd** from the last column. Leftward local motion is defined from the left edge only when **bw** is given; then an **le** from the left edge will move to the right edge of the previous row. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the **termcap** description usually assumes that this feature is on, *i.e.*, **am**. If the terminal has a command that moves to the first column of the next line, that command can be given as **nw** (newline). It is permissible for this to clear the remainder of the current line, so if the terminal has no correctly-working CR and LF it may still be possible to craft a working **nw** out of one or both of them.

These capabilities suffice to describe hardcopy and glass-tty terminals. Thus the Teletype model 33 is described as

```
T3|tty33|33|tty|Teletype model 33:\
        :bl=^G:co#72:cr=^M:do=^J:hc:os:
```

and the Lear Siegler ADM−3 is described as

```
l3|adm3|3|LSI ADM-3:\
        :am:bl=^G:cl=^Z:co#80:cr=^M:do=^J:le=^H:li#24:sf=^J:
```

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**(3)-like escapes **%x** in it, while other characters are passed through unchanged. For example, to address the cursor the **cm** capability is given, using two parameters: the row and column to move 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 an analogous **CM** capability.)

The **%** encodings have the following meanings:

%%	output '%'
%d	output value as in **printf** %d
%2	output value as in **printf** %2d
%3	output value as in **printf** %3d
%.	output value as in **printf** %c
%+*x*	add *x* to value, then do %.
%>*xy*	if value > *x* then add *y*, no output
%r	reverse order of two parameters, no output
%i	increment by one, no output
%n	exclusive-or all parameters with 0140 (Datamedia 2500)
%B	BCD (16∗(value/10)) + (value%10), no output
%D	Reverse coding (value − 2∗(value%16)), no output (Delta Data)

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its **cm** capability is cm=6\E&%r%2c%2Y.

The Microterm ACT-IV needs the current row and column sent simply encoded in binary preceded by a **^T**, cm=^T%.%.. Terminals that use %. need to be able to backspace the cursor (**le**) and to move the cursor up one line on the screen (**up**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (Programs using **termcap** must set terminal modes so that tabs are not expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM−3a, which offsets row and column by a blank character, thus cm=\E=%+ %+ .

Row or column absolute cursor addressing can be given as single parameter capabilities **ch** (horizontal position absolute) and **cv** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cm**. If there are parameterized local motions (*e.g.*, move *n* positions to the right) these can be given as **DO**, **LE**, **RI**, and **UP** with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have **cm**, 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 the screen), 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 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 cursor address (0,0): to the top left corner of the screen, not of memory. (Therefore, the \EH sequence on Hewlett-Packard terminals cannot be used for **ho**.)

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, this should be given as **cd**. **cd** must only be invoked from the first column of a line. (Therefore, it can be simulated by a request to delete a large number of lines, if a true **cd** is not available.)

Insert/Delete Line

If the terminal can open a new blank line before the line containing the cursor, this should be given as **al**; this must be invoked only from the first position of a line. The cursor must then appear at the left of the newly blank line. If the terminal can delete the line that the cursor is on, this should be given as **dl**; this must only be used from the first position on the line to be deleted. Versions of **al** and **dl** which take a single parameter and insert or delete that many lines can be given as **AL** and **DL**. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the **cs** 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 **sr** or **sf** 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 **wi**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order. (This **terminfo** capability is described for completeness. It is unlikely that any **termcap**-using program will support it.)

If the terminal can retain display memory above the screen, 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 **sr** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using **termcap**. 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 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, then 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.* multi-line insert mode, and special treatment of untyped spaces), we have seen no terminals whose insert mode cannot be described with the single attribute.

Termcap can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode. Give as **ei** the sequence to leave insert mode. Now give as **ic** any sequence that needs to be sent just before each character to be inserted. Most terminals with a true insert mode will not give **ic**; terminals that use a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ic**. 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 that may need to be sent after insertion of a single character can also be given in **ip**. If your terminal needs to be placed into an ‘insert mode’ and needs a special code preceding each inserted character, then both **im**/**ei** and **ic** can be given, and both will be used. The **IC** capability, with one parameter *n*, will repeat the effects of **ic** *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 **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify **dc** to delete a single character, **DC** with one parameter *n* to delete *n* characters, and delete mode by giving **dm** and **ed** to enter and exit delete mode (which is any mode the terminal needs to be placed in for **dc** to work).

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 **so** and **se**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces or garbage characters on the screen, as the TVI 912 and Teleray 1061 do, then **sg** should be given to tell how many characters are left.

Codes to begin underlining and end underlining can be given as **us** and **ue**, respectively. Underline mode change garbage is specified by **ug**, similar to **sg**. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **mb** (blinking), **md** (bold or extra bright), **mh** (dim or half-bright), **mk** (blanking or invisible text), **mp** (protected), **mr** (reverse video), **me** (turn off *all* attribute modes), **as** (enter alternate character set mode), and **ae** (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 mode, this should be given as **sa** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attributes is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. Not all modes need be supported by **sa**, only those for which corresponding attribute commands exist. (It is unlikely that a **termcap**-using program will support this capability, which is defined for compatibility with **terminfo**.)

Terminals with the magic cookie glitches (**sg** and **ug**), rather than maintaining extra attribute bits for each character cell, instead deposit special cookies, or garbage characters, when they receive mode-setting sequences, which affect the display algorithm.

Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or when the cursor is addressed. Programs using standout mode should exit standout mode on such terminals before moving the cursor or sending a newline. On terminals where this is not a problem, the **ms** capability should be present to say that this overhead is unnecessary.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as **vb**; 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 change, for example, a non-blinking underline into an easier-to-find block or blinking underline), give this sequence as **vs**. If there is a way to make the cursor completely invisible, give that as **vi**. The capability **ve**, which undoes the effects of both of these modes, should also be given.

If your terminal correctly displays 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, 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 Hewlett-Packard 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**. The codes transmitted by certain other special keys can be given: **kH** (home down), **kb** (backspace), **ka** (clear all tabs), **kt** (clear the tab stop in this column), **kC** (clear screen or erase), **kD** (delete character), **kL** (delete line), **kM** (exit insert mode), **kE** (clear to end of line), **kS** (clear to end of screen), **kI** (insert character or enter insert mode), **kA** (insert line), **kN** (next page), **kP** (previous page), **kF** (scroll forward/down), **kR** (scroll backward/up), and **kT** (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, then the other five keys can be given as **K1**, **K2**, **K3**, **K4**, and **K5**. These keys are useful when the effects of a 3 by 3 directional pad are needed. The obsolete **ko** capability formerly used to describe other function keys has been completely supplanted by the above capabilities.

The **ma** entry is also used to indicate arrow keys on terminals that have single-character arrow keys. It is obsolete but still in use in version 2 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, and 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 have ma=ˆHhˆKjˆZkˆXl indicating arrow keys left (ˆH), down (ˆK), up (ˆZ), and right (ˆX). (There is no home key on the Mime.)

Tabs and Initialization

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 **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 screen-sized window must be fixed into the display for cursor addressing to work properly. This is also used for the Tektronix 4025, where **ti** sets the command character to be the one used by **termcap**.

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 set the terminal into modes consistent with the rest of the **termcap** description. They are normally sent to the terminal by the **tset** program each time the user logs in. They will be printed in the following order: **is**; setting tabs using **ct** and **st**; and finally **if**. (*Terminfo* uses **i1-i2** instead of **is** and runs the program **iP** and prints **i3** after the other initializations.) A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs** and **if**. These strings are output by the **reset** program, which is used when the terminal gets into a wedged state. (*Terminfo* uses **r1-r3** instead of **rs**.) Commands are normally placed in **rs** and **rf** only if they produce 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 **is**, 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 the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ta** (usually **ˆI**). A backtab command which moves leftward to the previous tab stop can be given as **bt**. By convention, if the terminal driver modes indicate that tab stops are being expanded by the computer rather than being sent to the terminal, programs should not use **ta** or **bt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every *n* positions when the terminal is powered up, then the numeric parameter **it** is given, showing the number of positions between tab stops. This is normally used by the **tset** command to determine whether to set the driver mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **termcap** description can assume that they are properly set.

If there are commands to set and clear tab stops, they can be given as **ct** (clear all tab stops) and **st** (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 **is** or **if**.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the **tset** program to set terminal driver modes appropriately. Delays embedded in the capabilities **cr**, **sf**, **le**, **ff**, and **ta** will cause the appropriate delay bits to be set in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**. For 4.2BSD **tset**, the delays are given as numeric capabilities **dC**, **dN**, **dB**, **dF**, and **dT** instead.

Miscellaneous

If the terminal requires other than a NUL (zero) character as a pad, this can be given as **pc**. Only the first character of the **pc** string is used.

If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**.

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, then the capability **hs** should be given. Special strings to go to a position in the status line and to return from the status line can be given as **ts** and **fs**. (**fs** must leave the cursor position in the same place that it was before **ts**. If necessary, the **sc** and **rc** strings can be included in **ts** and **fs** to get this effect.) The capability **ts** takes one parameter, which is the column number of the status line to which the cursor is to be moved. If escape sequences and other special commands such as tab work while in the status line, the flag **es** can be given. A string that turns off the status line (or otherwise erases its contents) should be given as **ds**. The status line is normally assumed to be the same width as the rest of the screen, *i.e.*, **co**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), then its width in columns can be indicated with the numeric parameter **ws**.

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 **^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 **rp**. The first parameter is the character to be repeated and the second is the number of times to repeat it. (This is a **terminfo** feature that is unlikely to be supported by a program that uses **termcap**.)

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **CC**. A prototype command character is chosen which is used in all capabilities. This character is given in the **CC** 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 by the character in the environment variable. This use of the **CC** environment variable is a very bad idea, as it conflicts with **make**(1).

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 xoff/xon (DC3/DC1) handshaking for flow control, give **xo**. 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 which acts as a shift key, setting the 8th bit of any character transmitted, then this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this meta mode on and off, they can be given as **mm** and **mo**.

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**. An explicit value of 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 system 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 **ps**: print the contents of the screen; **pf**: turn off the printer; and **po**: 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 **pO** 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 **pf**, is transparently passed to the printer while **pO** is in effect.

Strings to program function keys can be given as **pk**, **pl**, and **px**. Each of these strings takes two parameters: the function key number to program (from 0 to 9) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The differences among the capabilities are that **pk** causes pressing the given key to be the same as the user typing the given string; **pl** causes the string to be executed by the terminal in local mode; and **px** causes the string to be transmitted to the computer. Unfortunately, due to lack of a definition for string parameters in **termcap**, only **terminfo** supports these capabilities.

Glitches and Braindamage

Hazeltine terminals, which do not allow '~' characters to be displayed, should indicate **hz**.

The **nc** capability, now obsolete, formerly indicated Datamedia terminals, which echo **\r \n** for carriage return then ignore a following linefeed.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept, should indicate **xn**.

If **ce** is required to get rid of standout (instead of merely writing normal text on top of it), **xs** 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, and that to erase standout mode it is necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the ESC or ^C characters, has **xb**, indicating that the f1 key is used for ESC and f2 for ^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 **x***x*.

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 entries must not exceed 1024. The capabilities given before **tc** override those in the terminal type invoked by **tc**. A capability can be canceled by placing **xx@** to the left of the **tc** invocation, where *xx* is the capability. For example, the entry

    hn|2621−nl:ks@:ke@:tc=2621:

defines a 2621−nl that does not have the **ks** or **ke** capabilities, 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.

AUTHOR

William Joy
Mark Horton added underlining and keypad support

FILES

/etc/termcap file containing terminal descriptions **/usr/etc/termcap** file containing more terminal descriptions (Minix-vmd)

SEE ALSO

elvis(1), **more**(1), **termcap**(3), **printf**(3).

CAVEATS AND BUGS

Lines and columns are now stored by the kernel as well as in the termcap entry. Most programs now use the kernel information primarily; the information in this file is used only if the kernel does not have any information.

Not all programs support all entries.

The Minix **termcap**(3) does not understand everything described here, unlike the one Minix-vmd uses.

NAME

ttytab − table of login terminals

SYNOPSIS

/etc/ttytab

DESCRIPTION

The **ttytab** file lists all the terminal devices that one can login on. It is a simple text file that contains lines of the form:

name type "getty" "init"

The *name* and *type* fields are simple words, *name* is the name of the terminal device with **/dev** stripped off, and *type* tells the type of terminal to initialize the **TERM** environment variable.

The *getty* and *init* fields may name commands that are run to allow one to login on the line, or to initialize the line. Both these fields may be more than one word if the whole field is enclosed in double quotes. *Getty* is usually simply the word **getty**, the command that prints a system identification banner and allows on to type a name to log in. *Init* is usually an **stty** command to set the baud rate and parity of a serial line.

The *init* field may be omitted to indicate that no initialization is necessary, and the *getty* field may be left out to not start a login process. Terminals should not be left out, because their place in the **ttytab** file determines their slot number as returned by **ttyslot**(3).

Comments (introduced by #) and empty lines are ignored.

EXAMPLE

A **ttytab** for the console, two serial lines, and a pseudo tty entry:

console	minix	getty	
tty00	vt100	getty	"stty 9600"
tty01	dialup	getty	"stty 38400"
ttyp0	network		

ENVIRONMENT

TERM Terminal type

NOTES

It is customary to set the type to **dialup** for a dialin line. One can check for that name in one's **.profile**.

SEE ALSO

gettyent(3), **ttyslot**(3), **init**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

TZ − Time zone environment variable

SYNOPSIS

TZ=*zone*[**−**]*offset*[*dst*[*offset*][**,***start*[/*time*]**,***end*[/*time*]]]

DESCRIPTION

The **TZ** environment variable tells functions such as the **ctime**(3) family and programs like **date** what the time zone and daylight saving rule is. The value of **TZ** has the POSIX standardized form shown in the synopsis. This form specifies the zone names, offsets from GMT, and daylight savings changeover times for at least the current year.

zone A three or more letter name for the time zone in normal (winter) time.

[−]*offset*
: A signed time telling the offset of the time zone westwards from Greenwich. The time has the form *hh[:mm[:ss]]* with a one of two digit hour, and optional two digit minutes and seconds.

dst The name of the time zone when daylight savings is in effect. It may be followed by an offset telling how big the clock correction is other than the default of 1 hour.

start/*time*,*end*/*time*
: Specifies the start and end of the daylight savings period. The *start* and *end* fields indicate on what day the changeover occurs. They must be in one of the following formats:

J*n* The Julian day *n* (1 <= *n* <= 365) ignoring leap days, i.e. there is no February 29.

n The zero-based Julian day (0 <= *n* <= 365). Leap days are not ignored.

M*m***.***n***.***d*
: This indicates month *m*, the *n*-th occurrence of day *d* (1 <= *m* <= 12, 1 <= *n* <= 5, 0 <= *d* <= 6, 0=Sunday). The 5-th occurrence means the last occurrence of that day in a month. So **M4.1.0** is the first Sunday in April, **M9.5.0** is the last Sunday in September.

The *time* field indicates the time the changeover occurs on the given day.

EXAMPLES

Greenwich Mean Time:

TZ=GMT0

Middle European Time, 1 hour east from Greenwich, daylight savings starts on the last Sunday in March at 2 AM and ends on the last Sunday in October at 3 AM:

TZ='MET−1MET DST,M3.5.0/2,M10.5.0/3'

British time, daylight savings starts and ends at the same moment as MET, but in an earlier time zone:

TZ=GMT0BST,M3.5.0/1,M10.5.0/2

The eastern european time zones also have the changeovers at the same absolute time as British time and MET.

U.S. Eastern Standard Time, 5 hours west from Greenwich, daylight savings starts on the first Sunday in April at 2 AM and ends on the last Sunday in October at 2 AM:

TZ=EST5EDT,M4.1.0/2,M10.5.0/2

It shouldn't surprise you that daylight savings in New Zealand is observed in the months opposite from the previous examples. It starts on the first Sunday in October at 2 AM and ends on the third Sunday in March at 3 AM:

TZ=NZST−12NZDT,M10.1.0/2,M3.3.0/3

SEE ALSO

readclock(8), **date**(1).

BUGS

You may have noticed that many fields are optional. Do no omit them, because the defaults are bogus. If you need daylight savings then fully specify the changeovers.

West is negative, east is positive, ask any sailor.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

utmp, wtmp − logged in users, login and logout history

SYNOPSIS

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

DESCRIPTION

The files **/etc/utmp** and **/usr/adm/wtmp** respectively contain the currently logged in users, and the history of logins and logouts.

Each file is an array of the following structure defined in <utmp.h>:

```
struct utmp {
        char            ut_user[8];     /* user name */
        char            ut_line[12];    /* terminal name */
        char            ut_host[16];    /* host name, when remote */
        time_t          ut_time;        /* login/logout time */
};
```

#define ut_name ut_user/* for compatibility with other systems */

The structure contains more fields than those listed, but they are only of interest to **init** and **login**. Note that the **ut_name** field is a compatibility alias for **ut_user**, it is actually better to use it.

A login entry is completely specified. A logout entry has a null string for **ut_name**. A shutdown or reboot entry has an **ut_line** field containing a "~" (tilde). The **ut_name** field is usually the name of the program that did the shutdown, or "reboot" at reboot. This is a bit confusing, but note that there should always be two such entries. If you see just one entry then the system has crashed, if you see two entries then the system was properly shut down and later rebooted.

FILES

/etc/utmp Currently logged in users.

/usr/adm/wtmp History of logins and logouts.

SEE ALSO

who(1), **ttyslot**(3).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

whatis − database of online manual pages

SYNOPSIS

/usr/man/*/whatis
/usr/man/whatis

DESCRIPTION

The **whatis** file in each manual page directory is a database of titles for manual pages. This database is used by **man**(1) to map titles to manual pages names. The database is created by **makewhatis**(1) from the NAME sections of the manual pages.

The NAME secions must be simple lines with no troff fluff but one backslash like these two:

> whatis \− database of online manual pages
> cawf, nroff \− C version of the nroff-like, Amazingly Workable (text) Formatter

These lines are transformed by **makewhatis** to these two lines for the database:

> cawf, nroff (1) − C version of the nroff-like, Amazingly Workable (text) Formatter
> whatis (5) − database of online manual pages

As you can see they are in section number order, so that **man** searches them in section order.

Each entry is just a single line, restricting the NAME section to a single line too with just one dash, and commas and spaces before the dash as you see above.

SEE ALSO

man(1), **whatis**(1), **makewhatis**(1), **man**(7).

BUGS

It seems to be impossible for many manual page writers to keep the NAME section simple. They also like to use every font available in their documents. My simple scripts can’t read their NAME sections, my simple me can’t read their texts.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)