filename: VT100_escape_codes_20180228.txt
https://www.csie.ntu.edu.tw/~r92094/c++/VT100.html

VT100 escape codes

This document describes how to control a VT100 terminal. The entries are of the form "name, description, escape code".

The name isn't important, and the description is just to help you find what you're looking for. What you have
to do is send the "escape code" to the screen. These codes are often several characters long, but they all
begin with ^[. This isn't the two characters ^ and [, but rather a representation of the ASCII code ESC (which is why these are called escape codes).

ESC has the value:

	Ada	27, #8#33 #16#1B
	C	27, 033, 0x1b
	English	27 decimal, 33 octal, 1b hexadecimal

and should be sent before the rest of the code, which is simply an ASCII string.

As an example of how to use this information, here's how to clear the screen in Ada and C, using the VT100
escape codes:

Ada

```
PUT( ASCII.ESC );
PUT_LINE( "[2J" );
```

C

```
#define ASCII_ESC 27
printf( "%c[2J", ESC );
```

or

```
puts( "\033[2J" );
```

Name	Description	Esc Code
setnl LMN	Set new line mode	^[[20h
setappl DECCKM	Set cursor key to application	^[[?1h
setansi DECANM	Set ANSI (versus VT52)	none
setcol DECCOLM	Set number of columns to 132	^[[?3h
setsmooth DECSCLM	Set smooth scrolling	^[[?4h
setrevscrn DECSCNM	Set reverse video on screen	^[[?5h
setorgrel DECOM	Set origin to relative	^[[?6h
setwrap DECAWM	Set auto-wrap mode	^[[?7h
setrep DECARM	Set auto-repeat mode	^[[?8h
setinter DECINLM	Set interlacing mode	^[[?9h
setlf LMN	Set line feed mode	^[[20l
setcursor DECCKM	Set cursor key to cursor	^[[?1l
setvt52 DECANM	Set VT52 (versus ANSI)	^[[?2l
resetcol DECCOLM	Set number of columns to 80	^[[?3l
setjump DECSCLM	Set jump scrolling	^[[?4l
setnormscrn DECSCNM	Set normal video on screen	^[[?5l
setorgabs DECOM	Set origin to absolute	^[[?6l
resetwrap DECAWM	Reset auto-wrap mode	^[[?7l
resetrep DECARM	Reset auto-repeat mode	^[[?8l
resetinter DECINLM	Reset interlacing mode	^[[?9l
altkeypad DECKPAM	Set alternate keypad mode	^[=
numkeypad DECKPNM	Set numeric keypad mode	^[>
setukg0	Set United Kingdom G0 character set	^[(A

```
setukg1               Set United Kingdom G1 character set    ^[)A
setusg0               Set United States G0 character set     ^[(B
setusg1               Set United States G1 character set     ^[)B
setspecg0             Set G0 special chars. & line set       ^[(0
setspecg1             Set G1 special chars. & line set       ^[)0
setaltg0              Set G0 alternate character ROM         ^[(1
setaltg1              Set G1 alternate character ROM         ^[)1
setaltspecg0          Set G0 alt char ROM and spec. graphics ^[(2
setaltspecg1          Set G1 alt char ROM and spec. graphics ^[)2

setss2 SS2            Set single shift 2                     ^[N
setss3 SS3            Set single shift 3                     ^[O

modesoff SGR0         Turn off character attributes          ^[[m
modesoff SGR0         Turn off character attributes          ^[[0m
bold SGR1             Turn bold mode on                      ^[[1m
lowint SGR2           Turn low intensity mode on             ^[[2m
underline SGR4        Turn underline mode on                 ^[[4m
blink SGR5            Turn blinking mode on                  ^[[5m
reverse SGR7          Turn reverse video on                  ^[[7m
invisible SGR8        Turn invisible text mode on            ^[[8m

setwin DECSTBM        Set top and bottom line#s of a window  ^[[;r

cursorup(n) CUU       Move cursor up n lines                 ^[[A
cursordn(n) CUD       Move cursor down n lines               ^[[B
cursorrt(n) CUF       Move cursor right n lines              ^[[C
cursorlf(n) CUB       Move cursor left n lines               ^[[D
cursorhome            Move cursor to upper left corner       ^[[H
cursorhome            Move cursor to upper left corner       ^[[;H
cursorpos(v,h) CUP    Move cursor to screen location v,h     ^[[;H
hvhome                Move cursor to upper left corner       ^[[f
hvhome                Move cursor to upper left corner       ^[[;f
hvpos(v,h) CUP        Move cursor to screen location v,h     ^[[;f
index IND             Move/scroll window up one line         ^[D
revindex RI           Move/scroll window down one line       ^[M
nextline NEL          Move to next line                      ^[E
savecursor DECSC      Save cursor position and attributes    ^[7
restorecursor DECSC   Restore cursor position and attributes ^[8

tabset HTS            Set a tab at the current column        ^[H
tabclr TBC            Clear a tab at the current column      ^[[g
tabclr TBC            Clear a tab at the current column      ^[[0g
tabclrall TBC         Clear all tabs                         ^[[3g

dhtop DECDHL          Double-height letters, top half        ^[#3
dhbot DECDHL          Double-height letters, bottom half     ^[#4
swsh DECSWL           Single width, single height letters    ^[#5
dwsh DECDWL           Double width, single height letters    ^[#6

cleareol EL0          Clear line from cursor right           ^[[K
cleareol EL0          Clear line from cursor right           ^[[0K
clearbol EL1          Clear line from cursor left            ^[[1K
clearline EL2         Clear entire line                      ^[[2K

cleareos ED0          Clear screen from cursor down          ^[[J
cleareos ED0          Clear screen from cursor down          ^[[0J
clearbos ED1          Clear screen from cursor up            ^[[1J
clearscreen ED2       Clear entire screen                    ^[[2J

devstat DSR           Device status report                   ^[5n
termok DSR               Response: terminal is OK            ^[0n
termnok DSR              Response: terminal is not OK        ^[3n

getcursor DSR         Get cursor position                    ^[6n
cursorpos CPR            Response: cursor is at v,h          ^[;R

ident DA              Identify what terminal type            ^[[c
ident DA              Identify what terminal type (another)  ^[[0c
gettype DA               Response: terminal type code n      ^[[?1;0c

reset RIS             Reset terminal to initial state        ^[c

align DECALN          Screen alignment display               ^[#8
```

```
testpu DECTST         Confidence power up test              ^[[2;1y
testlb DECTST         Confidence loopback test              ^[[2;2y
testpurep DECTST      Repeat power up test                  ^[[2;9y
testlbrep DECTST      Repeat loopback test                  ^[[2;10y

ledsoff DECLL0        Turn off all four leds                ^[[0q
led1 DECLL1           Turn on LED #1                        ^[[1q
led2 DECLL2           Turn on LED #2                        ^[[2q
led3 DECLL3           Turn on LED #3                        ^[[3q
led4 DECLL4           Turn on LED #4                        ^[[4q

#
#  All codes below are for use in VT52 compatibility mode.
#

setansi               Enter/exit ANSI mode (VT52)           ^[<

altkeypad             Enter alternate keypad mode           ^[=
numkeypad             Exit alternate keypad mode            ^[>

setgr                 Use special graphics character set    ^[F
resetgr               Use normal US/UK character set        ^[G

cursorup              Move cursor up one line               ^[A
cursordn              Move cursor down one line             ^[B
cursorrt              Move cursor right one char            ^[C
cursorlf              Move cursor left one char             ^[D
cursorhome            Move cursor to upper left corner      ^[H
cursorpos(v,h)        Move cursor to v,h location           ^[
revindex              Generate a reverse line-feed          ^[I

cleareol              Erase to end of current line          ^[K
cleareos              Erase to end of screen                ^[J

ident                 Identify what the terminal is         ^[Z
identresp             Correct response to ident              ^[/Z

-------------------------------------------------------------------------------------------------------------
-----


#
# VT100 Special Key Codes
#
# These are sent from the terminal back to the computer when the
# particular key is pressed.  Note that the numeric keypad keys
# send different codes in numeric mode than in alternate mode.
# See escape codes above to change keypad mode.
#

# Function Keys:

PF1             ^[OP
PF2             ^[OQ
PF3             ^[OR
PF4             ^[OS


# Arrow Keys:
                Reset    Set
                -----    ---
up              ^[A     ^[OA
down            ^[B     ^[OB
right           ^[C     ^[OC
left            ^[D     ^[OD


# Numeric Keypad Keys:

                Keypad Mode
                -----------------
Keypad Key      Numeric Alternate
----------      ------- ---------
0               0       ^[Op
```

1	1	^[Oq
2	2	^[Or
3	3	^[Os
4	4	^[Ot
5	5	^[Ou
6	6	^[Ov
7	7	^[Ow
8	8	^[Ox
9	9	^[Oy
- (minus)	-	^[Om
, (comma)	,	^[Ol
. (period)	.	^[On
ENTER	^M	^[OM

Last Modified: 10^th September 1997

First written by Elliott Hughes

HTML tidied up by Oliver Kiddle

<opk101@york.ac.uk>

---

http://www.termsys.demon.co.uk/vtansi.htm

ANSI/VT100 Terminal Control Escape Sequences

Many computer terminals and terminal emulators support colour and cursor control through a system of escape sequences. One such standard is commonly referred to as ANSI Colour. Several terminal specifications are based on the ANSI colour standard, including VT100.

The following is a partial listing of the VT100 control set.

<ESC> represents the ASCII "escape" character, 0x1B. Bracketed tags represent modifiable decimal parameters; eg. {ROW} would be replaced by a row number.

Device Status

The following codes are used for reporting terminal/display settings, and vary depending on the implementation:

Query Device Code `<ESC>[c`
- Requests a Report Device Code response from the device.

Report Device Code `<ESC>[{code}0c`
- Generated by the device in response to Query Device Code request.

Query Device Status `<ESC>[5n`
- Requests a Report Device Status response from the device.

Report Device OK `<ESC>[0n`
- Generated by the device in response to a Query Device Status request; indicates that device is functioning correctly.

Report Device Failure `<ESC>[3n`
- Generated by the device in response to a Query Device Status request; indicates that device is functioning improperly.

Query Cursor Position `<ESC>[6n`
- Requests a Report Cursor Position response from the device.

Report Cursor Position `<ESC>[{ROW};{COLUMN}R`
- Generated by the device in response to a Query Cursor Position request; reports current cursor position.

Terminal Setup

The h and l codes are used for setting terminal/display mode, and vary depending on the implementation. Line Wrap is one of the few setup codes that tend to be used consistently:

Reset Device <ESC>c
  * Reset all terminal settings to default.

Enable Line Wrap <ESC>[7h
  * Text wraps to next line if longer than the length of the display area.

Disable Line Wrap <ESC>[7l
  * Disables line wrapping.

Fonts

Some terminals support multiple fonts: normal/bold, swiss/italic, etc. There are a variety of special codes for certain terminals; the following are fairly standard:

Font Set G0 <ESC>(
  * Set default font.

Font Set G1 <ESC>)
  * Set alternate font.

Cursor Control

Cursor Home <ESC>[{ROW};{COLUMN}H
  * Sets the cursor position where subsequent text will begin. If no row/column parameters are provided (ie. <ESC>[H), the cursor will move to the home position, at the upper left of the screen.

Cursor Up <ESC>[{COUNT}A
  * Moves the cursor up by COUNT rows; the default count is 1.

Cursor Down <ESC>[{COUNT}B
  * Moves the cursor down by COUNT rows; the default count is 1.

Cursor Forward <ESC>[{COUNT}C
  * Moves the cursor forward by COUNT columns; the default count is 1.

Cursor Backward <ESC>[{COUNT}D
  * Moves the cursor backward by COUNT columns; the default count is 1.

Force Cursor Position <ESC>[{ROW};{COLUMN}f
  * Identical to Cursor Home.

Save Cursor <ESC>[s
  * Save current cursor position.

Unsave Cursor <ESC>[u
  * Restores cursor position after a Save Cursor.

Save Cursor & Attrs <ESC>7
  * Save current cursor position.

Restore Cursor & Attrs <ESC>8
  * Restores cursor position after a Save Cursor.

Scrolling

Scroll Screen <ESC>[r
  * Enable scrolling for entire display.

Scroll Screen <ESC>[{start};{end}r
  * Enable scrolling from row {start} to row {end}.

Scroll Down <ESC>D
  * Scroll display down one line.

Scroll Up <ESC>M
  * Scroll display up one line.

Tab Control

Set Tab <ESC>H
* Sets a tab at the current position.

Clear Tab <ESC>[g
* Clears tab at the current position.

Clear All Tabs <ESC>[3g
* Clears all tabs.

Erasing Text

Erase End of Line <ESC>[K
* Erases from the current cursor position to the end of the current line.

Erase Start of Line <ESC>[1K
* Erases from the current cursor position to the start of the current line.

Erase Line <ESC>[2K
* Erases the entire current line.

Erase Down <ESC>[J
* Erases the screen from the current line down to the bottom of the screen.

Erase Up <ESC>[1J
* Erases the screen from the current line up to the top of the screen.

Erase Screen <ESC>[2J
* Erases the screen with the background colour and moves the cursor to home.

Printing

Some terminals support local printing:
Print Screen <ESC>[i
* Print the current screen.

Print Line <ESC>[1i
* Print the current line.

Stop Print Log <ESC>[4i
* Disable log.

Start Print Log <ESC>[5i
* Start log; all received text is echoed to a printer.

Define Key

Set Key Definition <ESC>[{key};"{string}"p
* Associates a string of text to a keyboard key. {key} indicates the key by its ASCII value in decimal.

Set Display Attributes

Set Attribute Mode <ESC>[{attr1};...;{attrn}m
* Sets multiple display attribute settings. The following lists standard attributes:

0 Reset all attributes
1 Bright
2 Dim
4 Underscore
5 Blink
7 Reverse
8 Hidden

Foreground Colours
30 Black
31 Red
32 Green
33 Yellow
34 Blue
35 Magenta

36	Cyan
37	White

	Background Colours
40	Black
41	Red
42	Green
43	Yellow
44	Blue
45	Magenta
46	Cyan
47	White

---
http://wiki.bash-hackers.org/scripting/terminalcodes

scripting:terminalcodes

Terminal codes (ANSI/VT100) introduction
Terminal (control) codes are used to issue specific commands to your terminal. This can be related to switching colors or positioning the cursor, i.e. anything that can't be done by the application itself.

How it technically works

A terminal control code is a special sequence of characters that is printed (like any other text). If the terminal understands the code, it won't display the character-sequence, but will perform some action. You can
print the codes with a simple echo command.

Note: I see codes referenced as "Bash colors" sometimes (several "Bash tutorials" etc…): That's a completely
incorrect definition.

The tput command
Because there's a large number of different terminal control languages, usually a system has an intermediate
communication layer. The real codes are looked up in a database for the currently detected terminal type and
you give standardized requests to an API or (from the shell) to a command.

One of these commands is tput. Tput accepts a set of acronyms called capability names and any parameters, if
appropriate, then looks up the correct escape sequences for the detected terminal in the terminfo database
and prints the correct codes (the terminal hopefully understands).

The codes
In this list I'll focus on ANSI/VT100 control codes for the most common actions - take it as quick reference.
The documentation of your terminal or the terminfo database is always the preferred source when something is
unclear! Also the tput acronyms are usually the ones dedicated for ANSI escapes!

I listed only the most relevant codes, of course, any ANSI terminal understands many more! But let's keep the
discussion centered on common shell scripting ;-)

If I couldn't find a matching ANSI escape, you'll see a :?: as the code. Feel free to mail me or fix it.

The ANSI codes always start with the ESC character. (ASCII 0x1B or octal 033) This isn't part of the list,
but you should avoid using the ANSI codes directly - use the tput command!

All codes that can be used with tput can be found in terminfo(5). (on OpenBSD at least) See OpenBSD's terminfo(5) under the Capabilities section. The cap-name is the code to use with tput. A description of each
code is also provided.

General useful ASCII codes

The Ctrl-Key representation is simply associating the non-printable characters from ASCII code 1 with the printable (letter) characters from ASCII code 65 ("A"). ASCII code 1 would be ^A (Ctrl-A), while ASCII code 7

(BEL) would be ^G (Ctrl-G). This is a common representation (and input method) and historically comes from
one of the VT series of terminals.

Name	decimal	octal	hex	C-escape	Ctrl-Key	Description
BEL	7	007	0x07	\a	^G	Terminal bell
BS	8	010	0x08	\b	^H	Backspace
HT	9	011	0x09	\t	^I	Horizontal TAB
LF	10	012	0x0A	\n	^J	Linefeed (newline)
VT	11	013	0x0B	\v	^K	Vertical TAB
FF	12	014	0x0C	\f	^L	Formfeed (also: New page NP)
CR	13	015	0x0D	\r	^M	Carriage return
ESC	27	033	0x1B	<none>	^[	Escape character
DEL	127	177	0x7F	<none>	<none>	Delete character

Cursor handling

ANSI	terminfo equivalent	Description
[<X> ; <Y> H		Home-positioning to X and Y coordinates
[<X> ; <Y> f	cup <X> <Y>	it seems that ANSI uses 1-1 as home while tput uses 0-0
[H	home	Move cursor to home position (0-0)
7	sc	Save current cursor position
8	rc	Restore saved cursor position
most likely a normal code like \b	cub1	move left one space (backspace)
VT100 [? 25 l	civis	make cursor invisible
VT100 [? 25 h	cvvis	make cursor visible

Erasing text

ANSI	terminfo equivalent	Description
[K	el	Clear line from current cursor position to end of line
[0 K		
[1 K	el1	Clear line from beginning to current cursor position
[2 K	el2	Clear whole line (cursor position unchanged)

General text attributes

ANSI	terminfo equivalent	Description
[0 m	sgr0	Reset all attributes
[1 m	bold	Set "bright" attribute
[2 m	dim	Set "dim" attribute
[3 m	smso	Set "standout" attribute
[4 m	set smul unset rmul	Set "underscore" (underlined text) attribute
[5 m	blink	Set "blink" attribute
[7 m	rev	Set "reverse" attribute
[8 m	invis	Set "hidden" attribute

Foreground coloring

ANSI	terminfo equivalent	Description
[3 0 m	setaf 0	Set foreground to color #0 - black
[3 1 m	setaf 1	Set foreground to color #1 - red
[3 2 m	setaf 2	Set foreground to color #2 - green
[3 3 m	setaf 3	Set foreground to color #3 - yellow
[3 4 m	setaf 4	Set foreground to color #4 - blue
[3 5 m	setaf 5	Set foreground to color #5 - magenta
[3 6 m	setaf 6	Set foreground to color #6 - cyan
[3 7 m	setaf 7	Set foreground to color #7 - white
[3 9 m	setaf 9	Set default color as foreground color

Background coloring

ANSI	terminfo equivalent	Description
[4 0 m	setab 0	Set background to color #0 - black
[4 1 m	setab 1	Set background to color #1 - red
[4 2 m	setab 2	Set background to color #2 - green
[4 3 m	setab 3	Set background to color #3 - yellow
[4 4 m	setab 4	Set background to color #4 - blue
[4 5 m	setab 5	Set background to color #5 - magenta
[4 6 m	setab 6	Set background to color #6 - cyan
[4 7 m	setab 7	Set background to color #7 - white
[4 9 m	setaf 9	Set default color as background color

Misc codes
Save/restore screen

Used capabilities: smcup, rmcup

You've undoubtedly already encountered programs that restore the terminal contents after they do their work
(like vim). This can be done by the following commands:

```
# save, clear screen
tput smcup
clear

# example "application" follows...
read -n1 -p "Press any key to continue..."
# example "application" ends here

# restore
tput rmcup
```

These features require that certain capabilities exist in your termcap/terminfo. While xterm and most of its
clones (rxvt, urxvt, etc) will support the instructions, your operating system may not include references to
them in its default xterm profile. (FreeBSD, in particular, falls into this category.) If `tput smcup` appears to do nothing for you, and you don't want to modify your system termcap/terminfo data, and you KNOW
that you are using a compatible xterm application, the following may work for you:

```
echo -e '\033[?47h' # save screen
echo -e '\033[?47l' # restore screen
```

Certain software uses these codes (via their termcap capabilities) as well. You may have seen the screen save
/restore in less, vim, top, screen and others. Some of these applications may also provide configuration options to *disable* this behaviour. For example, less has a -X option for this, which can also be set in an
environment variable:

```
export LESS=X
less /path/to/file
```

Similarly, vim can be configured not to "restore" the screen by adding the following to your ~/.vimrc:

```
set t_ti= t_te=
```

Additional colors

Some terminal emulators support additional colors. The most common extension used by xterm-compatible terminals supports 256 colors. These can be generated by tput with seta{f,b} [0-255] when the TERM value has
a -256color suffix. Some terminals also support full 24-bit colors, and any X11 color code can be written directly into a special escape sequence. (More infos) Only a few programs make use of anything beyond 256 colors, and tput doesn't know about them. Colors beyond 16 usually only apply to modern terminal emulators
running in graphical environments.

The Virtual Terminal implemented in the Linux kernel supports only 16 colors, and the usual default terminfo
entry for TERM=linux defines only 8. There is sometimes an alternate "linux-16color" that you can switch to,
to get the other 8 colors.

Bash examples

Hardcoded colors

```
printf '%b\n' 'It is \033[31mnot\033[39m intelligent to use \033[32mhardcoded ANSI\033[39m codes!'
```

Colors using tput

Directly inside the echo:

```
echo "TPUT is a $(tput setaf 2)nice$(tput setaf 9) and $(tput setaf 5)user friendly$(tput setaf 9) terminal capability database."
```

With preset variables:

```
COL_NORM="$(tput setaf 9)"
COL_RED="$(tput setaf 1)"
COL_GREEN="$(tput setaf 2)"
echo "It's ${COL_RED}red${COL_NORM} and ${COL_GREEN}green${COL_NORM} - have you seen?"
```

Misc
HOME function

```
home() {
  # yes, actually not much shorter ;-)
  tput home
}
```

Silly but nice effect

```
#!/bin/bash

DATA[0]="     _/  _/    _/                            _/    "
DATA[1]="  _/_/_/_/_/  _/_/_/      _/_/_/    _/_/_/  _/_/_/ "
DATA[2]="   _/  _/    _/    _/  _/    _/  _/_/      _/    _/"
DATA[3]="_/_/_/_/_/  _/    _/  _/    _/      _/_/  _/    _/ "
DATA[4]=" _/  _/    _/_/_/      _/_/_/  _/_/_/    _/    _/  "

# virtual coordinate system is X*Y ${#DATA} * 5

REAL_OFFSET_X=0
REAL_OFFSET_Y=0

draw_char() {
  V_COORD_X=$1
  V_COORD_Y=$2

  tput cup $((REAL_OFFSET_Y + V_COORD_Y)) $((REAL_OFFSET_X + V_COORD_X))

  printf %c ${DATA[V_COORD_Y]:V_COORD_X:1}
}


trap 'exit 1' INT TERM
trap 'tput setaf 9; tput cvvis; clear' EXIT

tput civis
clear

while :; do

for ((c=1; c <= 7; c++)); do
  tput setaf $c
  for ((x=0; x<${#DATA[0]}; x++)); do
    for ((y=0; y<=4; y++)); do
      draw_char $x $y
    done
  done
done

done
```

Mandelbrot set
This is a slightly modified version of Charles Cooke's colorful Mandelbrot plot scripts (original w/ screenshot) – ungolfed, optimized a bit, and without hard-coded terminal escapes. The colorBox function is
memoized to collect tput output only when required and output a new escape only when a color change is needed. This limits the number of tput calls to at most 16, and reduces raw output by more than half. The doBash function uses integer arithmetic, but is still ksh93-compatible (run as e.g. bash ./mandelbrot to use
it). The ksh93-only floating-point doKsh is almost 10x faster than doBash (thus the ksh shebang by default),
but uses only features that don't make the Bash parser crash.

```
#!/usr/bin/env ksh

# Charles Cooke's 16-color Mandelbrot
# http://earth.gkhs.net/ccooke/shell.html
# Combined Bash/ksh93 flavors by Dan Douglas (ormaaj)

function doBash {
        typeset P Q X Y a b c i v x y
        for ((P=10**8,Q=P/100,X=320*Q/cols,Y=210*Q/lines,y=-105*Q,v=-220*Q,x=v;y<105*Q;x=v,y+=Y)); do
```

```
                for ((;x<P;a=b=i=c=0,x+=X)); do
                        for ((;a**2+b**2<4*P**2&&i++<99;a=((c=a)**2-b**2)/P+x,b=2*c*b/P+y)); do :
                        done
                        colorBox $((i<99?i%16:0))
                done
                echo
        done
}

function doKsh {
        integer i
        float a b c x=2.2 y=-1.05 X=3.2/cols Y=2.1/lines
        while
                for ((a=b=i=0;(c=a)**2+b**2<=2&&i++<99&&(a=a**2-b**2+x,b=2*c*b+y);)); do :
                done
                . colorBox $((i<99?i%16:0))
                if ((x<1?!(x+=X):(y+=Y,x=-2.2))); then
                        print
                        ((y<1.05))
                fi
                do :
        done
}

function colorBox {
        (($1==lastclr)) || printf %s "${colrs[lastclr=$1]:=$(tput setaf "$1")}"
        printf '\u2588'
}

unset -v lastclr
((cols=$(tput cols)-1, lines=$(tput lines)))
typeset -a colrs
trap 'tput sgr0; echo' EXIT
${KSH_VERSION+. doKsh} ${BASH_VERSION+doBash}
```

Discussion
Lucas H, 2011/07/28 06:37

RE: your "Note: I found no code to entirely erase the current line ("delete line" is something else!). It might be a combination of positioning the cursor and erase to the end of line."

Try this: [2 K el2 Clear whole line

James, 2011/09/01 00:12

In the table showing

[3 9 m setaf 9 Set default foreground color

the Description "Set default foreground color" is ambiguous.

That phrase could mean either that the commands will 1) store the value of a specified color as the "default"
color value, or that 2) a stored "default" color value will be used to re-set the current foreground or background color to a new value. Which is it? In one case there can be a visible change on the screen. In the
other case, the will never be a visible change on the screen.

As it is, some people will create termcap files which gratuitously reset the display to the "default" colors,
which makes using custom foreground and background colors impossible. Of course, this is just mean, and requires rewriting the termcap file.

Also, the Descriptions of the "Dim", "Bright", and "Reverse" attributes could actually say what these are suppose to do. For instance, what is suppose to happen when setting both "Dim" and "Bright"? Or, does "Reverse" apply to both the foreground and background colors? Does "Reverse" mean to exchange the foreground
and background colors? Or to set some kind of "complement" color to each of the foreground and background?

These "Descriptions" that do not describe are not useful.

James

Constantine, 2011/09/21 14:43

print shortcuts for all ansi codes, NB: please add plus plus in for statements!

```
      ansi-test()
      {
      for a in 0 1 4 5 7; do
              echo "a=$a "
              for (( f=0; f<=9; f++ )) ; do
                      for (( b=0; b<=9; b++ )) ; do
                              #echo -ne "f=$f b=$b"
                              echo -ne "\\033[${a};3${f};4${b}m"
                              echo -ne "\\\\\\\\\033[${a};3${f};4${b}m"
                              echo -ne "\\033[0m "
                      done
              echo
              done
              echo
      done
      echo
      }
```

Aubrey Bourke, 2011/12/19 02:38

Hi,

Very cool tutorial. I recently purchased a beagleboard XM, so this site is a perfect place to start serial
port programming.

And the "silly but nice effects" is awesome. I love it!

Here's a link to a "cool splash screen for my website". Its just a Java animation... (open with Java web start - jws)

file:///home/aubrey/Downloads/circlescroller.jnlp

Best Regards.

Jan Schampera, 2011/12/21 12:35, 2011/12/21 12:36

Hi,

thank you :-)

I don't think this link will work for anybody except you (file:) :-)

Bill Gradwohl, 2012/04/08 01:57

This describes things from the display end. What about the keyboard? How does someone read the codes from the
keyboard and figure out that the user pressed the up arrow key, for example?

I'm interested in this for using the bash read -s -n 1 mechanism to bring in keystrokes 1 character at a time
and then try to figure out what key the user pressed. Up arrow for example is \E[A . I want to get the entire
list of possible character combinations that are legitimate for a given environment.

The infocmp utility can dump the terminfo for a particular entity (xterm, linux, etc) but I can't find the
equivalent for a keybaord.

Jan Schampera, 2012/04/21 12:45

A very good question. Sorry, I can't answer it. I think there are no such things as "standardized" key codes.

Ruthard Baudach, 2013/04/03 21:35, 2014/10/06 06:25

well, just use read!

read does not only read the input from the keyboard, but reflects it on the terminal – resulting in the

keycodes you are looking for.

I used my findings for the following python script: (sorry for not using bash)

```
#!/usr/bin/env python

import sys,termios, time

## Font attributes ##
# off
off = '\x1b[0m' # off
default = '\x1b[39m' # default foreground
DEFAULT = '\x1b[49m' # default background
#
bd = '\x1b[1m' # bold
ft = '\x1b[2m' # faint
st = '\x1b[3m' # standout
ul = '\x1b[4m' # underlined
bk = '\x1b[5m' # blink
rv = '\x1b[7m' # reverse
hd = '\x1b[8m' # hidden
nost = '\x1b[23m' # no standout
noul = '\x1b[24m' # no underlined
nobk = '\x1b[25m' # no blink
norv = '\x1b[27m' # no reverse
# colors
black = '\x1b[30m'
BLACK = '\x1b[40m'
red = '\x1b[31m'
RED = '\x1b[41m'
green = '\x1b[32m'
GREEN = '\x1b[42m'
yellow = '\x1b[33m'
YELLOW = '\x1b[43m'
blue = '\x1b[34m'
BLUE = '\x1b[44m'
magenta = '\x1b[35m'
MAGENTA = '\x1b[45m'
cyan = '\x1b[36m'
CYAN = '\x1b[46m'
white = '\x1b[37m'
WHITE = '\x1b[47m'
# light colors
dgray = '\x1b[90m'
DGRAY = '\x1b[100m'
lred = '\x1b[91m'
LRED = '\x1b[101m'
lgreen = '\x1b[92m'
LGREEN = '\x1b[102m'
lyellow = '\x1b[93m'
LYELLOW = '\x1b[103m'
lblue = '\x1b[94m'
LBLUE = '\x1b[104m'
lmagenta = '\x1b[95m'
LMAGENTA = '\x1b[105m'
lcyan = '\x1b[96m'
LCYAN = '\x1b[106m'
lgray = '\x1b[97m'
LGRAY = '\x1b[107m'

## 256 colors ##
# \x1b[38;5;#m foreground, # = 0 - 255
# \x1b[48;5;#m background, # = 0 - 255
## True Color ##
# \x1b[38;2;r;g;bm r = red, g = green, b = blue foreground
# \x1b[48;2;r;g;bm r = red, g = green, b = blue background

# ------------------------------------------------------------------------------
# prepare terminal settings
fd = sys.stdin.fileno()
old_settings = termios.tcgetattr(fd)
new_settings = termios.tcgetattr(fd)
new_settings[3] &= ~termios.ICANON
new_settings[3] &= ~termios.ECHO
```

```
# --------------------------------------------------------------------------------
def clear(what='screen'):
        '''
        erase functions:
                what: screen => erase screen and go home
                      line   => erase line and go to start of line
                      bos    => erase to begin of screen
                      eos    => erase to end of screen
                      bol    => erase to begin of line
                      eol    => erase to end of line
        '''
        clear = {
                'screen': '\x1b[2J\x1b[H',
                'line': '\x1b[2K\x1b[G',
                'bos': '\x1b[1J',
                'eos': '\x1b[J',
                'bol': '\x1b[1K',
                'eol': '\x1b[K',
                }
        sys.stdout.write(clear[what])
        sys.stdout.flush()

# --------------------------------------------------------------------------------
def move(pos):
        '''
        move cursor to pos
        pos = tuple (x,y)
        '''
        x,y = pos
        sys.stdout.write('\x1b[{};{}H'.format(str(x),str(y)))
        sys.stdout.flush()

# --------------------------------------------------------------------------------
def put(*args):
        '''
        put text on on screen
        a tuple as first argument tells absolute position for the text
       does not change cursor position
        args = list of optional position, formatting tokens and strings
        '''
        args = list(args)
        if type(args[0]) == type(()):
                x,y = args[0]
                del args[0]
                args.insert(0,'\x1b[{};{}H'.format(str(x),str(y)))
        args.insert(0,'\x1b[s')
        args.append('\x1b[u')
        sys.stdout.write(''.join(args))
        sys.stdout.flush()

# --------------------------------------------------------------------------------
def write(*args):
        '''
        writes text on on screen
        a tuple as first argument gives the relative position to current cursor position
        does change cursor position
        args = list of optional position, formatting tokens and strings
        '''
        args = list(args)
        if type(args[0]) == type(()):
                pos = []
                x,y = args[0]
                if x > 0:
                        pos.append('\x1b[{}A'.format(str(x)))
                elif x < 0:
                        pos.append('\x1b[{}B'.format(abs(str(x))))
                if y > 0:
                        pos.append('\x1b[{}C'.format(str(y)))
                elif y < 0:
                        pos.append('\x1b[{}D'.format(abs(str(y))))
                del args[0]
                args = pos + args
        sys.stdout.write(''.join(args))
```

```
        sys.stdout.flush()


# --------------------------------------------------------------------------------
def getch():
        '''
        Get character.
        '''
        # get character
        try:
                termios.tcsetattr(fd,termios.TCSANOW,new_settings)
                ch = sys.stdin.read(1)
        finally:
                termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        # return
        return ch

# --------------------------------------------------------------------------------


if __name__ == '__main__':
        clear()
        esc_mode = False
        esc_string = ''
        esc_codes = {
                '[A': 'Up',
                '[B': 'Down',
                '[C': 'Right',
                '[D': 'Left',
                '[F': 'End',
                '[H': 'Pos1',
                '[2~': 'Ins',
                '[3~': 'Del',
                '[5~': 'PgUp',
                '[6~': 'PdDown',
                'OP': 'F1',
                'OQ': 'F2',
                'OR': 'F3',
                'OS': 'F4',
                '[15~': 'F5',
                '[17~': 'F6',
                '[18~': 'F7',
                '[19~': 'F8',
                '[20~': 'F9',
                '[21~': 'F10',
                '[23~': 'F11',
                '[24~': 'F12',
                '[29~': 'Apps',
                '[34~': 'Win',
                '[1;2A': 'S-Up',
                '[1;2B': 'S-Down',
                '[1;2C': 'S-Right',
                '[1;2D': 'S-Left',
                '[1;2F': 'S-End',
                '[1;2H': 'S-Pos1',
                '[2;2~': 'S-Ins',
                '[3;2~': 'S-Del',
                '[5;2~': 'S-PgUp',
                '[6;2~': 'S-PdDown',
                '[1;2P': 'S-F1',
                '[1;2Q': 'S-F2',
                '[1;2R': 'S-F3',
                '[1;2S': 'S-F4',
                '[15;2~': 'S-F5',
                '[17;2~': 'S-F6',
                '[18;2~': 'S-F7',
                '[19;2~': 'S-F8',
                '[20;2~': 'S-F9',
                '[21;2~': 'S-F10',
                '[23;2~': 'S-F11',
                '[24;2~': 'S-F12',
                '[29;2~': 'S-Apps',
                '[34;2~': 'S-Win',
                '[1;3A': 'M-Up',
```

```
                    '[1;3B': 'M-Down',
'[1;3C': 'M-Right',
'[1;3D': 'M-Left',
'[1;3F': 'M-End',
'[1;3H': 'M-Pos1',
'[2;3~': 'M-Ins',
'[3;3~': 'M-Del',
'[5;3~': 'M-PgUp',
'[6;3~': 'M-PdDown',
'[1;3P': 'M-F1',
'[1;3Q': 'M-F2',
'[1;3R': 'M-F3',
'[1;3S': 'M-F4',
'[15;3~': 'M-F5',
'[17;3~': 'M-F6',
'[18;3~': 'M-F7',
'[19;3~': 'M-F8',
'[20;3~': 'M-F9',
'[21;3~': 'M-F10',
'[23;3~': 'M-F11',
'[24;3~': 'M-F12',
'[29;3~': 'M-Apps',
'[34;3~': 'M-Win',
'[1;5A': 'C-Up',
'[1;5B': 'C-Down',
'[1;5C': 'C-Right',
'[1;5D': 'C-Left',
'[1;5F': 'C-End',
'[1;5H': 'C-Pos1',
'[2;5~': 'C-Ins',
'[3;5~': 'C-Del',
'[5;5~': 'C-PgUp',
'[6;5~': 'C-PdDown',
'[1;5P': 'C-F1',
'[1;5Q': 'C-F2',
'[1;5R': 'C-F3',
'[1;5S': 'C-F4',
'[15;5~': 'C-F5',
'[17;5~': 'C-F6',
'[18;5~': 'C-F7',
'[19;5~': 'C-F8',
'[20;5~': 'C-F9',
'[21;5~': 'C-F10',
'[23;5~': 'C-F11',
'[24;5~': 'C-F12',
'[29;5~': 'C-Apps',
'[34;5~': 'C-Win',
'[1;6A': 'S-C-Up',
'[1;6B': 'S-C-Down',
'[1;6C': 'S-C-Right',
'[1;6D': 'S-C-Left',
'[1;6F': 'S-C-End',
'[1;6H': 'S-C-Pos1',
'[2;6~': 'S-C-Ins',
          '[3;6~': 'S-C-Del',
'[5;6~': 'S-C-PgUp',
'[6;6~': 'S-C-PdDown',
'[1;6P': 'S-C-F1',
'[1;6Q': 'S-C-F2',
'[1;6R': 'S-C-F3',
'[1;6S': 'S-C-F4',
'[15;6~': 'S-C-F5',
'[17;6~': 'S-C-F6',
'[18;6~': 'S-C-F7',
'[19;6~': 'S-C-F8',
'[20;6~': 'S-C-F9',
'[21;6~': 'S-C-F10',
'[23;6~': 'S-C-F11',
'[24;6~': 'S-C-F12',
'[29;6~': 'S-C-Apps',
'[34;6~': 'S-C-Win',
'[1;7A': 'C-M-Up',
'[1;7B': 'C-M-Down',
'[1;7C': 'C-M-Right',
```

```
        '[1;7D': 'C-M-Left',
        '[1;7F': 'C-M-End',
        '[1;7H': 'C-M-Pos1',
        '[2;7~': 'C-M-Ins',
        '[3;7~': 'C-M-Del',
        '[5;7~': 'C-M-PgUp',
        '[6;7~': 'C-M-PdDown',
        '[1;7P': 'C-M-F1',
        '[1;7Q': 'C-M-F2',
        '[1;7R': 'C-M-F3',
        '[1;7S': 'C-M-F4',
        '[15;7~': 'C-M-F5',
        '[17;7~': 'C-M-F6',
        '[18;7~': 'C-M-F7',
        '[19;7~': 'C-M-F8',
        '[20;7~': 'C-M-F9',
        '[21;7~': 'C-M-F10',
        '[23;7~': 'C-M-F11',
        '[24;7~': 'C-M-F12',
        '[29;7~': 'C-M-Apps',
        '[34;7~': 'C-M-Win',
        }
        # 8 wäre S-C-M

ctrl_codes = {
        0: 'C-2',
        1: 'C-A',
        2: 'C-B',
        3: 'C-C',
        4: 'C-D',
        5: 'C-E',
        6: 'C-F',
        7: 'C-G',
        8: 'C-H',
               9: 'C-I',
        10: 'C-J',
        11: 'C-K',
        12: 'C-L',
        13: 'C-M',
        14: 'C-N',
        15: 'C-O',
        16: 'C-P',
        17: 'C-Q',
        18: 'C-R',
        19: 'C-S',
        20: 'C-T',
        21: 'C-U',
        22: 'C-V',
        23: 'C-W',
        24: 'C-X',
        25: 'C-Y',
        26: 'C-Z',
        27: 'C-3',
        29: 'C-5',
        30: 'C-6',
        31: 'C-7',
        }

while True:
        move((1,1))
        clear('line')
        put((1,1),green,':',off)
        move((1,3))
        ch = getch()
        if esc_mode:
                esc_string += ch
                # esc string terminators
                if ch in ['A','B','C','D','F','H','P','Q','R','S','~']:
                        esc_mode = False
                        move((2,0))
                        clear('line')
                        put((2,5),esc_codes[esc_string])
                        esc_string = ''
                elif ch == '\x1b':
```

```
                        esc_mode = False
        else:
                # esc mode
                if ch == '\x1b':
                        esc_mode = True
                # ctrl
                elif ord(ch) in ctrl_codes.keys():
                        move((2,0))
                        clear('line')
                        put((2,5),ctrl_codes[ord(ch)])

                move((2,0))
                put((2,3),str(ch))
```

Iskren Hadzhinedev, 2013/11/04 13:33

If you're using X, you can get keycodes from the keyboard with the 'xev' program; it opens a window that prints in the terminal every event (mouse move, mouse button press, keypress, keyrelease, etc). I know I'm
more than a year late, but google brought me here, so hopefully someone will find this useful. Cheers.

JK Benedict, 2014/08/30 05:36

First - thank you for this article as I have written a sub-routine for various *nix and non-nix systems to
parse ANSI (as best as possible). Point is - I WORSHIP THIS OVERVIEW - especially when I come across individuals interested in making the most of bash, etc.

Second - I get to contribute!

@Bill – From bash, leverage the read command. I've included a few links for reference, but the general idea
is that it can be used for "Hey, type in a something and press enter" to being nested in a loop condition to
"trap" (that is a term you will want to look at) single key strokes. The command even goes as far to give a
"timeout" if the user doesn't press any key!

http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_02.html

http://www.unix.com/shell-programming-and-scripting/140231-bash-keypress-read-single-character.html

and this sorta brings the previous links together in a practical example:

http://top-scripts.blogspot.com/2011/01/blog-post.html

HTH! -jkbs @xenfomation

Albert25, 2015/07/10 10:46

Quickly see the foreground/background colors:

```
for b in {0..7} 9; do for f in {0..7} 9; do for attr in "" bold; do echo -e "$(tput setab $b; tput setaf $f; \
[ -n "$attr" ] && tput $attr) $f ON $b $attr $(tput sgr0)"; done; done; done
```

Or the same on several lines for readability:

```
  for b in {0..7} 9; do
      for f in {0..7} 9; do
          for attr in "" bold; do
             echo -e "$(tput setab $b; tput setaf $f; [ -n "$attr" ] && tput $attr) $f ON $b $attr $(tput sgr0)"
          done
      done
  done
```

You could leave a comment if you were logged in.

- scripting/terminalcodes.txt
- Last modified: 2017/09/13 15:26
- by nudin

Except where otherwise noted, content on this wiki is licensed under the following license: GNU Free Documentation License 1.3