NAME

console_codes – Linux console escape and control sequences

DESCRIPTION

The Linux console implements a large subset of the VT102 and ECMA-48/ISO 6429/ANSI X3.64 terminal controls, plus certain private-mode sequences for changing the color palette, character-set mapping, and so on. In the tabular descriptions below, the second column gives ECMA-48 or DEC mnemonics (the latter if prefixed with DEC) for the given function. Sequences without a mnemonic are neither ECMA-48 nor VT102.

After all the normal output processing has been done, and a stream of characters arrives at the console driver for actual printing, the first thing that happens is a translation from the code used for processing to the code used for printing.

If the console is in UTF-8 mode, then the incoming bytes are first assembled into 16-bit Unicode codes. Otherwise, each byte is transformed according to the current mapping table (which translates it to a Unicode value). See the **Character Sets** section below for discussion.

In the normal case, the Unicode value is converted to a font index, and this is stored in video memory, so that the corresponding glyph (as found in video ROM) appears on the screen. Note that the use of Unicode (and the design of the PC hardware) allows us to use 512 different glyphs simultaneously.

If the current Unicode value is a control character, or we are currently processing an escape sequence, the value will treated specially. Instead of being turned into a font index and rendered as a glyph, it may trigger cursor movement or other control functions. See the **Linux Console Controls** section below for discussion.

It is generally not good practice to hard-wire terminal controls into programs. Linux supports a terminfo(5) database of terminal capabilities. Rather than emitting console escape sequences by hand, you will almost always want to use a terminfo-aware screen library or utility such as **ncurses(3),** tput(1), or reset(1).

Linux console controls

This section describes all the control characters and escape sequences that invoke special functions (i.e., anything other than writing a glyph at the current cursor location) on the Linux console.

Control characters

A character is a control character if (before transformation according to the mapping table) it has one of the 14 codes 00 (NUL), 07 (BEL), 08 (BS), 09 (HT), 0a (LF), 0b (VT), 0c (FF), 0d (CR), 0e (SO), 0f (SI), 18 (CAN), 1a (SUB), 1b (ESC), 7f (DEL). One can set a "display control characters" mode (see below), and allow 07, 09, 0b, 18, 1a, 7f to be displayed as glyphs. On the other hand, in UTF-8 mode all codes 00–1f are regarded as control characters, regardless of any "display control characters" mode.

If we have a control character, it is acted upon immediately and then discarded (even in the middle of an escape sequence) and the escape sequence continues with the next character. (However, ESC starts a new escape sequence, possibly aborting a previous unfinished one, and CAN and SUB abort any escape sequence.) The recognized control characters are BEL, BS, HT, LF, VT, FF, CR, SO, SI, CAN, SUB, ESC, DEL, CSI. They do what one would expect:

BEL (0x07, **^G**) beeps;

BS (0x08, **^H**) backspaces one column (but not past the beginning of the line);

HT (0x09, **^I**) goes to the next tab stop or to the end of the line if there is no earlier tab stop;

LF (0x0A, **^J**), VT (0x0B, **^K**) and FF (0x0C, **^L**) all give a linefeed, and if LF/NL (new-line mode) is set also a carriage return;

CR (0x0D, **^M**) gives a carriage return;

SO (0x0E, **^N**) activates the G1 character set;

SI (0x0F, **^O**) activates the G0 character set;

CAN (0x18, **^X**) and SUB (0x1A, **^Z**) interrupt escape sequences;

ESC (0x1B, **^[**) starts an escape sequence;

DEL (0x7F) is ignored;

CSI (0x9B) is equivalent to ESC [.

ESC- but not CSI-sequences

ESC c	RIS	Reset.
ESC D	IND	Linefeed.
ESC E	NEL	Newline.
ESC H	HTS	Set tab stop at current column.
ESC M	RI	Reverse linefeed.
ESC Z	DECID	DEC private identification. The kernel returns the string ESC [? 6 c, claiming that it is a VT102.
ESC 7	DECSC	Save current state (cursor coordinates, attributes, character sets pointed at by G0, G1).
ESC 8	DECRC	Restore state most recently saved by ESC 7.
ESC [	CSI	Control sequence introducer
ESC %		Start sequence selecting character set
ESC % @		Select default (ISO 646 / ISO 8859-1)
ESC % G		Select UTF-8
ESC % 8		Select UTF-8 (obsolete)
ESC # 8	DECALN	DEC screen alignment test – fill screen with E's.
ESC (		Start sequence defining G0 character set
ESC (B		Select default (ISO 8859-1 mapping)
ESC (0		Select VT100 graphics mapping
ESC (U		Select null mapping – straight to character ROM
ESC (K		Select user mapping – the map that is loaded by the utility **mapscrn(8)**.
ESC)		Start sequence defining G1 (followed by one of B, 0, U, K, as above).
ESC >	DECPNM	Set numeric keypad mode
ESC =	DECPAM	Set application keypad mode
ESC]	OSC	(Should be: Operating system command) ESC] P *nrrggbb*: set palette, with parameter given in 7 hexadecimal digits after the final P :-(. Here *n* is the color (0–15), and *rrggbb* indicates the red/green/blue values (0–255). ESC] R: reset palette

ECMA-48 CSI sequences

CSI (or ESC []) is followed by a sequence of parameters, at most NPAR (16), that are decimal numbers separated by semicolons. An empty or absent parameter is taken to be 0. The sequence of parameters may be preceded by a single question mark.

However, after CSI [(or ESC [[]) a single character is read and this entire sequence is ignored. (The idea is to ignore an echoed function key.)

The action of a CSI sequence is determined by its final character.

@	ICH	Insert the indicated # of blank characters.
A	CUU	Move cursor up the indicated # of rows.
B	CUD	Move cursor down the indicated # of rows.
C	CUF	Move cursor right the indicated # of columns.
D	CUB	Move cursor left the indicated # of columns.
E	CNL	Move cursor down the indicated # of rows, to column 1.

F	CPL	Move cursor up the indicated # of rows, to column 1.
G	CHA	Move cursor to indicated column in current row.
H	CUP	Move cursor to the indicated row, column (origin at 1,1).
J	ED	Erase display (default: from cursor to end of display). ESC [1 J: erase from start to cursor. ESC [2 J: erase whole display. ESC [3 J: erase whole display including scroll-back buffer (since Linux 3.0).
K	EL	Erase line (default: from cursor to end of line). ESC [1 K: erase from start of line to cursor. ESC [2 K: erase whole line.
L	IL	Insert the indicated # of blank lines.
M	DL	Delete the indicated # of lines.
P	DCH	Delete the indicated # of characters on current line.
X	ECH	Erase the indicated # of characters on current line.
a	HPR	Move cursor right the indicated # of columns.
c	DA	Answer ESC [? 6 c: "I am a VT102".
d	VPA	Move cursor to the indicated row, current column.
e	VPR	Move cursor down the indicated # of rows.
f	HVP	Move cursor to the indicated row, column.
g	TBC	Without parameter: clear tab stop at current position. ESC [3 g: delete all tab stops.
h	SM	Set Mode (see below).
l	RM	Reset Mode (see below).
m	SGR	Set attributes (see below).
n	DSR	Status report (see below).
q	DECLL	Set keyboard LEDs. ESC [0 q: clear all LEDs ESC [1 q: set Scroll Lock LED ESC [2 q: set Num Lock LED ESC [3 q: set Caps Lock LED
r	DECSTBM	Set scrolling region; parameters are top and bottom row.
s	?	Save cursor location.
u	?	Restore cursor location.
`	HPA	Move cursor to indicated column in current row.

ECMA-48 Set Graphics Rendition

The ECMA-48 SGR sequence ESC [*parameters* m sets display attributes. Several attributes can be set in the same sequence, separated by semicolons. An empty parameter (between semicolons or string initiator or terminator) is interpreted as a zero.

param	result
0	reset all attributes to their defaults
1	set bold
2	set half-bright (simulated with color on a color display)
4	set underscore (simulated with color on a color display) (the colors used to simulate dim or underline are set using ESC] ...)
5	set blink
7	set reverse video
10	reset selected mapping, display control flag, and toggle meta flag (ECMA-48 says "primary font").
11	select null mapping, set display control flag, reset toggle meta flag (ECMA-48 says "first alternate font").

12 select null mapping, set display control flag, set toggle meta flag (ECMA-48 says "second alternate font"). The toggle meta flag causes the high bit of a byte to be toggled before the mapping table translation is done.
21 set normal intensity (ECMA-48 says "doubly underlined")
22 set normal intensity
24 underline off
25 blink off
27 reverse video off
30 set black foreground
31 set red foreground
32 set green foreground
33 set brown foreground
34 set blue foreground
35 set magenta foreground
36 set cyan foreground
37 set white foreground
38 set underscore on, set default foreground color
39 set underscore off, set default foreground color
40 set black background
41 set red background
42 set green background
43 set brown background
44 set blue background
45 set magenta background
46 set cyan background
47 set white background
49 set default background color

ECMA-48 Mode Switches

ESC [3 h
DECCRM (default off): Display control chars.

ESC [4 h
DECIM (default off): Set insert mode.

ESC [20 h
LF/NL (default off): Automatically follow echo of LF, VT or FF with CR.

ECMA-48 Status Report Commands

ESC [5 n
Device status report (DSR): Answer is ESC [0 n (Terminal OK).

ESC [6 n
Cursor position report (CPR): Answer is ESC [*y* ; *x* R, where *x,y* is the cursor location.

DEC Private Mode (DECSET/DECRST) sequences

These are not described in ECMA-48. We list the Set Mode sequences; the Reset Mode sequences are obtained by replacing the final 'h' by 'l'.

ESC [? 1 h
DECCKM (default off): When set, the cursor keys send an ESC O prefix, rather than ESC [.

ESC [? 3 h
DECCOLM (default off = 80 columns): 80/132 col mode switch. The driver sources note that this alone does not suffice; some user-mode utility such as **resizecons(8)** has to change the hardware registers on the console video card.

ESC [? 5 h
DECSCNM (default off): Set reverse-video mode.

ESC [? 6 h
DECOM (default off): When set, cursor addressing is relative to the upper left corner of the scrolling region.

ESC [? 7 h
DECAWM (default on): Set autowrap on. In this mode, a graphic character emitted after column 80 (or column 132 of DECCOLM is on) forces a wrap to the beginning of the following line first.

ESC [? 8 h
DECARM (default on): Set keyboard autorepeat on.

ESC [? 9 h
X10 Mouse Reporting (default off): Set reporting mode to 1 (or reset to 0)—see below.

ESC [? 25 h
DECTECM (default on): Make cursor visible.

ESC [? 1000 h
X11 Mouse Reporting (default off): Set reporting mode to 2 (or reset to 0)—see below.

Linux Console Private CSI Sequences

The following sequences are neither ECMA-48 nor native VT102. They are native to the Linux console driver. Colors are in SGR parameters: 0 = black, 1 = red, 2 = green, 3 = brown, 4 = blue, 5 = magenta, 6 = cyan, 7 = white.

ESC [1 ; *n*]	Set color *n* as the underline color
ESC [2 ; *n*]	Set color *n* as the dim color
ESC [8]	Make the current color pair the default attributes.
ESC [9 ; *n*]	Set screen blank timeout to *n* minutes.
ESC [10 ; *n*]	Set bell frequency in Hz.
ESC [11 ; *n*]	Set bell duration in msec.
ESC [12 ; *n*]	Bring specified console to the front.
ESC [13]	Unblank the screen.
ESC [14 ; *n*]	Set the VESA powerdown interval in minutes.
ESC [15]	Bring the previous console to the front (since Linux 2.6.0).
ESC [16 ; *n*]	Set the cursor blink interval in milliseconds (since Linux 4.2)

Character sets

The kernel knows about 4 translations of bytes into console-screen symbols. The four tables are: a) Latin1 -> PC, b) VT100 graphics -> PC, c) PC -> PC, d) user-defined.

There are two character sets, called G0 and G1, and one of them is the current character set. (Initially G0.) Typing **^N** causes G1 to become current, **^O** causes G0 to become current.

These variables G0 and G1 point at a translation table, and can be changed by the user. Initially they point at tables a) and b), respectively. The sequences ESC (B and ESC (0 and ESC (U and ESC (K cause G0 to point at translation table a), b), c) and d), respectively. The sequences ESC) B and ESC) 0 and ESC) U and ESC) K cause G1 to point at translation table a), b), c) and d), respectively.

The sequence ESC c causes a terminal reset, which is what you want if the screen is all garbled. The oft-advised "echo ^V^O" will make only G0 current, but there is no guarantee that G0 points at table a). In some distributions there is a program reset(1) that just does "echo ^[c". If your terminfo entry for the console is correct (and has an entry rs1=\Ec), then "tput reset" will also work.

The user-defined mapping table can be set using **mapscrn(8).** The result of the mapping is that if a symbol c is printed, the symbol s = map[c] is sent to the video memory. The bitmap that corresponds to s is found in the character ROM, and can be changed using **setfont(8).**

Mouse tracking

The mouse tracking facility is intended to return **xterm(1)-compatible** mouse status reports. Because the console driver has no way to know the device or type of the mouse, these reports are returned in the console input stream only when the virtual terminal driver receives a mouse update ioctl. These ioctls must be generated by a mouse-aware user-mode application such as the **gpm(8)** daemon.

The mouse tracking escape sequences generated by **xterm(1)** encode numeric parameters in a single character as *value*+040. For example, '!' is 1. The screen coordinate system is 1-based.

The X10 compatibility mode sends an escape sequence on button press encoding the location and the mouse button pressed. It is enabled by sending ESC [? 9 h and disabled with ESC [? 9 l. On button press, **xterm(1)** sends ESC [M *bxy* (6 characters). Here *b* is button−1, and *x* and *y* are the x and y coordinates of the mouse when the button was pressed. This is the same code the kernel also produces.

Normal tracking mode (not implemented in Linux 2.0.24) sends an escape sequence on both button press and release. Modifier information is also sent. It is enabled by sending ESC [? 1000 h and disabled with ESC [? 1000 l. On button press or release, **xterm(1)** sends ESC [M *bxy*. The low two bits of *b* encode button information: 0=MB1 pressed, 1=MB2 pressed, 2=MB3 pressed, 3=release. The upper bits encode what modifiers were down when the button was pressed and are added together: 4=Shift, 8=Meta, 16=Control. Again *x* and *y* are the x and y coordinates of the mouse event. The upper left corner is (1,1).

Comparisons with other terminals

Many different terminal types are described, like the Linux console, as being "VT100-compatible". Here we discuss differences between the Linux console and the two most important others, the DEC VT102 and **xterm(1).**

Control-character handling

The VT102 also recognized the following control characters:

NUL (0x00) was ignored;

ENQ (0x05) triggered an answerback message;

DC1 (0x11, **^Q**, XON) resumed transmission;

DC3 (0x13, **^S**, XOFF) caused VT100 to ignore (and stop transmitting) all codes except XOFF and XON.

VT100-like DC1/DC3 processing may be enabled by the terminal driver.

The **xterm(1)** program (in VT100 mode) recognizes the control characters BEL, BS, HT, LF, VT, FF, CR, SO, SI, ESC.

Escape sequences

VT100 console sequences not implemented on the Linux console:

ESC N	SS2	Single shift 2. (Select G2 character set for the next character only.)
ESC O	SS3	Single shift 3. (Select G3 character set for the next character only.)
ESC P	DCS	Device control string (ended by ESC \)
ESC X	SOS	Start of string.
ESC ^	PM	Privacy message (ended by ESC \)
ESC \	ST	String terminator
ESC * ...		Designate G2 character set
ESC + ...		Designate G3 character set

The program **xterm(1)** (in VT100 mode) recognizes ESC c, ESC # 8, ESC >, ESC =, ESC D, ESC E, ESC H, ESC M, ESC N, ESC O, ESC P ... ESC \, ESC Z (it answers ESC [? 1 ; 2 c, "I am a VT100 with advanced video option") and ESC ^ ... ESC \ with the same meanings as indicated above. It accepts ESC (, ESC), ESC *, ESC + followed by 0, A, B for the DEC special character and line drawing set, UK, and US-ASCII, respectively.

The user can configure **xterm(1)** to respond to VT220-specific control sequences, and it will identify itself

as a VT52, VT100, and up depending on the way it is configured and initialized.

It accepts ESC] (OSC) for the setting of certain resources. In addition to the ECMA-48 string terminator (ST), **xterm(1)** accepts a BEL to terminate an OSC string. These are a few of the OSC control sequences recognized by **xterm(1)**:

ESC] 0 ; *txt* ST	Set icon name and window title to *txt*.
ESC] 1 ; *txt* ST	Set icon name to *txt*.
ESC] 2 ; *txt* ST	Set window title to *txt*.
ESC] 4 ; *num*; *txt* ST	Set ANSI color *num* to *txt*.
ESC] 10 ; *txt* ST	Set dynamic text color to *txt*.
ESC] 4 6 ; *name* ST	Change log file to *name* (normally disabled by a compile-time option)
ESC] 5 0 ; *fn* ST	Set font to *fn*.

It recognizes the following with slightly modified meaning (saving more state, behaving closer to VT100/VT220):

ESC 7 DECSC	Save cursor
ESC 8 DECRC	Restore cursor

It also recognizes

ESC F		Cursor to lower left corner of screen (if enabled by **xterm**(1)'s **hpLowerleftBugCompat** resource)
ESC l		Memory lock (per HP terminals). Locks memory above the cursor.
ESC m		Memory unlock (per HP terminals).
ESC n	LS2	Invoke the G2 character set.
ESC o	LS3	Invoke the G3 character set.
ESC |	LS3R	Invoke the G3 character set as GR. Has no visible effect in xterm.
ESC }	LS2R	Invoke the G2 character set as GR. Has no visible effect in xterm.
ESC ~	LS1R	Invoke the G1 character set as GR. Has no visible effect in xterm.

It also recognizes ESC % and provides a more complete UTF-8 implementation than Linux console.

CSI Sequences

Old versions of **xterm(1)**, for example, from X11R5, interpret the blink SGR as a bold SGR. Later versions which implemented ANSI colors, for example, XFree86 3.1.2A in 1995, improved this by allowing the blink attribute to be displayed as a color. Modern versions of xterm implement blink SGR as blinking text and still allow colored text as an alternate rendering of SGRs. Stock X11R6 versions did not recognize the color-setting SGRs until the X11R6.8 release, which incorporated XFree86 xterm. All ECMA-48 CSI sequences recognized by Linux are also recognized by *xterm*, however **xterm(1)** implements several ECMA-48 and DEC control sequences not recognized by Linux.

The **xterm(1)** program recognizes all of the DEC Private Mode sequences listed above, but none of the Linux private-mode sequences. For discussion of **xterm(1)**'s own private-mode sequences, refer to the *Xterm Control Sequences* document by Edward Moy, Stephen Gildea, and Thomas E. Dickey available with the X distribution. That document, though terse, is much longer than this manual page. For a chronological overview,

Unknown

details changes to xterm.

The *vttest* program

Unknown

demonstrates many of these control sequences. The **xterm(1)** source distribution also contains sample scripts which exercise other features.

NOTES

ESC 8 (DECRC) is not able to restore the character set changed with ESC %.

BUGS

In 2.0.23, CSI is broken, and NUL is not ignored inside escape sequences.

Some older kernel versions (after 2.0) interpret 8-bit control sequences. These "C1 controls" use codes between 128 and 159 to replace ESC [, ESC] and similar two-byte control sequence initiators. There are fragments of that in modern kernels (either overlooked or broken by changes to support UTF-8), but the implementation is incomplete and should be regarded as unreliable.

Linux "private mode" sequences do not follow the rules in ECMA-48 for private mode control sequences. In particular, those ending with] do not use a standard terminating character. The OSC (set palette) sequence is a greater problem, since **xterm(1)** may interpret this as a control sequence which requires a string terminator (ST). Unlike the **setterm(1)** sequences which will be ignored (since they are invalid control sequences), the palette sequence will make **xterm(1)** appear to hang (though pressing the return-key will fix that). To accommodate applications which have been hardcoded to use Linux control sequences, set the **xterm(1)** resource **brokenLinuxOSC** to true.

An older version of this document implied that Linux recognizes the ECMA-48 control sequence for invisible text. It is ignored.

SEE ALSO

ioctl_console(2), charsets(7)

COLOPHON

This page is part of release 4.16 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man−pages/.