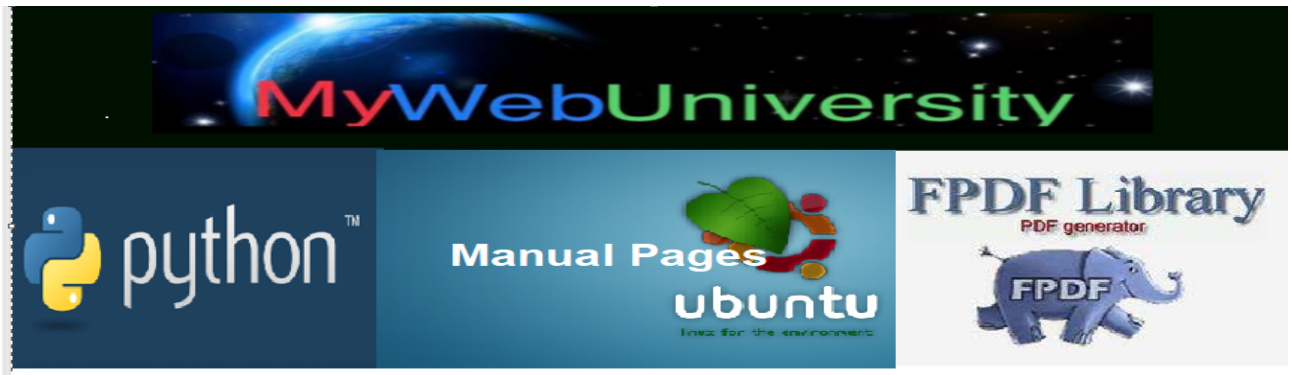

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'tput.1'

$ man tput.1

tput(1) User commands tput(1)

NAME

tput - initialize a terminal, exercise its capabilities, or query terminfo database

SYNOPSIS

tput [-v] [-T terminal-type] {cap-code [parameter ...]} ...

tput [-v] [-T terminal-type] [-x] clear

tput [-v] [-T terminal-type] init

tput [-v] [-T terminal-type] reset

tput [-v] [-T terminal-type] longname

tput [-v] -S

tput [-v] -V

DESCRIPTION

tput uses the terminfo library and database to make terminal-specific capabilities and in? formation available to the shell, to initialize or reset the terminal, or to report a de? scription of the current (or specified) terminal type. Terminal capabilities are accessed by cap-code.

terminfo(5) discusses terminal capabilities at length and presents a complete list of stan? dardized cap-codes. user_caps(5) presents other widely used but non-standard capabilities.

When retrieving capability values, the result depends upon the capability's type.

Boolean tput sets its exit status to 0 if the terminal possesses cap-code, and 1 if it does not.

numeric tput writes cap-code's decimal value to the standard output stream if defined (-1 if it is not) followed by a newline.

string tput writes cap-code's value to the standard output stream if defined, without a trailing newline.

Before using a value returned on the standard output, the application should test tput's exit status to be sure it is 0; see section ?EXIT STATUS? below.

Operands

Generally, an operand is a cap-code, a capability code from the terminal database, or a parameter thereto. Three others are specially recognized by tput: init, reset, and longname. Although these resemble capability codes, they in fact receive special handling; we term them ?pseudo-capabilities?.

cap-code indicates a capability from the terminal database.

If cap-code is of string type and takes parameters, tput interprets arguments following cap-code as the parameters, up to the (fixed) quantity the capability requires.

Most parameters are numeric. Only a few terminal capabilities require string parameters; tput uses a table to decide which to pass as strings. Normally tput uses tparm(3NCURSES) to perform the substitution. If no parameters are given for the capability, tput writes the string without performing the substitution.

init initializes the terminal. If the terminal database is present and an entry for the user's terminal type exists, the following occur.

(1) tput retrieves the terminal's mode settings. It successively tests the file descriptors corresponding to

- the standard error stream,
- the standard output stream,
- the standard input stream, and
- /dev/tty

to obtain terminal settings. Having retrieved them, tput remembers which descriptor to use for further updates.

(2) If the terminal dimensions cannot be obtained from the operating system, but the environment or terminal type database entry describes them, tput updates the operating system's notion of them.

(3) tput updates the terminal modes.

- Tab expansion is turned on or off per the specification in the entry, and

? if tabs are not expanded, standard tabs (every 8 spaces) are set.

(4) If initialization capabilities, detailed in subsection ?Tabs and Initializa?
tion? of terminfo(5), are present, tput writes them to the standard output stream.

(5) tput flushes the standard output stream.

If an entry lacks the information needed for an activity above, that activity is silently skipped.

reset re-initializes the terminal. A reset differs from initialization in two ways.

(1) tput sets the terminal modes to a ?sane? state,

? enabling canonical (?cooked?) and echo modes,

? disabling cbreak and raw modes,

? enabling newline translation, and

? setting any special input characters to their default values.

(2) If any reset capabilities are defined for the terminal type, tput writes them to the output stream. Otherwise, tput uses any defined initialization capabilities. Reset capabilities are detailed in subsection ?Tabs and Ini?
tialization? of terminfo(5).

longname A terminfo entry begins with one or more names by which an application can refer to the entry, before the list of terminal capabilities. The names are separated by ?|? characters. X/Open Curses terms the last name the ?long name?, and indi?
cates that it may include blanks.

tic warns if the last name does not include blanks, to accommodate old terminfo entries that treated the long name as an optional feature. The long name is of?
ten referred to as the description field.

If the terminal database is present and an entry for the user's terminal type ex?
ists, tput reports its description to the standard output stream, without a trailing newline. See terminfo(5).

Note: Redirecting the output of ?tput init? or ?tput reset? to a file will capture only part of their actions. Changes to the terminal modes are not affected by file descriptor redi?
rection, since the terminal modes are altered via ioctl(2).

Aliases

If tput is invoked via link with any of the names clear, init, or reset, it operates as if run with the corresponding (pseudo-)capability operand. For example, executing a link named

reset that points to tput has the same effect as ?tput reset?.

This feature was introduced by ncurses 5.2 in 2000. It is rarely used.

clear is a separate program, which is both smaller and more frequently executed.

init has the same name as another program in widespread use.

reset is provided by the tset(1) utility (also via a link named reset).

Terminal Size

Besides the pseudo-capabilities (such as init), tput treats the lines and cols cap-codes specially: it may call setupterm(3NCURSES) to obtain the terminal size.

? First, tput attempts to obtain these capabilities from the terminal database. This gen? erally fails for terminal emulators, which lack a fixed window size and thus omit the capabilities.

? It then asks the operating system for the terminal's size, which generally works, unless the connection is via a serial line that does not support ?NAWS?: negotiations about window size.

? Finally, it inspects the environment variables LINES and COLUMNS, which may override the terminal size.

If the -T option is given, tput ignores the environment variables by calling use_tioctl(TRUE), relying upon the operating system (or, ultimately, the terminal database).

OPTIONS

-S retrieves more than one capability per invocation of tput. The capabilities must be passed to tput from the standard input stream instead of from the command line (see section ?EXAMPLES? below). Only one cap-code is allowed per line. The -S op? tion changes the meanings of the 0 and 1 exit statuses (see section ?EXIT STATUS? below).

Some capabilities use string parameters rather than numeric ones. tput employs a built-in table and the presence of parameters in its input to decide how to inter? pret them, and whether to use tparm(3NCURSES).

-T type indicates the terminal's type. Normally this option is unnecessary, because a de? fault is taken from the TERM environment variable. If specified, the environment variables LINES and COLUMNS are also ignored.

-v causes tput to operate verbosely, reporting warnings.

-V reports the version of ncurses associated with tput, and exits with a successful status.

-x prevents ?tput clear? from attempting to clear the scrollback buffer.

EXIT STATUS

Normally, one should interpret tput's exit statuses as follows.

Status Meaning When -S Not Specified

??

0 Boolean or string capability present

1 Boolean or numeric capability absent

2 usage error or no terminal type specified

3 unrecognized terminal type

4 unrecognized capability code

>4 system error (4 + errno)

When the -S option is used, some statuses change meanings.

Status Meaning When -S Specified

??

0 all operands interpreted

1 unused

4 some operands not interpreted

ENVIRONMENT

tput reads up to three environment variables if the -T option is not specified.

COLUMNS specifies the width of the screen in characters.

LINES specifies the height of the screen in characters.

TERM denotes the terminal type. Each terminal type is distinct, though many are simi?

lar.

FILES

/usr/share/tabset

tab stop initialization database

/etc/terminfo

compiled terminal description database

PORTABILITY

Over time ncurses tput has differed from that of System V in two important respects, one now

mostly historical.

? ?tput cap-code? writes to the standard output, which need not be a terminal device.

However, the operands that manipulate terminal modes might not use the standard output.

System V tput's init and reset operands use logic from 4.1cBSD tset, manipulating termi? nal modes. It checks the same file descriptors (and /dev/tty) for association with a terminal device as ncurses now does, and if none are, finally assumes a 1200 baud termi? nal. When updating terminal modes, it ignores errors.

Until ncurses 6.1 (see section ?HISTORY? below), tput did not modify terminal modes. It now employs a scheme similar to System V, using functions shared with tset (and ulti? mately based on 4.4BSD tset). If it is not able to open a terminal (for instance, when run by cron(1)), tput exits with an error status.

? System V tput assumes that the type of a cap-code operand is numeric if all the charac? ters of its value are decimal numbers; if they are not, it treats cap-code as a string capability.

Most implementations that provide support for cap-code operands use the tparm(3NCURSES) function to expand its parameters. That function expects a mixture of numeric and string parameters, requiring tput to know which type to use.

ncurses tput uses a table to determine the parameter types for the standard cap-code operands, and an internal function to analyze nonstandard cap-code operands.

While more reliable than System V's utility, a portability problem is introduced by this analysis. An OpenBSD developer adapted the internal library function from ncurses to port NetBSD's termcap-based tput to terminfo, and modified it to interpret multiple cap-codes (and parameters) on the command line. Portable applications should not rely upon this feature; ncurses offers it to support applications written specifically for OpenBSD.

ncurses's implementation of tput, unlike others, accepts both termcap and terminfo cap-codes if termcap support is compiled in. In that case, however, termcap and terminfo codes have two ambiguities; ncurses assumes the terminfo code.

? The cap-code dl means delete_line to termcap but parm_delete_line to terminfo. termcap uses the code DL for parm_delete_line. terminfo uses the code dl1 for delete_line.

? The cap-code ed means exit_delete_mode to termcap but clr_eos to terminfo. termcap uses the code cd for clr_eos. terminfo uses the code rmdc for exit_delete_mode.

The longname operand, -S option, and the parameter-substitution features used in the cup ex? ample below, were not supported in AT&T/USL curses before SVr4 (1989). Later, 4.3BSD-Reno (1990) added support for longname, and in 1994, NetBSD added support for the parameter-sub? stitution features.

IEEE Std 1003.1/The Open Group Base Specifications Issue 7 (POSIX.1-2008) documents only the clear, init, and reset operands. A few observations of interest arise from that selection.

? ncurses supports clear as it does any other standard cap-code. The others (init and longname) do not correspond to terminal capabilities.

? The tput on SVr4-based systems such as Solaris, IRIX64, and HP-UX, as well as others such as AIX and Tru64, also support standard cap-code operands.

? A few platforms such as FreeBSD recognize termcap codes rather than terminfo capability codes in their respective tput commands. Since 2010, NetBSD's tput uses terminfo codes. Before that, it (like FreeBSD) recognized termcap codes.

Beginning in 2021, FreeBSD uses ncurses tput, configured for both terminfo (tested first) and termcap (as a fallback).

Because (apparently) all certified Unix systems support the full set of capability codes, the reason for documenting only a few may not be apparent.

? X/Open Curses Issue 7 documents tput differently, with cap-code and the other features used in this implementation.

? That is, there are two standards for tput: POSIX (a subset) and X/Open Curses (the full implementation). POSIX documents a subset to avoid the complication of including X/Open Curses and the terminal capability database.

? While it is certainly possible to write a tput program without using curses, no system with a curses implementation provides a tput utility that does not also support standard cap-codes.

X/Open Curses Issue 7 (2009) is the first version to document utilities. However that part of X/Open Curses does not follow existing practice (that is, System V curses behavior).

? It assigns exit status 4 to ?invalid operand?, which may have the same meaning as ?un? known capability?. For instance, the source code for Solaris xcurses uses the term ?in? valid? in this case.

? It assigns exit status 255 to a numeric variable that is not specified in the terminfo database. That likely is a documentation error, mistaking the ?-1? written to the stan? dard output to indicate an absent or canceled numeric capability for an (unsigned) exit status.

The various System V implementations (AIX, HP-UX, Solaris) use the same exit statuses as ncurses.

NetBSD curses documents exit statuses that correspond to neither ncurses nor X/Open Curses.

HISTORY

Bill Joy wrote a tput command during development of 4BSD in October 1980. This initial ver? sion only cleared the screen, and did not ship with official distributions.

System V developed a different tput command.

? SVr2 (1984) provided a rudimentary tput that checked the parameter against each capabil? ity name and returned the corresponding value. This version of tput did not use tparm(3NCURSES) for parameterized capabilities.

? SVr3 (1987) replaced that with a more extensive program whose support for init and reset operands (more than half the program) incorporated the reset feature of BSD tset written by Eric Allman.

? SVr4 (1989) added color initialization by using the orig_colors (oc) and orig_pair (op) capabilities in its init logic.

Keith Bostic refactored BSD tput for shipment in 4.3BSD-Reno (1990), making it follow the interface of System V tput by accepting some parameters named for terminfo (pseudo-)capabil? ities: clear, init, longname, and reset. However, because he had only termcap available, it accepted termcap codes for other capabilities. Also, Bostic's BSD tput did not modify the terminal modes as the earlier BSD tset had done. At the same time, Bostic added a shell script named ?clear? that used tput to clear the screen. These became the ?modern? BSD im? plementation of tput.

The origin of ncurses tput lies outside both System V and BSD, in Ross Ridge's mytinfo pack? age, published on comp.sources.unix in December 1992. Ridge's program made more sophisti? cated use of the terminal capabilities than the BSD program. Eric Raymond used that tput program (and other parts of mytinfo) in ncurses in June 1995. Incorporating the portions dealing with terminal capabilities almost without change, Raymond made improvements to the way command-line parameters were handled.

Before ncurses 6.1 (2018), its tset and tput utilities differed.

? tset was more effective, resetting the terminal's modes and special input characters.

? On the other hand, tset's repertoire of terminal capabilities for resetting the terminal was more limited; it had only equivalents of reset_1string (rs1), reset_2string (rs2), and reset_file (rf), and not the tab stop and margin update features of tput.

The reset program is traditionally an alias for tset due to its ability to reset the termi? nal's modes and special input characters.

As of ncurses 6.1, the ?reset? features of the two programs are (mostly) the same. Two mi?

nor differences remain.

? When issuing a reset, the tset program checks whether the device appears to be a pseudoterminal (as might be used by a terminal emulator program), and, if it does not, waits one second in case it is communicating with a hardware terminal.

? The two programs write the terminal initialization strings to different streams; that is, standard error for tset and standard output for tput.

EXAMPLES

tput init

Initialize the terminal according to the type of terminal in the TERM environment variable. If the system does not reliably initialize the terminal upon login, this command can be included in $HOME/.profile after exporting the TERM environment vari? able.

tput -T5620 reset

Reset an AT&T 5620 terminal, overriding the terminal type in the TERM environment variable.

tput cnorm

Set cursor to normal visibility.

tput home

Move the cursor to line 0, column 0: the upper left corner of the screen, usually known as the ?home? cursor position.

tput clear

Clear the screen: write the clear_screen capability's value to the standard output stream.

tput cols

Report the number of columns used by the current terminal type.

tput -Tadm3a cols

Report the number of columns used by an ADM-3A terminal.

strong=`tput smso` normal=`tput rmso`

Set shell variables to capability values: strong and normal, to begin and end, re? spectively, stand-out mode for the terminal. One might use these to present a prompt.

```
printf "${strong}Username:${normal} "
```

tput hc

Indicate via exit status whether the terminal is a hard copy device.

tput cup 23 4

Move the cursor to line 23, column 4.

tput cup

Report the value of the cursor_address (cup) capability (used for cursor movement), with no parameters substituted.

tput longname

Report the terminfo database's description of the terminal type specified in the TERM environment variable.

tput -S

Process multiple capabilities. The -S option can be profitably used with a shell ?here document?.

$ tput -S <<!

> clear

> cup 10 10

> bold

> !

The foregoing clears the screen, moves the cursor to position (10, 10) and turns on bold (extra bright) mode.

tput clear cup 10 10 bold

Perform the same actions as the foregoing ?tput -S? example.

SEE ALSO

clear(1), stty(1), tabs(1), tset(1), termcap(3NCURSES), terminfo(5), user_caps(5)