Wikipedia
The Free Encyclopedia

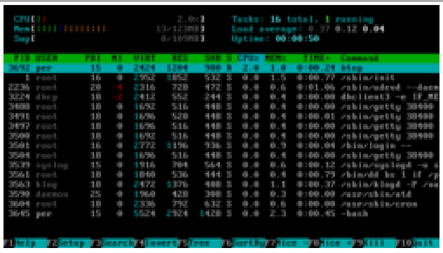
ANSI escape code

ANSI escape sequences are a standard for in-band signaling to control cursor location, color, font styling, and other options on video text terminals and terminal emulators. Certain sequences of bytes, most starting with an ASCII escape character and a bracket character, are embedded into text. The terminal interprets these sequences as commands, rather than text to display verbatim.

ANSI sequences were introduced in the 1970s to replace vendor-specific sequences and became widespread in the computer equipment market by the early 1980s. They are used in development, scientific, commercial text-based applications as well as bulletin board systems to offer standardized functionality.

Although hardware text terminals have become increasingly rare in the 21st century, the relevance of the ANSI standard persists because a great majority of terminal emulators and command consoles interpret at least a portion of the ANSI standard.

ANSI X3.64 (ISO/IEC 6429)


Output of the system-monitor htop, an ncurses-application (which uses SGR and other ANSI/ISO control sequences).

Standard	ECMA-48 ISO/IEC 6429 FIPS 86 ANSI X3.64 JIS X 0211
Classification	ISO/IEC 2022 based control code and control sequence set
Other related encoding(s)	ITU T.61 ISO/IEC 8613-6 / ITU T.416 Other control function standards: ITU T.101 · JIS X 0207 · ISO 6630 · DIN 31626 · ETS 300 706

History

Almost all manufacturers of video terminals added vendor-specific escape sequences to perform operations such as placing the cursor at arbitrary positions on the screen. One example is the VT52 terminal, which allowed the cursor to be placed at an x,y location on the screen by sending the `ESC` character, a `Y` character, and then two characters representing numerical values equal to the x,y location plus 32 (thus starting at the ASCII space character and avoiding the control characters). The Hazeltine 1500 had a similar feature, invoked using `~`, `DC1` and then the X and Y positions separated with a comma. While the two terminals had identical functionality in this regard, different control sequences had to be used to invoke them.

As these sequences were different for different terminals, elaborate libraries such as termcap ("terminal capabilities") and utilities such as tput had to be created so programs could use the same API to work with any terminal. In addition, many of these terminals required sending numbers (such as row and column) as the binary values of the characters; for some programming languages, and for systems that did not use ASCII internally, it was often difficult to turn a number into the correct character.

The ANSI standard attempted to address these problems by making a command set that all terminals would use and requiring all numeric information to be transmitted as ASCII numbers. The first standard in the series was ECMA-48, adopted in 1976.[1] It was a continuation of a series of character coding standards, the first one being ECMA-6 from 1965, a 7-bit standard from which ISO 646 originates. The name "ANSI escape sequence" dates from 1979 when ANSI adopted ANSI X3.64. The ANSI X3L2 committee collaborated with the ECMA committee TC 1 to produce nearly identical standards. These two standards were merged into an international standard, ISO 6429.[1] In 1994, ANSI withdrew its standard in favor of the international standard.

The first popular video terminal to support these sequences was the Digital VT100, introduced in 1978.[2] This model was very successful in the market, which sparked a variety of VT100 clones, among the earliest and most popular of which was the much more affordable Zenith Z-19 in 1979.[3] Others included the Qume QVT-108, Televideo TVI-970, Wyse WY-99GT as well as optional "VT100" or "VT103" or "ANSI" modes with varying degrees of compatibility on many other brands. The popularity of these gradually led to more and more software (especially bulletin board systems and other online services) assuming the escape sequences worked, leading to almost all new terminals and emulator programs supporting them.

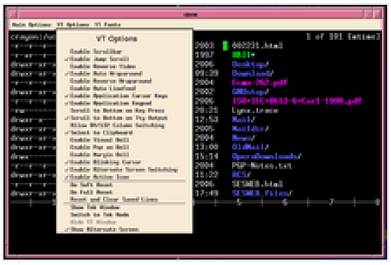
The DEC VT100 video display terminal.

In 1981, ANSI X3.64 was adopted for use in the US government by FIPS publication 86. Later, the US government stopped duplicating industry standards, so FIPS pub. 86 was withdrawn.[4]

ECMA-48 has been updated several times and is currently at its 5th edition, from 1991. It is also adopted by ISO and IEC as standard **ISO/IEC 6429**.[5] A version is adopted as a Japanese Industrial Standard, as JIS X 0211.

Related standards include ITU T.61, the Teletex standard, and the **ISO/IEC 8613**, the Open Document Architecture standard (mainly ISO/IEC 8613-6 or ITU T.416). The two systems share many escape codes with the ANSI system, with extensions that are not necessarily meaningful to computer terminals. Both systems quickly fell into disuse, but ECMA-48 does mark the extensions used in them as reserved.

Platform support


The Xterm terminal emulator.

In the early 1980s, large amounts of software directly used these sequences to update screen displays. This included everything on VMS (which assumed Dec terminals), most software designed to be portable on CP/M home computers, and even lots of Unix software as it was easier to use than the termcap libraries, such as the shell script examples below in this article.

Terminal emulators for communicating with remote machines almost always implement ANSI escape codes. This includes anything written to communicate with bulletin-board systems on home and personal computers. On Unix terminal emulators such as xterm also can communicate with software running on the same machine, and thus software running in X11 under a terminal emulator could assume the ability to write these sequences.

As computers got more powerful even built-in displays started supporting them, allowing software to be portable between CP/M systems. There were attempts to extend the escape sequences to support printers[6] and as an early PDF-like document storage format, the Open Document Architecture.

DOS and Windows

The IBM PC, introduced in 1983, did not support these or any other escape sequences for updating the screen. Only a few control characters (BEL, CR, LF, BS) were interpreted by the underlying BIOS. Any display effects had to be done with BIOS calls, which were notoriously slow, or by directly manipulating the IBM PC hardware. This made any interesting software non-portable and led to the need to duplicate details of the display hardware in PC Clones.

DOS version 2.0 included optional support with a device driver named `ANSI.SYS`. Poor performance, and the fact that it was not installed by default, meant software rarely took advantage of it. Some other systems did try to address the need for these sequences, many clones of DOS handled them without a driver, and OS/2 had an `ANSI` command that enabled the sequences.

The Windows Console did not support ANSI escape sequences, nor did Microsoft provide any method to enable them. Some replacements or additions for the console window such as JP Software's TCC (formerly 4NT), Michael J. Mefford's ANSI.COM, Jason Hood's `ANSICON`[7] and Maximus5's ConEmu interpreted ANSI escape sequences printed by programs. A Python package named colorama[8] internally interprets ANSI escape sequences in text being printed, translating them to win32 calls to modify the state of the terminal, to make it easier to port Python code using ANSI to Windows. Cygwin performs similar translation to all output written to the console using Cygwin file descriptors, the filtering is done by the output functions of `cygwin1.dll`, to allow porting of POSIX C code to Windows.

In 2016, Microsoft released the Windows 10 version 1511 update which unexpectedly implemented support for ANSI escape sequences, over two decades after the debut of Windows NT.[9] This was done alongside Windows Subsystem for Linux, apparently to allow Unix-like terminal-based software to use the Windows Console. Windows PowerShell 5.1 enabled this by default, and PowerShell 6 made it possible to embed the necessary ESC character into a string with `` `e ``.[10]

Windows Terminal, introduced in 2019, supports the sequences by default, and Microsoft intends to replace the Windows Console with Windows Terminal.[11]

Description

C0 control codes

Almost all users assume some functions of some single-byte characters. Initially defined as part of ASCII, the default C0 control code set is now defined in ISO 6429 (ECMA-48), making it part of the same standard as the C1 set invoked by the ANSI escape sequences (although ISO 2022 allows the ISO 6429 C0 set to be used without the ISO 6429 C1 set, and *vice versa*, provided that 0x1B is always ESC). This is used to shorten the amount of data transmitted, or to perform some functions that are unavailable from escape sequences:

Popular C0 control codes (not an exhaustive list)

^	C0	Abbr	Name	Effect
`^G`	0x07	`BEL`	Bell	Makes an audible noise.
`^H`	0x08	`BS`	Backspace	Moves the cursor left (but may "backwards wrap" if cursor is at start of line).
`^I`	0x09	`HT`	Tab	Moves the cursor right to next multiple of 8.
`^J`	0x0A	`LF`	Line Feed	Moves to next line, scrolls the display up if at bottom of the screen. Usually does not move horizontally, though programs should not rely on this.
`^L`	0x0C	`FF`	Form Feed	Move a printer to top of next page. Usually does not move horizontally, though programs should not rely on this. Effect on video terminals varies.
`^M`	0x0D	`CR`	Carriage Return	Moves the cursor to column zero.
`^[`	0x1B	`ESC`	Escape	Starts all the escape sequences

Escape sequences vary in length. The general format for an ANSI-compliant escape sequence is defined by ANSI X3.41 (equivalent to ECMA-35 or ISO/IEC 2022).[12]: 13.1 The escape sequences consist only of bytes in the range `0x20—0x7F` (all the non-control ASCII characters), and can be parsed without looking ahead. The behavior when a control character, a byte with the high bit set, or a byte that is not part of any valid sequence, is encountered before the end is undefined.

Fe Escape sequences

If the `ESC` is followed by a byte in the range 0x40 to 0x5F, the escape sequence is of type `Fe`. Its interpretation is delegated to the applicable C1 control code standard.[12]: 13.2.1 Accordingly, all escape sequences corresponding to C1 control codes from ANSI X3.64 / ECMA-48 follow this format.[5]: 5.3.a

The standard says that, in 8-bit environments, the control functions corresponding to type `Fe` escape sequences (those from the set of C1 control codes) can be represented as single bytes in the 0x80–0x9F range.[5]: 5.3.b This is possible in character encodings conforming to the provisions for an 8-bit code made in ISO 2022, such as the ISO 8859 series. However, in character encodings used on modern devices such as UTF-8 or CP-1252, those codes are often used for other purposes, so only the 2-byte sequence is typically used. In the case of UTF-8, representing a C1 control code via the C1 Controls and Latin-1 Supplement block results in a different two-byte code (e.g. `0xC2,0x8E` for `U+008E`), but no space is saved this way.

Some type Fe (C1 set element) ANSI escape sequences (not an exhaustive list)

Code	C1	Abbr	Name	Effect
`ESC N`	`0x8E`	`SS2`	Single Shift Two	Select a single character from one of the alternative character sets. SS2 selects the G2 character set, and SS3 selects the G3 character set.[13] In a 7-bit environment, this is followed by one or more GL bytes (0x20–0x7F) specifying a character from that set.[12]:9.4 In an 8-bit environment, these may instead be GR bytes (0xA0–0xFF).[12]:8.4
`ESC O`	`0x8F`	`SS3`	Single Shift Three	
`ESC P`	`0x90`	`DCS`	Device Control String	Terminated by ST.[5]:5.6 Xterm's uses of this sequence include defining User-Defined Keys, and requesting or setting Termcap/Terminfo data.[13]
`ESC [`	`0x9B`	`CSI`	Control Sequence Introducer	Starts most of the useful sequences, terminated by a byte in the range 0x40 through 0x7E.[5]:5.4
`ESC \`	`0x9C`	`ST`	String Terminator	Terminates strings in other controls.[5]:8.3.143
`ESC ]`	`0x9D`	`OSC`	Operating System Command	Starts a control string for the operating system to use, terminated by ST.[5]:8.3.89
`ESC X`	`0x98`	`SOS`	Start of String	Takes an argument of a string of text, terminated by ST.[5]:5.6 The uses for these string control sequences are defined by the application[5]:8.3.2,8.3.128 or privacy discipline.[5]:8.3.94 These functions are rarely implemented and the arguments are ignored by xterm.[13] Some Kermit clients allow the server to automatically execute Kermit commands on the client by embedding them in APC sequences; this is a potential security risk if the server is untrusted.[14]
`ESC ^`	`0x9E`	`PM`	Privacy Message	
`ESC _`	`0x9F`	`APC`	Application Program Command	

CSI (Control Sequence Introducer) sequences

For Control Sequence Introducer, or CSI, commands, the `ESC [` (written as `\e[` or `\033[` in several programming languages) is followed by any number (including none) of "parameter bytes" in the range 0x30–0x3F (ASCII `0–9:;<=>?`), then by any number of "intermediate bytes" in the range 0x20–0x2F (ASCII space and `!"#$%&'()*+,-./`), then finally by a single "final byte" in the range 0x40–0x7E (ASCII ``@A–Z[\]^_`a–z{|}~``).[5]:5.4

All common sequences just use the parameters as a series of semicolon-separated numbers such as `1;2;3`. Missing numbers are treated as `0` (`1;;3` acts like the middle number is `0`, and no parameters at all in `ESC[m` acts like a `0` reset code). Some sequences (such as CUU) treat `0` as `1` in order to make missing parameters useful.[5]:F.4.2

A subset of arrangements was declared "private" so that terminal manufacturers could insert their own sequences without conflicting with the standard. Sequences containing the parameter bytes `<=>?` or the final bytes 0x70–0x7E (`p–z{|}~`) are private.

The behavior of the terminal is undefined in the case where a CSI sequence contains any character outside of the range 0x20–0x7E. These illegal characters are either C0 control characters (the range 0–0x1F), DEL (0x7F), or bytes with the high bit set. Possible responses are to ignore the byte, to process it immediately, and furthermore whether to continue with the CSI sequence, to abort it immediately, or to ignore the rest of it.

Some ANSI control sequences (not an exhaustive list)

Code	Abbr	Name	Effect
`CSI` *n* `A`	`CUU`	Cursor Up	Moves the cursor *n* (default `1`) cells in the given direction. If the cursor is already at the edge of the screen, this has no effect.
`CSI` *n* `B`	`CUD`	Cursor Down	
`CSI` *n* `C`	`CUF`	Cursor Forward	
`CSI` *n* `D`	`CUB`	Cursor Back	
`CSI` *n* `E`	`CNL`	Cursor Next Line	Moves cursor to beginning of the line *n* (default `1`) lines down. (not ANSI.SYS)
`CSI` *n* `F`	`CPL`	Cursor Previous Line	Moves cursor to beginning of the line *n* (default `1`) lines up. (not ANSI.SYS)
`CSI` *n* `G`	`CHA`	Cursor Horizontal Absolute	Moves the cursor to column *n* (default `1`). (not ANSI.SYS)
`CSI` *n* `;` *m* `H`	`CUP`	Cursor Position	Moves the cursor to row *n*, column *m*. The values are 1-based, and default to `1` (top left corner) if omitted. A sequence such as `CSI ;5H` is a synonym for `CSI 1;5H` as well as `CSI 17;H` is the same as `CSI 17H` and `CSI 17;1H`
`CSI` *n* `J`	`ED`	Erase in Display	Clears part of the screen. If *n* is `0` (or missing), clear from cursor to end of screen. If *n* is `1`, clear from cursor to beginning of the screen. If *n* is `2`, clear entire screen (and moves cursor to upper left on DOS ANSI.SYS). If *n* is `3`, clear entire screen and delete all lines saved in the scrollback buffer (this feature was added for xterm and is supported by other terminal applications).
`CSI` *n* `K`	`EL`	Erase in Line	Erases part of the line. If *n* is `0` (or missing), clear from cursor to the end of the line. If *n* is `1`, clear from cursor to beginning of the line. If *n* is `2`, clear entire line. Cursor position does not change.
`CSI` *n* `S`	`SU`	Scroll Up	Scroll whole page up by *n* (default `1`) lines. New lines are added at the bottom. (not ANSI.SYS)
`CSI` *n* `T`	`SD`	Scroll Down	Scroll whole page down by *n* (default `1`) lines. New lines are added at the top. (not ANSI.SYS)
`CSI` *n* `;` *m* `f`	`HVP`	Horizontal Vertical Position	Same as CUP, but counts as a format effector function (like CR or LF) rather than an editor function (like CUD or CNL). This can lead to different handling in certain terminal modes.[5]:Annex A
`CSI` *n* `m`	`SGR`	Select Graphic Rendition	Sets colors and style of the characters following this code
`CSI 5i`		AUX Port On	Enable aux serial port usually for local serial printer
`CSI 4i`		AUX Port Off	Disable aux serial port usually for local serial printer
`CSI 6n`	`DSR`	Device Status Report	Reports the cursor position (CPR) by transmitting `ESC[n;mR`, where *n* is the row and *m* is the column.

Some popular private sequences

Code	Abbr	Name	Effect
`CSI s`	`SCP, SCOSC`	Save Current Cursor Position	Saves the cursor position/state in SCO console mode.[15] In vertical split screen mode, instead used to set (as `CSI n ; n s`) or reset left and right margins.[16]
`CSI u`	`RCP, SCORC`	Restore Saved Cursor Position	Restores the cursor position/state in SCO console mode.[17]
`CSI ? 25 h`	`DECTCEM`		Shows the cursor, from the VT220.
`CSI ? 25 l`	`DECTCEM`		Hides the cursor.
`CSI ? 1004 h`			Enable reporting focus. Reports whenever terminal emulator enters or exits focus as `ESC [I` and `ESC [O`, respectively.
`CSI ? 1004 l`			Disable reporting focus.
`CSI ? 1049 h`			Enable alternative screen buffer, from xterm
`CSI ? 1049 l`			Disable alternative screen buffer, from xterm
`CSI ? 2004 h`			Turn on bracketed paste mode.[18] In bracketed paste mode, text pasted into the terminal will be surrounded by `ESC [200~` and `ESC [201~`; programs running in the terminal should not treat characters bracketed by those sequences as commands (Vim, for example, does not treat them as commands).[19] From xterm[20]
`CSI ? 2004 l`			Turn off bracketed paste mode.

SGR (Select Graphic Rendition) parameters

The control sequence `CSI n m`, named Select Graphic Rendition (SGR), sets display attributes. Several attributes can be set in the same sequence, separated by semicolons.[21] Each display attribute remains in effect until a following occurrence of SGR resets it.[5] If no codes are given, `CSI m` is treated as `CSI 0 m` (reset / normal).

n	Name	Note
0	Reset *or* normal	All attributes become turned off
1	Bold or increased intensity	As with faint, the color change is a PC (SCO / CGA) invention.[22]
2	Faint, decreased intensity, *or* dim	May be implemented as a light font weight like bold.[23]
3	Italic	Not widely supported. Sometimes treated as inverse or blink.[22]
4	Underline	Style extensions exist for Kitty, VTE, mintty, iTerm2 and Konsole.[24][25][26]
5	Slow blink	Sets blinking to less than 150 times per minute
6	Rapid blink	MS-DOS ANSI.SYS, 150+ per minute; not widely supported
7	Reverse video *or* invert	Swap foreground and background colors; inconsistent emulation[27]
8	Conceal *or* hide	Not widely supported.
9	Crossed-out, *or* strike	Characters legible but marked as if for deletion. Not supported in Terminal.app.
10	Primary (default) font	
11–19	Alternative font	Select alternative font $n - 10$
20	Fraktur (Gothic)	Rarely supported
21	Doubly underlined; or: not bold	Double-underline per ECMA-48,[5]:8.3.117 but instead disables bold intensity on several terminals, including in the Linux kernel's console before version 4.17.[28]
22	Normal intensity	Neither bold nor faint; color changes where intensity is implemented as such.
23	Neither italic, nor blackletter	
24	Not underlined	Neither singly nor doubly underlined
25	Not blinking	Turn blinking off
26	Proportional spacing	ITU T.61 and T.416, not known to be used on terminals
27	Not reversed	
28	Reveal	Not concealed
29	Not crossed out	
30–37	Set foreground color	
38	Set foreground color	Next arguments are `5;n` or `2;r;g;b`
39	Default foreground color	Implementation defined (according to standard)
40–47	Set background color	
48	Set background color	Next arguments are `5;n` or `2;r;g;b`
49	Default background color	Implementation defined (according to standard)
50	Disable proportional spacing	T.61 and T.416
51	Framed	Implemented as "emoji variation selector" in mintty.[29]
52	Encircled	
53	Overlined	Not supported in Terminal.app
54	Neither framed nor encircled	
55	Not overlined	
58	Set underline color	Not in standard; implemented in Kitty, VTE, mintty, and iTerm2.[24][25] Next arguments are `5;n` or `2;r;g;b`.
59	Default underline color	Not in standard; implemented in Kitty, VTE, mintty, and iTerm2.[24][25]
60	Ideogram underline or right side line	Rarely supported

`61`	Ideogram double underline, *or* double line on the right side	
`62`	Ideogram overline or left side line	
`63`	Ideogram double overline, *or* double line on the left side	
`64`	Ideogram stress marking	
`65`	No ideogram attributes	Reset the effects of all of `60`–`64`
`73`	Superscript	Implemented only in mintty[29]
`74`	Subscript	
`75`	Neither superscript nor subscript	
`90–97`	Set bright foreground color	Not in standard; originally implemented by aixterm[13]
`100–107`	Set bright background color	

Colors

3-bit and 4-bit

The original specification only had 8 colors, and just gave them names. The SGR parameters 30–37 selected the foreground color, while 40–47 selected the background. Quite a few terminals implemented "bold" (SGR code 1) as a brighter color rather than a different font, thus providing 8 additional foreground colors. Usually you could not get these as background colors, though sometimes inverse video (SGR code 7) would allow that. Examples: to get black letters on white background use `ESC[30;47m`, to get red use `ESC[31m`, to get bright red use `ESC[1;31m`. To reset colors to their defaults, use `ESC[39;49m` (not supported on some terminals), or reset all attributes with `ESC[0m`. Later terminals added the ability to directly specify the "bright" colors with 90–97 and 100–107.

The chart below shows a few examples of how VGA standard and modern terminal emulators translate the 4-bit color codes into 24-bit color codes.

FG	BG	Name	VGA[a]	Windows XP Console[b]	Windows PowerShell& 1.0–6.0[c]	Visual Studio Code[d]	Windows 10 Console[e]	Terminal.app
30	40	Black	0, 0, 0				12, 12, 12	0, 0, 0
31	41	Red	170, 0, 0	128, 0, 0		205, 49, 49	197, 15, 31	153, 0, 0
32	42	Green	0, 170, 0	0, 128, 0		13, 188, 121	19, 161, 14	0, 166, 0
33	43	Yellow	170, 85, 0[g]	128, 128, 0	238, 237, 240	229, 229, 16	193, 156, 0	153, 153, 0
34	44	Blue	0, 0, 170	0, 0, 128		36, 114, 200	0, 55, 218	0, 0, 178
35	45	Magenta	170, 0, 170	128, 0, 128	1, 36, 86	188, 63, 188	136, 23, 152	178, 0, 178
36	46	Cyan	0, 170, 170	0, 128, 128		17, 168, 205	58, 150, 221	0, 166, 178
37	47	White	170, 170, 170	192, 192, 192		229, 229, 229	204, 204, 204	191, 191, 191
90	100	Bright Black (Gray)	85, 85, 85	128, 128, 128		102, 102, 102	118, 118, 118	102, 102, 102
91	101	Bright Red	255, 85, 85	255, 0, 0		241, 76, 76	231, 72, 86	230, 0, 0
92	102	Bright Green	85, 255, 85	0, 255, 0		35, 209, 139	22, 198, 12	0, 217, 0
93	103	Bright Yellow	255, 255, 85	255, 255, 0		245, 245, 67	249, 241, 165	230, 230, 0
94	104	Bright Blue	85, 85, 255	0, 0, 255		59, 142, 234	59, 120, 255	0, 0, 255
95	105	Bright Magenta	255, 85, 255	255, 0, 255		214, 112, 214	180, 0, 158	230, 0, 230
96	106	Bright Cyan	85, 255, 255	0, 255, 255		41, 184, 219	97, 214, 214	0, 230, 230
97	107	Bright White	255, 255, 255			229, 229, 229	242, 242, 242	230, 230, 230

8-bit

As 256-color lookup tables became common on graphic cards, escape sequences were added to select from a pre-defined set of 256 colors:

```
ESC[38;5;⟨n⟩m Select foreground color      where n is a number from the table below
ESC[48;5;⟨n⟩m Select background color
  0-  7:  standard colors (as in ESC [ 30–37 m)
  8- 15:  high intensity colors (as in ESC [ 90–97 m)
 16-231:  6 × 6 × 6 cube (216 colors): 16 + 36 × r + 6 × g + b (0 ≤ r, g, b ≤ 5)
232-255:  grayscale from dark to light in 24 steps
```

The ITU's T.416 Information technology - Open Document Architecture (ODA) and interchange format: Character content architectures[33] uses ":" as separator characters instead:

```
ESC[38:5:⟨n⟩m Select foreground color      where n is a number from the table below
ESC[48:5:⟨n⟩m Select background color
```

256-color mode — foreground: ESC[38;5;#m   background: ESC[48;5;#m

Standard colors

0	1	2	3	4	5	6	7

High-intensity colors

8	9	10	11	12	13	14	15

216 colors

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231

Grayscale colors

232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

There has also been a similar but incompatible 88-color encoding using the same escape sequence, seen in `rxvt` and `xterm-88color`. Not much is known about the scheme besides the color codes. It uses a 4×4×4 color cube.

24-bit

As "true color" graphic cards with 16 to 24 bits of color became common, applications began to support 24-bit colors. Terminal emulators supporting setting 24-bit foreground and background colors with escape sequences include Xterm,[13] KDE's Konsole,[34][35] and iTerm, as well as all libvte based terminals,[36] including GNOME Terminal.[37]

```
ESC[38;2;⟨r⟩;⟨g⟩;⟨b⟩ m Select RGB foreground color
ESC[48;2;⟨r⟩;⟨g⟩;⟨b⟩ m Select RGB background color
```

The syntax is likely based on the ITU's T.416 Open Document Architecture (ODA) and interchange format: Character content architectures,[33] which was adopted as ISO/IEC 8613-6 but ended up as a commercial failure. The ODA version is more elaborate and thus incompatible:

- The parameters after the '2' (r, g, and b) are optional and can be left empty.
- Semicolons are replaced by colons, as above.
- There is a leading "colorspace ID".[13] The definition of the colorspace ID is not included in that document so it may be blank to represent the unspecified default.
- In addition to the '2' value after 48 to specify a Red-Green-Blue format (and the '5' above for a 0-255 indexed color), there are alternatives of '0' for implementation-defined and '1' for transparent - neither of which have any further parameters; '3' specifies colors using a Cyan-Magenta-Yellow scheme, and '4' for a Cyan-Magenta-Yellow-Black one, the latter using the position marked as "unused" for the Black component:[33]

```
ESC[38:2:⟨Color-Space-ID⟩:⟨r⟩:⟨g⟩:⟨b⟩:⟨unused⟩:⟨CS tolerance⟩:
⟨Color-Space associated with tolerance: 0 for "CIELUV"; 1 for "CIELAB"⟩ m Select RGB foreground color
ESC[48:2:⟨Color-Space-ID⟩:⟨r⟩:⟨g⟩:⟨b⟩:⟨unused⟩:⟨CS tolerance⟩:
⟨Color-Space associated with tolerance: 0 for "CIELUV"; 1 for "CIELAB"⟩ m Select RGB background color
```

The ITU-RGB variation is supported by xterm, with the colorspace ID and tolerance parameters ignored. The simpler scheme using semicolons is initially found in Konsole.[13]:Can I set a color by its number?

Unix environment variables relating to color support

Rather than using the color support in termcap and terminfo introduced in SVr3.2 (1987),[38] the S-Lang library (version 0.99-32 (https://www.nic.funet.fi/index/languages/slang/v0.99/0.99-31__0.99-32.diff.gz), June 1996) used a separate environment variable `$COLORTERM` to indicate whether a terminal emulator could use colors at all, and later added values to indicate if it supported 24-bit color.[39][40] This system, although poorly documented, became widespread enough for Fedora and RHEL to consider using it as a simpler and more universal detection mechanism compared to querying the now-updated libraries.[41]

Some terminal emulators (urxvt, konsole) set `$COLORFGBG` to report the color scheme of the terminal (mainly light vs. dark background). This behavior originated in S-Lang[40] and is used by vim. Gnome-terminal refuses to add this behavior, as the syntax for the value is not agreed upon, the value cannot be changed upon a runtime change of the palette, and more "proper" xterm OSC 4/10/11 sequences already exist.[42]

OSC (Operating System Command) sequences

Most Operating System Command sequences were defined by Xterm, but many are also supported by other terminal emulators. For historical reasons, Xterm can end the command with `BEL` (0x07) as well as the standard `ST` (0x9C or 0x1B 0x5C).[13] For example, Xterm allows the window title to be set by `ESC ]0;this is the window title BEL`.

A non-xterm extension is the hyperlink, `ESC ]8;;link ST` from 2017, used by VTE,[43] iTerm2,[43] and mintty,[44] among others.[45]

The Linux console uses `ESC ] P n rr gg bb` to change the palette, which, if hard-coded into an application, may hang other terminals.[46] However, appending `ST` will be ignored by Linux and form a proper, ignorable sequence for other terminals.

Fs Escape sequences

If the `ESC` is followed by a byte in the range `0x60—0x7E`, the escape sequence is of type `Fs`. This type is used for control functions individually registered with the ISO-IR registry.[47] A table of these is listed under ISO/IEC 2022.

Fp Escape sequences

If the `ESC` is followed by a byte in the range `0x30—0x3F`, the escape sequence is of type `Fp`, which is set apart for up to sixteen private-use control functions.[12]: 6.5.3

Some type `Fp` (private-use) escape sequences recognised by the VT100

	Abbr	Name	Effect
`ESC 7`	`DECSC`	DEC Save Cursor	Saves the cursor position, encoding shift state and formatting attributes.[48][13]
`ESC 8`	`DECRC`	DEC Restore Cursor	Restores the cursor position, encoding shift state and formatting attributes from the previous DECSC if any, otherwise resets these all to their defaults.[48][13]

nF Escape sequences

If the `ESC` is followed by a byte in the range `0x20—0x2F`, the escape sequence is of type `nF`. Said byte is followed by any number of additional bytes in this range, and then a byte in the range `0x30-0x7E`. These escape sequences are further subcategorised by the low two bits of the first byte, e.g. "type `2F`" for sequences where the first byte is `0x22`; and by whether the final byte is in the range `0x30—0x3F` indicating private use (e.g. "type `2Fp`") or not (e.g. "type `2Ft`").[12]: 13.2.1

Most of the `nFt` sequences are for changing the current character set, and are listed in ISO/IEC 2022. Some others:

Some type `0Ft` (announcement) ANSI escape sequences[13][12]: 15.2

	Abbr	Name	Effect
`ESC SP F`	ACS6 S7C1T	Announce Code Structure 6 Send 7-bit C1 Control Character to the Host	Makes the function keys send ESC + letter instead of 8-bit C1 codes.
`ESC SP G`	ACS7 S8C1T	Announce Code Structure 7 Send 8-bit C1 Control Character to the Host	Makes the function keys send 8-bit C1 codes.

If the first byte is '#' the public sequences are reserved for additional ISO-IR registered individual control functions.[12]: 6.5.2 No such sequences are presently registered.[47] Type `3Fp` sequences (which includes ones starting with '#') are available for private-use control functions.[12]: 6.5.3

Some type 3Fp (private-use) escape sequences recognised by the VT100

	Abbr	Name	Effect
`ESC # 3`	`DECDHL`	DEC Double-Height Letters, Top Half	Makes the current line use characters twice as tall. This code is for the top half.[49]
`ESC # 4`	`DECDHL`	DEC Double-Height Letters, Bottom Half	Makes the current line use characters twice as tall. This code is for the bottom half.[49]
`ESC # 5`	`DECSWL`	DEC Single-Width Line	Makes the current line use single-width characters, per the default behaviour.[50][13]
`ESC # 6`	`DECDWL`	DEC Double-Width Line	Makes the current line use double-width characters, discarding any characters in the second half of the line.[51][13]

Examples

`CSI 2 J` — This clears the screen and, on some devices, locates the cursor to the y,x position 1,1 (upper left corner).

`CSI 32 m` — This makes text green. The green may be a dark, dull green, so you may wish to enable Bold with the sequence `CSI 1 m` which would make it bright green, or combined as `CSI 32 ; 1 m`. Some implementations use the Bold state to make the character Bright.

`CSI 0 ; 6 8 ; "DIR" ; 13 p` — This reassigns the key F10 to send to the keyboard buffer the string "DIR" and ENTER, which in the DOS command line would display the contents of the current directory. (MS-DOS ANSI.SYS only) This was sometimes used for ANSI bombs. This is a private-use code (as indicated by the letter p), using a non-standard extension to include a string-valued parameter. Following the letter of the standard would consider the sequence to end at the letter D.

`CSI s` — This saves the cursor position. Using the sequence `CSI u` will restore it to the position. Say the current cursor position is 7(y) and 10(x). The sequence `CSI s` will save those two numbers. Now you can move to a different cursor position, such as 20(y) and 3(x), using the sequence `CSI 20 ; 3 H` or `CSI 20 ; 3 f`. Now if you use the sequence CSI u the cursor position will return to 7(y) and 10(x). Some terminals require the DEC sequences `ESC 7` / `ESC 8` instead which is more widely supported.

In shell scripting

ANSI escape codes are often used in UNIX and UNIX-like terminals to provide syntax highlighting. For example, on compatible terminals, the following *list* command color-codes file and directory names by type.

```
ls --color
```

Users can employ escape codes in their scripts by including them as part of *standard output* or *standard error*. For example, the following GNU *sed* command embellishes the output of the *make* command by displaying lines containing words starting with "WARN" in reverse video and words starting with "ERR" in bright yellow on a dark red background (letter case is ignored). The representations of the codes are highlighted.[52]

```
make 2>&1 | sed -e 's/.*\bWARN.*/\x1b[7m&\x1b[0m/i' -e 's/.*\bERR.*/\x1b[93;41m&\x1b[0m/i'
```

The following Bash function flashes the terminal (by alternately sending reverse and normal video mode codes) until the user presses a key.[53]

```
flasher () { while true; do printf \\e[?5h; sleep 0.1; printf \\e[?5l; read -s -n1 -t1 && break; done; }
```

This can be used to alert a programmer when a lengthy command terminates, such as with `make ; flasher` .[54]

```
printf \\033c
```

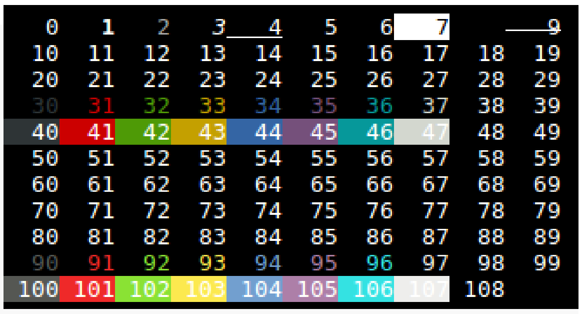
This will reset the console, similar to the command `reset` on modern Linux systems; however it should work even on older Linux systems and on other (non-Linux) UNIX variants.

In C

```c
#include <stdio.h>

int main(void)
{
    int i, j, n;

    for (i = 0; i < 11; i++) {
        for (j = 0; j < 10; j++) {
            n = 10 * i + j;
            if (n > 108) break;
            printf("\033[%dm %3d\033[m", n, n);
        }
        printf("\n");
    }
    return 0;
}
```



```
  0   1   2   3   4   5   6   7       9
 10  11  12  13  14  15  16  17  18  19
 20  21  22  23  24  25  26  27  28  29
 30  31  32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47  48  49
 50  51  52  53  54  55  56  57  58  59
 60  61  62  63  64  65  66  67  68  69
 70  71  72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99
100 101 102 103 104 105 106 107 108
```

Terminal input sequences

Pressing special keys on the keyboard, as well as outputting many xterm CSI, DCS, or OSC sequences, often produces a CSI, DCS, or OSC sequence, sent from the terminal to the computer as though the user typed it.

When typing input on a terminal keypresses outside the normal main alphanumeric keyboard area can be sent to the host as ANSI sequences. For keys that have an equivalent output function, such as the cursor keys, these often mirror the output sequences. However, for most keypresses there isn't an equivalent output sequence to use.

There are several encoding schemes, and unfortunately most terminals mix sequences from different schemes, so host software has to be able to deal with input sequences using any scheme. To complicate the matter, the VT terminals themselves have two schemes of input, *normal mode* and *application mode* that can be switched by the application.

(draft section)

```
<char>                                          -> char
<esc> <nochar>                                  -> esc
<esc> <esc>                                     -> esc
<esc> <char>                                    -> Alt-keypress or keycode sequence
<esc> '[' <nochar>                              -> Alt-[
<esc> '[' (<modifier>) <char>                   -> keycode sequence, <modifier> is a decimal number and
defaults to 1 (xterm)
<esc> '[' (<keycode>) (';'<modifier>) '~'       -> keycode sequence, <keycode> and <modifier> are decimal
numbers and default to 1 (vt)
```

If the terminating character is '~', the first number must be present and is a keycode number, the second number is an optional modifier value. If the terminating character is a letter, the letter is the keycode value, and the optional number is the modifier value.

The modifier value defaults to 1, and after subtracting 1 is a bitmap of modifier keys being pressed: Meta+Ctrl+Alt+⇧ Shift. So, for example, `<esc>[4;2~` is ⇧ Shift+End, `<esc>[20~` is function key F9, `<esc>[5C` is Ctrl+→.

In other words, the modifier is the sum of the following numbers:

Key pressed	Number	Comment
	1	always added, the rest are optional
Shift	1	
(Left) Alt	2	
Control	4	
Meta	8	

```
vt sequences:
<esc>[1~    - Home        <esc>[16~   -             <esc>[31~   - F17
<esc>[2~    - Insert      <esc>[17~   - F6          <esc>[32~   - F18
<esc>[3~    - Delete      <esc>[18~   - F7          <esc>[33~   - F19
<esc>[4~    - End         <esc>[19~   - F8          <esc>[34~   - F20
<esc>[5~    - PgUp        <esc>[20~   - F9          <esc>[35~   -
<esc>[6~    - PgDn        <esc>[21~   - F10
<esc>[7~    - Home        <esc>[22~   -
<esc>[8~    - End         <esc>[23~   - F11
<esc>[9~    -             <esc>[24~   - F12
<esc>[10~   - F0          <esc>[25~   - F13
<esc>[11~   - F1          <esc>[26~   - F14
<esc>[12~   - F2          <esc>[27~   -
<esc>[13~   - F3          <esc>[28~   - F15
<esc>[14~   - F4          <esc>[29~   - F16
<esc>[15~   - F5          <esc>[30~   -

xterm sequences:
<esc>[A     - Up          <esc>[K     -             <esc>[U     -
<esc>[B     - Down        <esc>[L     -             <esc>[V     -
<esc>[C     - Right       <esc>[M     -             <esc>[W     -
<esc>[D     - Left        <esc>[N     -             <esc>[X     -
<esc>[E     -             <esc>[O     -             <esc>[Y     -
<esc>[F     - End         <esc>[1P    - F1          <esc>[Z     -
<esc>[G     - Keypad 5    <esc>[1Q    - F2
<esc>[H     - Home        <esc>[1R    - F3
<esc>[I     -             <esc>[1S    - F4
<esc>[J     -             <esc>[T     -
```

`<esc>[A` to `<esc>[D` are the same as the ANSI output sequences. The `<modifier>` is normally omitted if no modifier keys are pressed, but most implementations always emit the `<modifier>` for F1–F4. (draft section)

Xterm has a comprehensive documentation page on the various function-key and mouse input sequence schemes from DEC's VT terminals and various other terminals it emulates.[13] Thomas Dickey has added a lot of support to it over time;[55] he also maintains a list of default keys used by other terminal emulators for comparison.[56]

- On the Linux console, certain function keys generate sequences of the form `CSI [` *`char`*. The CSI sequence should terminate on the `[`.
- Old versions of Terminator generate `SS3 1;` *`modifiers char`* when F1 – F4 are pressed with modifiers. The faulty behavior was copied from GNOME Terminal.
- xterm replies `CSI` *`row`* `;` *`column`* `R` if asked for cursor position and `CSI 1 ;` *`modifiers`* `R` if the F3 key is pressed with modifiers, which collide in the case of *`row`* `== 1`. This can be avoided by using the *?* private modifier as `CSI ? 6 n`, which will be reflected in the response as `CSI ?` *`row`* `;` *`column`* `R`.
- many terminals prepend `ESC` to any character that is typed with the alt key down. This creates ambiguity for uppercase letters and symbols `@[\]^_`, which would form C1 codes.
- Konsole generates `SS3` *`modifiers char`* when F1 – F4 are pressed with modifiers.
- iTerm2 supports reporting additional keys via an enhanced CSI u mode.[57]

See also

- ANSI art
- Control character
- Advanced Video Attribute Terminal Assembler and Recreator (AVATAR)
- ISO/IEC JTC 1/SC 2

- C0 and C1 control codes

Notes

a. Typical colors that are used when booting PCs and leaving them in text mode, which used a 16-entry color table. The colors are different in the EGA/VGA graphic modes.
b. Seen in Windows XP through Windows 8.1
c. PowerShell's default shortcut .lnk, unchanged for over a decade, remaps yellow and magenta to give PowerShell distinctive foreground/background colors compared to the Command Prompt.[30] PowerShell 7 is unaffected.
d. Debug console, "Dark+" theme
e. Campbell theme, used as of Windows 10 version 1709.
f. For virtual terminals, from /etc/vtrgb.
g. On terminals based on CGA compatible hardware, such as ANSI.SYS running on DOS, this normal intensity foreground color is rendered as Orange. CGA RGBI monitors contained hardware to modify the dark yellow color to an orange/brown color by reducing the green component. See this ansi art (http://sixteencolors.net/pack/ciapak26/DH-JNS11.CIA) Archived (https://web.archive.org/web/20110725014401/http://sixteencolors.net/pack/ciapak26/DH-JNS11.CIA) 25 July 2011 at the Wayback Machine as an example.

References

1. "Standard ECMA-48: Control Functions for Character-Imaging I/O Devices" (https://www.ecma-international.org/wp-content/uploads/ECMA-48_2nd_edition_august_1979.pdf) (PDF) (Second ed.). Ecma International. August 1979. Brief History.
2. Williams, Paul (2006). "Digital's Video Terminals" (https://vt100.net/dec/vt_history). VT100.net. Retrieved 17 August 2011.
3. Heathkit Company (1979). "Heathkit Catalog 1979" (https://web.archive.org/web/20120113230301/http://www.pestingers.net/Computer_history/Computers_79.htm). Heathkit Company. Archived from the original (http://www.pestingers.net/Computer_history/Computers_79.htm) on 13 January 2012. Retrieved 4 November 2011.
4. "Withdrawn FIPS Listed by Number" (https://www.nist.gov/system/files/documents/2016/12/15/withdrawn_fips_by_numerical_order_index.pdf) (PDF). *National Institute of Standards and Technology*. 15 December 2016. Retrieved 2 January 2022.
5. "Standard ECMA-48: Control Functions for Coded Character Sets" (https://www.ecma-international.org/wp-content/uploads/ECMA-48_5th_edition_june_1991.pdf) (PDF) (Fifth ed.). Ecma International. June 1991.
6. "Amiga Printer Command Definitions" (http://wiki.amigaos.net/wiki/Printer_Device#Printer_Command_Definitions). Commodore. Retrieved 10 July 2013.
7. Hood, Jason (2005). "Process ANSI escape sequences for Windows console programs" (https://github.com/adoxa/ansicon). Jason Hood's Home page. Retrieved 9 May 2013.
8. "colorama · PyPI" (https://pypi.python.org/pypi/colorama). *Python Package Index*. Retrieved 27 February 2022.
9. bitcrazed. "Console Virtual Terminal Sequences - Windows Console" (https://docs.microsoft.com/en-us/windows/console/console-virtual-terminal-sequences). *docs.microsoft.com*. Retrieved 30 May 2018.
10. "PowerShell Help: About Special Characters" (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_special_characters?view=powershell-6). 12 January 2023.
11. "Windows Console and Terminal Ecosystem Roadmap" (https://docs.microsoft.com/en-us/windows/console/ecosystem-roadmap#roadmap-for-the-future). Microsoft. 2018. Retrieved 13 March 2021. "this includes retiring the classic console host user interface from its default position in favor of Windows Terminal, ConPTY, and virtual terminal sequences."
12. "Standard ECMA-35: Character Code Structure and Extension Techniques" (https://www.ecma-international.org/wp-content/uploads/ECMA-35_6th_edition_december_1994.pdf) (PDF) (Sixth ed.). Ecma International. 1994.
13. Moy, Edward; Gildea, Stephen; Dickey, Thomas (2019). "XTerm Control Sequences (ctlseqs)" (https://invisible-island.net/xterm/ctlseqs/ctlseqs.html). *Invisible Island*.
14. Frank da Cruz; Christine Gianone (1997). *Using C-Kermit* (https://books.google.com/books?id=QuYQPc-c67MC&pg=PA278). Digital Press. p. 278. ISBN 978-1-55558-164-0.

15. "SCOSC—Save Current Cursor Position" (https://vt100.net/docs/vt510-rm/SCOSC.html). *VT510 Video Terminal Programmer Information*. DEC.
16. "DECSLRM—Set Left and Right Margins" (https://vt100.net/docs/vt510-rm/DECSLRM.html). *VT510 Video Terminal Programmer Information*. DEC.
17. "SCORC—Restore Saved Cursor Position" (https://vt100.net/docs/vt510-rm/SCORC.html). *VT510 Video Terminal Programmer Information*. DEC.
18. Moy, Edward; Gildea, Stephen; Dickey, Thomas. "XTerm Control Sequences" (https://invisible-island.net/xterm/ctlseqs/ctlseqs.html#h3-Functions-using-CSI-_-ordered-by-the-final-character_s_). Functions using CSI, ordered by the final character(s). Retrieved 4 February 2022.
19. Conrad Irwin (April 2013). "bracketed paste mode" (https://cirw.in/blog/bracketed-paste). *cirw.in*.
20. Moy, Edward; Gildea, Stephen; Dickey, Thomas. "XTerm Control Sequences" (https://invisible-island.net/xterm/ctlseqs/ctlseqs.html#h2-Bracketed-Paste-Mode). Bracketed Paste Mode. Retrieved 4 February 2022.
21. "console_codes(4) - Linux manual page" (http://man7.org/linux/man-pages/man4/console_codes.4.html). *man7.org*. Retrieved 23 March 2018.
22. "screen(HW)" (http://osr507doc.sco.com/man/html.HW/screen.HW.html). *SCO OpenServer Release 5.0.7 Manual*. 11 February 2003.
23. "Bug 791596 – Thoughts about faint (SGR 2)" (https://bugzilla.gnome.org/show_bug.cgi?id=791596). *bugzilla.gnome.org*.
24. "Curly and colored underlines (#6382) · Issues · George Nachman / iterm2" (https://gitlab.com/gnachman/iterm2/-/issues/6382). *GitLab*. 11 December 2017.
25. "Extensions to the xterm protocol" (https://sw.kovidgoyal.net/kitty/protocol-extensions.html#colored-and-styled-underlines). *kitty documentation*. Retrieved 1 July 2020.
26. "Curly and colored underlines" (https://bugs.kde.org/387811). *KDE bugtracker*. 27 August 2022.
27. "console-termio-realizer" (http://jdebp.uk/Softwares/nosh/guide/commands/console-termio-realizer.xml). *jdebp.uk*.
28. "console_codes(4)" (https://man7.org/linux/man-pages/man4/console_codes.4.html). *Linux Programmer's Manual*. 5.10. Linux man-pages project.
29. "mintty/mintty: Text attributes and rendering" (https://github.com/mintty/mintty/wiki/Tips#text-attributes-and-rendering). *GitHub*.
30. "default shortcut on Windows remaps ANSI colors 35,36 · Issue #4266 · PowerShell/PowerShell" (https://github.com/PowerShell/PowerShell/issues/4266). *GitHub*. Retrieved 21 December 2022.
31. Changed from 0, 0, 205 in July 2004 "Patch #192 – 2004/7/12 – XFree86 4.4.99.9" (http://invisible-island.net/xterm/xterm.log.html#xterm_192).
32. Changed from 0, 0, 255 in July 2004 "Patch #192 – 2004/7/12 – XFree86 4.4.99.9" (http://invisible-island.net/xterm/xterm.log.html#xterm_192).
33. "T.416 Information technology - Open Document Architecture (ODA) and interchange format: Character content architectures" (https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.416-199303-I!!PDF-E&type=items).
34. "color-spaces.pl (a copy of 256colors2.pl from xterm dated 1999-07-11)" (https://quickgit.kde.org/?p=konsole.git&a=blob&f=tests%2Fcolor-spaces.pl). KDE. 6 December 2006.
35. "README.moreColors" (https://invent.kde.org/personalizedrefrigerator/konsole/-/blob/master/doc/user/README.moreColors). KDE. 22 April 2010.
36. "libvte's bug report and patches: Support for 16 million colors" (https://bugzilla.gnome.org/show_bug.cgi?id=704449). GNOME Bugzilla. 4 April 2014. Retrieved 5 June 2016.
37. "How to enable 24bit true color support in Gnome terminal?" (https://askubuntu.com/questions/512525/how-to-enable-24bit-true-color-support-in-gnome-terminal).
38. "History". *curs_color(3X) - manipulate terminal colors with curses* (https://invisible-island.net/ncurses/man/curs_color.3x.html#h2-HISTORY).
39. "Midnight Commander: lib/tty/color-slang.c" (https://fossies.org/linux/mc/lib/tty/color-slang.c). *Fossies*.
40. Dickey, Thomas E. (2017). "NCURSES — comments on S-Lang" (https://invisible-island.net/ncurses/ncurses-slang.html). *invisible-island.net*.
41. "Features/256 Color Terminals - Fedora Project Wiki" (https://fedoraproject.org/wiki/Features/256_Color_Terminals#Scope). *Fedora Project*. Archived (https://web.archive.org/web/20121004184942/https://fedoraproject.org/wiki/Features/256_Color_Terminals#Scope) from the original on 4 October 2012.

42. "Bug 733423 – Please set COLORFGBG environment variable for automatic color detection" (https://bugzilla.gnome.org/show_bug.cgi?id=733423). *bugzilla.gnome.org*.
43. Koblinger, Egmont. "Hyperlinks (a.k.a. HTML-like anchors) in terminal emulators" (https://gist.github.com/egmontkob/eb114294efbcd5adb1944c9f3cb5feda). *GitHub Gists*.
44. "mintty/mintty: Control Sequences" (https://github.com/mintty/mintty/wiki/CtrlSeqs#hyperlinks). *GitHub*.
45. Koblinger, Egmont. "OSC 8 adoption in terminal emulators" (https://github.com/Alhadis/OSC8-Adoption). *GitHub*.
46. "console_codes — Linux console escape and control sequences" (https://man7.org/linux/man-pages/man4/console_codes.4.html). *Linux Programmer's Manual*.
47. *ISO-IR: ISO/IEC International Register of Coded Character Sets To Be Used With Escape Sequences* (https://itscj.ipsj.or.jp/english/vbcqpr00000004qn-att/ISO-IR.pdf) (PDF). ITSCJ/IPSJ. Retrieved 12 May 2023.
48. Digital. "DECSC—Save Cursor" (https://vt100.net/docs/vt510-rm/DECSC.html). *VT510 Video Terminal Programmer Information*.
49. *ANSI Escape sequences - VT100 / VT52* (https://web.archive.org/web/20090227051140/http://ascii-table.com/ansi-escape-sequences-vt-100.php), archived from the original (http://ascii-table.com/ansi-escape-sequences-vt-100.php) on 27 February 2009, retrieved 21 August 2020
50. Digital. "DECSWL—Single-Width, Single-Height Line" (https://vt100.net/docs/vt510-rm/DECSWL.html). *VT510 Video Terminal Programmer Information*.
51. Digital. "DECDWL—Double-Width, Single-Height Line" (https://vt100.net/docs/vt510-rm/DECDWL.html). *VT510 Video Terminal Programmer Information*.
52. "Chapter 9. System tips" (http://www.debian.org/doc/manuals/debian-reference/ch09.en.html#_colorized_shell_echo). *debian.org*.
53. "VT100.net: Digital VT100 User Guide" (http://vt100.net/docs/vt100-ug/chapter3.html). Retrieved 19 January 2015.
54. "bash – How to get a notification when my commands are done – Ask Different" (http://apple.stackexchange.com/questions/9412/how-to-get-a-notification-when-my-commands-are-done). Retrieved 19 January 2015.
55. Dickey, Thomas. "XTerm FAQ: Comparing versions, by counting controls" (https://invisible-island.net/xterm/xterm.faq.html#compare_versions). *Invisible Island*. Retrieved 25 January 2020.
56. Dickey, Thomas (2016). "Table of function-keys for XTerm and other Terminal Emulators" (https://invisible-island.net/xterm/xterm-function-keys.html). *Invisible Island*. Retrieved 25 January 2020.
57. "CSI u - Documentation - iTerm2 - macOS Terminal Replacement" (https://iterm2.com/documentation-csiu.html). *iTerm2*. Retrieved 15 August 2023.

External links

- Standard ECMA-48, Control Functions For Coded Character Sets (https://www.ecma-international.org/publications-and-standards/standards/ecma-48/). (*5th edition, June 1991*), European Computer Manufacturers Association, Geneva 1991 (also published by ISO and IEC as standard ISO/IEC 6429)
- vt100.net DEC Documents (http://vt100.net/docs/)
- "ANSI.SYS -- ansi terminal emulation escape sequences" (https://web.archive.org/web/20060206022229/http://enterprise.aacc.cc.md.us/~rhs/ansi.html). Archived from the original (http://enterprise.aacc.cc.md.us/~rhs/ansi.html) on 6 February 2006. Retrieved 22 February 2007.
- Xterm / Escape Sequences (http://invisible-island.net/xterm/ctlseqs/ctlseqs.html)
- AIXterm / Escape Sequences (https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/a_commands/aixterm.html)
- A collection of escape sequences for terminals that are vaguely compliant with ECMA-48 and friends. (http://bjh21.me.uk/all-escapes/all-escapes.txt)
- "ANSI Escape Sequences" (https://web.archive.org/web/20110525032501/http://ascii-table.com/ansi-escape-sequences.php). Archived from the original (http://ascii-table.com/ansi-escape-sequences.php) on 25 May 2011.
- ITU-T Rec. T.416 (03/93) Information technology – Open Document Architecture (ODA) and interchange format: Character content architectures (https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.416-199303-I!!PDF-E&type=items)

Retrieved from "https://en.wikipedia.org/w/index.php?title=ANSI_escape_code&oldid=1220675362"