ANSI转义序列

维基百科，自由的百科全书

ANSI转义序列（ANSI escape sequences）是一种带内信号的转义序列标准，用于控制视频文本终端上的光标位置、颜色和其他选项。在文本中嵌入确定的字节序列，大部分以ESC转义字符和"["字符开始，终端会把这些字节序列解释为相应的指令，而不是普通的字符编码。

ANSI序列是在二十世纪七十年代引入的标准，用以取代特定于终端供应商的序列，并在二十世纪八十年代早期开始在计算机设备市场上广泛使用。与早期缺少光标移动功能的系统相比，新生的电子公告板系统使用ANSI序列改进其显示。正是因为这个原因，ANSI序列变成了所有制造商共同采用的标准。

在21世纪，尽管硬件文本终端已经越来越少了，但ANSI标准依然存在，因为大多数终端模拟器会对部分ANSI转义序列进行解释。一个值得注意的例外是，在微软Windows 10更新TH2之前，Windows操作系统的Win32控制台是不支持ANSI转义序列的。

目录

历史

最初，几乎每个视频终端制造商都各自添加了特定的转义序列用于执行一些特殊操作，比如把光标置于屏幕上的某个位置。举例来说，VT52终端允许通过发送ESC字符、y字符，后面跟上两个等于x,y位置的数值加上32的字符（这是为了从ASCII空格字符开始，并避开控制字符），将光标置于屏幕上的x,y位置。

由于这些序列对于不同的终端并不一样，因此人们不得不开发了一些复杂的库（比如termcap）和实用程序（比如tput），以便程序可以使用同一套API应对各种终端。另外，在很多终端中需要借助字符的二进制值发送数字（如行和列）。对于某些编程语言，以及内部不使用ASCII的系统来说，把数字转换为正确的字符常常是有困难的，甚至完全做不到。

ANSI标准试图解决这些问题。标准制订了一种所有终端共用的指令集，并要求用ASCII的数字字符传递所有数值信息。该系列的第一个标准是1976年通过的ECMA-48。它是一系列字符编码标准的延续，其中第一个是从1965年的ECMA-6，一个7位标准，ISO 646就源自此标准。“ANSI转义序列”的名称可以追溯到1979年ANSI采用ANSI X3.64。此外，ANSI X3L2委

员会与ECMA委员会TC 1合作制订了一个几乎一模一样的标准。以上两个标准合并为ISO 6429的国际标准[1]。1994年，ANSI取消了其标准，以支持国际标准。

第一个支持这个标准的流行视频终端是1978年推出的Digital VT100[2]。这个终端在市场上非常成功，引发了各种各样的仿制品，其中最早和最流行的是1979年的Zenith Z-19[3]。其他品牌还有Qume QVT-108，Televideo TVI-970，Wyse WY-99GT。另外，许多其他品牌的终端也不同程度地兼容可选的“VT100”、“VT103”或“ANSI”模式。 随着越来越多的软件（尤其是BBS系统）普及，越来越多的软件依赖转义序列起作用，导致几乎所有新的终端和终端模拟器都支持了此标准。

1981年，ANSI X3.64被美国政府采用（FIPS 86）。后来，美国政府停止复制行业标准，所以FIPS 86又被撤回了[4]。

ECMA-48已经经历了多次更新换代，目前是从1991年开始的第5版。它也被ISO和IEC用作标准**ISO/IEC 6429**。

平台支持

随着诸多BBS和线上服务广泛使用ANSI，到20世纪80年代中期，ANSI几乎得到了全平台支持。尽管许多操作系统在标准文本输出中越来越多地支持ANSI，但大多数情况下是以终端模拟器的形式（例如Unix上的xterm，或MacOS上的OS X Terminal或ZTerm，以及IBM PC上的许多通信程序）。

Unix和AmigaOS都在操作系统中包含了对ANSI的一些支持，导致在这些平台上运行的程序广泛使用ANSI。 类Unix操作系统可以通过像termcap和curses函式庫之类的库来生成ANSI代码，许多软件使用这些库升级显示方式。这些库也应该支持非ANSI终端，但是现在很少有人测试，所以很可能已经不起作用了。许多游戏和shell脚本直接输出ANSI序列（如彩色的提示信息），因此无法在不支持ANSI的终端上运行。

AmigaOS不仅支持输出到屏幕上的文本使用ANSI序列，打印机驱动程序也支持（用AmigaOS的专有扩展），并将它们转换为与特定打印机实际通信所需的代码[5]。

尽管ANSI很普及，却并没有得到全平台支持。比如原始的“经典”Mac OS就没有内置对ANSI的支持，再比如Atari ST使用的是VT52改编的命令系统，用一些扩展程序支持颜色显示[6]。

Windows和DOS

MS-DOS 1.x不支持ANSI或任何其他转义序列，只有少数控制字符（BEL、CR、LF、BS）可以由底层BIOS解释，所以几乎[nb 1]不可能做出任何全屏应用程序。所有显示效果都必须通过BIOS调用，或者直接控制IBM PC硬件来完成，调用速度非常慢。

DOS 2.0引入了添加设备驱动程序来支持ANSI转义序列的功能（事实上的标准是ANSI.SYS，但也使用了ANSI.COM[7]、NANSI.SYS[8]和ANSIPLUS.EXE等其他程序。因为绕过了BIOS，所以这些程序的速度比以前快了不少）。但由于实际运行速度仍然比较慢，以及默认并没有安装，所以还是很少得到利用。应用程序往往还是继续用直接控制硬件的方式来显示所需的文本。ANSI.SYS和类似的驱动程序继续在Windows 9x上工作，直到Windows Me，在NT衍生系统中用于在NTVDM下执行的16位传统程序。

Win32控制台完全不支持ANSI转义序列。不过有一些控制台的替代品或者附加软件具有解释程序输出的ANSI转义序列的功能，例如JP Software的TCC（以前的4NT）、Michael J. Mefford的ANSI.COM、Jason Hood的ANSICON[9]和Maximus5的ConEmu。有一个Python软件包[10]在内部解释了打印文本中的ANSI转义序列，将它们转换为系统调用来操纵颜色和光标位置，以便更容易地将使用ANSI的Python代码移植到Windows。

2016年，在Windows 10发布“Threshold 2”[11]时，微软开始在控制台应用程序中支持ANSI转义序列，使得从Unix移植软件或者远程访问Unix变得更容易。

转义序列

序列具有不同的长度。所有序列都以ASCII字符**ESC**（27 / 十六进制 0x1B）开头，第二个字节则是0x40–0x5F（ASCII `@A–Z[\]^_`）范围内的字符。[12]:5.3.a

标准规定，在8位环境中，这两个字节的序列可以合并为0x80-0x9F范围内的单个字节（详情请参阅C1控制字符集）。但是，在现代设备上，这些代码通常用于其他目的，例如UTF-8的一部分或CP-1252字符，因此并不使用这种合并的方式。

除ESC之外的其他C0代码（通常是BEL，BS，CR，LF，FF，TAB，VT，SO和SI）在输出时也可能会产生与某些控制序列相似或相同的效果。

一些ANSI转义序列（不完整列表）

序列	C1	名称	作用
ESC N	0x8e	SS2 – Single Shift Two	从其中一个替代字符集中选择一个字符。在xterm中，SS2选择G2字符集，SS3选择G3字符集。[13]
ESC O	0x8f	SS3 – Single Shift Three	
ESC P	0x90	DCS – 设备控制字符串（Device Control String）	控制设备。在xterm中，这个序列的使用包括定义用户自定义的密钥，以及请求或设置Termcap/Terminfo数据。[13]
ESC [	0x9b	CSI - 控制序列导入器（Control Sequence Introducer）	大部分有用的序列，请参阅下一节。结束于ASCII 64到126 (@到~/十六进制0x40到0x7E).[12]
ESC \	0x9c	ST – 字符串终止（String Terminator）	终止其他控件（包括APC，DCS，OSC，PM和SOS）中的字符串。[12]:8.3.143
ESC]	0x9d	OSC – 操作系统命令（Operating System Command）	启动操作系统使用的控制字符串。OSC序列与CSI序列相似，但不限于整数参数。通常，这些控制序列由ST终止[12]:8.3.89。在xterm中，它们也可能被BEL终止[13]。例如，在xterm中，窗口标题可以这样设置：`OSC 0;this is the window title BEL`。
ESC X	0x98	SOS – 字符串开始（Start of String）	引用由ST终止的一串文本的参数。这些字符串控制序列的用途由应用程序[12]:8.3.2,8.3.128或私有规则来定义[12]:8.3.94。这些函数没有实现，参数被xterm忽略[13]。
ESC ^	0x9e	PM – 私有消息（Privacy Message）	
ESC _	0x9f	APC – 应用程序命令（Application Program Command）	
ESC c		RIS – 重置为初始状态（Reset to Initial State）	将设备重置为原始状态。可能包括（如果适用的话）：重置图形格式，清除制表符，重置为默认字体等等。

按下键盘上的特殊键，以及输出xterm CSI、DCS或OSC序列，常常用于产生从终端发送到计算机的CSI，DCS或OSC序列，就像用户使用键盘输入的一样。

CSI序列

CSI序列由`ESC [`、若干个（包括0个）“参数字节”、若干个“中间字节”，以及一个“最终字节”组成。各部分的字符范围如下：

CSI序列在`ESC [`之后各个组成部分的字符范围[12]:5.4

组成部分	字符范围	ASCII
参数字节	0x30–0x3F	`0-9:;<=>?`
中间字节	0x20–0x2F	空格、`!"#$%&'()*+,-./`
最终字节	0x40–0x7E	`@A-Z[\]^_`a-z{|}~`

所有常见的序列都只是把参数用作一系列分号分隔的数字，如`1;2;3`。缺少的数字视为0（如`1;;3`相当于中间的数字是0，`ESC[m`这样没有参数的情况相当于参数为0）。某些序列（如CUU）把0视为1，以使缺少参数的情况下有意义:F.4.2。

一部分字符定义是“私有”的，以便终端制造商可以插入他们自己的序列而不与标准相冲突。包括参数字节`<=>?`的使用，或者最终字节0x70–0x7F（`p-z{|}~`）例如VT320序列`CSI?25h`和`CSI?25l`的作用是打开和关闭光标的显示。

当CSI序列含有超出0x20–0x7E范围的字符时，其行为是未定义的。这些非法字符包括C0控制字符（范围0–0x1F）、DEL（0x7F），以及高位字节。

一些ANSI控制序列（不完整列表）

代码	名称	作用
CSI n A	CUU – 光标上移（Cursor Up）	光标向指定的方向移动***n***（默认1）格。如果光标已在屏幕边缘，则无效。
CSI n B	CUD – 光标下移（Cursor Down）	
CSI n C	CUF – 光标前移（Cursor Forward）	
CSI n D	CUB – 光标后移（Cursor Back）	
CSI n E	CNL – 光标移到下一行（Cursor Next Line）	光标移动到下面第***n***（默认1）行的开头。（非ANSI.SYS）
CSI n F	CPL – 光标移到上一行（Cursor Previous Line）	光标移动到上面第***n***（默认1）行的开头。（非ANSI.SYS）
CSI n G	CHA – 光标水平绝对（Cursor Horizontal Absolute）	光标移动到第***n***（默认1）列。（非ANSI.SYS）
CSI n ; m H	CUP – 光标位置（Cursor Position）	光标移动到第***n***行、第***m***列。值从1开始，且默认为1（左上角）。例如`CSI ;5H`和`CSI 1;5H`含义相同；`CSI 17;H`、`CSI 17H`和`CSI 17;1H`三者含义相同。
CSI n J	ED – 擦除显示（Erase in Display）	清除屏幕的部分区域。如果***n***是0（或缺失），则清除从光标位置到屏幕末尾的部分。如果***n***是1，则清除从光标位置到屏幕开头的部分。如果***n***是2，则清除整个屏幕（在DOS ANSI.SYS中，光标还会向左上方移动）。如果***n***是3，则清除整个屏幕，并删除回滚缓存区中的所有行（这个特性是xterm添加的，其他终端应用程序也支持）。
CSI n K	EL – 擦除行（Erase in Line）	清除行内的部分区域。如果***n***是0（或缺失），清除从光标位置到该行末尾的部分。如果***n***是1，清除从光标位置到该行开头的部分。如果***n***是2，清除整行。光标位置不变。
CSI n S	SU – 向上滚动（Scroll Up）	整页向上滚动***n***（默认1）行。新行添加到底部。（非ANSI.SYS）
CSI n T	SD – 向下滚动（Scroll Down）	整页向下滚动***n***（默认1）行。新行添加到顶部。（非ANSI.SYS）
CSI n ; m f	HVP – 水平垂直位置（Horizontal Vertical Position）	同CUP。
CSI n m	SGR – 选择图形再现（Select Graphic Rendition）	设置SGR参数，包括文字颜色。CSI后可以是0或者更多参数，用分号分隔。如果没有参数，则视为`CSI 0 m`（重置/常规）。
CSI 5i	打开辅助端口	启用辅助串行端口，通常用于本地串行打印机
CSI 4i	关闭辅助端口	禁用辅助串行端口，通常用于本地串行打印机
CSI 6n	DSR – 设备状态报告（Device Status Report）	以`ESC[n;mR`（就像在键盘上输入）向应用程序报告光标位置（CPR），其中***n***是行，***m***是列。

CSI s	SCP – 保存光标位置（Save Cursor Position）	保存光标的当前位置。
CSI u	RCP – 恢复光标位置（Restore Cursor Position）	恢复保存的光标位置。

选择图形再现（SGR）参数

代码	作用	备注
0	重置/正常	关闭所有属性。
1	粗体或增加强度	
2	弱化（降低强度）	未广泛支持。
3	斜体	未广泛支持。有时视为反相显示。
4	下划线	
5	缓慢闪烁	低于每分钟150次。
6	快速闪烁	MS-DOS ANSI.SYS；每分钟150以上；未广泛支持。
7	反显	前景色与背景色交换。
8	隐藏	未广泛支持。
9	划除	字符清晰，但标记为删除。未广泛支持。
10	主要（默认）字体	
11–19	替代字体	选择替代字体$\boldsymbol{n-10}$。
20	尖角体	几乎无支持。
21	关闭粗体或双下划线	关闭粗体未广泛支持；双下划线几乎无支持。
22	正常颜色或强度	不强不弱。
23	非斜体、非尖角体	
24	关闭下划线	去掉单双下划线。
25	关闭闪烁	
27	关闭反显	
28	关闭隐藏	
29	关闭划除	
30–37	设置前景色	参见下面的颜色表。
38	设置前景色	下一个参数是`5;n`或`2;r;g;b`，见下。
39	默认前景色	由具体实现定义（按照标准）。
40–47	设置背景色	参见下面的颜色表。
48	设置背景色	下一个参数是`5;n`或`2;r;g;b`，见下。
49	默认背景色	由具体实现定义（按照标准）。
51	Framed	
52	Encircled	
53	上划线	
54	Not framed or encircled	
55	关闭上划线	
60	表意文字下划线或右边线	几乎无支持。
61	表意文字双下划线或双右边线	
62	表意文字上划线或左边线	
63	表意文字双上划线或双左边线	
64	表意文字着重标志	
65	表意文字属性关闭	重置`60`–`64`的所有效果。
90–97	设置明亮的前景色	aixterm（非标准）。
100–107	设置明亮的背景色	aixterm（非标准）。

颜色

3/4位

初始的规格只有8种颜色，只给了它们的名字。SGR参数30-37选择前景色，40-47选择背景色。相当多的终端将“粗体”（SGR代码1）实现为更明亮的颜色而不是不同的字体，从而提供了8种额外的前景色，但通常情况下并不能用于背景色，虽然有时候反显（SGR代码7）可以允许这样。例如：在白色背景上显示黑色文字使用ESC[30;47m，显示红色文字用ESC[31m，显示明亮的红色文字用ESC[1;31m。重置为默认颜色用ESC[39;49m（某些终端不支持），重置所有属性用ESC[0m。后来的终端新增了功能，可以直接用90-97和100-107指定“明亮”的颜色。

当硬件开始使用8位DAC时，多个软件为这些颜色名称分配了24位的代码。下面的图表显示了发送到DAC的一些常用硬件和软件的值。

名称	前景色代码	背景色代码	VGA[nb 2]	CMD[nb 3]	Terminal.app	PuTTY	mIRC	xterm	X[nb 4]	Ubuntu[nb 5]
黑	30	40	0,0,0							1,1,1
红	31	41	170,0,0	128,0,0	194,54,33	187,0,0	127,0,0	205,0,0	255,0,0	222,56,43
绿	32	42	0,170,0	0,128,0	37,188,36	0,187,0	0,147,0	0,205,0	0,255,0	57,181,74
黄	33	43	170,85,0[nb 6]	128,128,0	173,173,39	187,187,0	252,127,0	205,205,0	255,255,0	255,199,6
蓝	34	44	0,0,170	0,0,128	73,46,225	0,0,187	0,0,127	0,0,238[14]	0,0,255	0,111,184
品红	35	45	170,0,170	128,0,128	211,56,211	187,0,187	156,0,156	205,0,205	255,0,255	118,38,113
青	36	46	0,170,170	0,128,128	51,187,200	0,187,187	0,147,147	0,205,205	0,255,255	44,181,233
白	37	47	170,170,170	192,192,192	203,204,205	187,187,187	210,210,210	229,229,229	255,255,255	204,204,204
亮黑（灰）	90	100	85,85,85	128,128,128	129,131,131	85,85,85	127,127,127	127,127,127		128,128,128
亮红	91	101	255,85,85	255,0,0	252,57,31	255,85,85	255,0,0	255,0,0		255,0,0
亮绿	92	102	85,255,85	0,255,0	49,231,34	85,255,85	0,252,0	0,255,0	144,238,144	0,255,0
亮黄	93	103	255,255,85	255,255,0	234,236,35	255,255,85	255,255,0	255,255,0	255,255,224	255,255,0
亮蓝	94	104	85,85,255	0,0,255	88,51,255	85,85,255	0,0,252	92,92,255[15]	173,216,230	0,0,255
亮品红	95	105	255,85,255	255,0,255	249,53,248	255,85,255	255,0,255	255,0,255		255,0,255
亮青	96	106	85,255,255	0,255,255	20,240,240	85,255,255	0,255,255	0,255,255	224,255,255	0,255,255
亮白	97	107	255,255,255	255,255,255	233,235,235	255,255,255	255,255,255	255,255,255		255,255,255

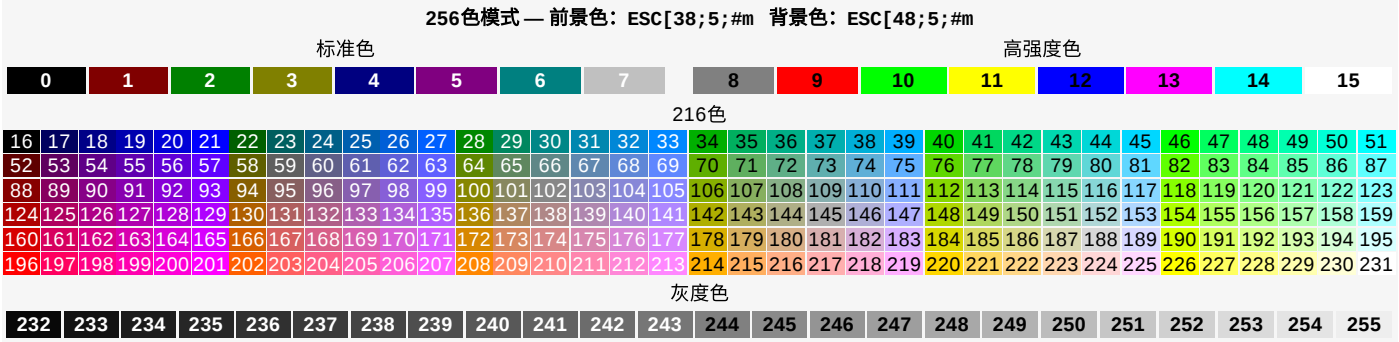
8位

随着256色查找表在显卡上越来越常见，相应的转义序列也增加了，以从预定义的256种颜色中选择：[16]

```
ESC[ … 38;5;<n> … m选择前景色
ESC[ … 48;5;<n> … m选择背景色
  0-  7: 标准颜色（同ESC [ 30–37 m）
  8- 15: 高强度颜色（同ESC [ 90–97 m）
 16-231：6 × 6 × 6 立方（216色）：16 + 36 × r + 6 × g + b (0 ≤ r, g, b ≤ 5)
232-255: 从黑到白的24阶灰度色
```

ITU的T.416信息技术-开放文档体系结构（ODA）和交换格式：字符内容体系结构[17]使用“:”作为分隔符：

```
ESC[ … 38:5:<n> … m选择前景色
ESC[ … 48:5:<n> … m选择背景色
```



24位

随着16位到24位颜色的“真彩色”显卡的普及，Xterm[13]、KDE的Konsole[18]，以及所有基于libvte的终端[19]（包括GNOME终端）支持了ISO-8613-3的24位前景色和背景色设置。[16]

```
ESC[ … 38;2;<r>;<g>;<b> … m选择RGB前景色
ESC[ … 48;2;<r>;<g>;<b> … m选择RGB背景色
```

作为ISO / IEC国际标准8613-6采用的ITU的T.416信息技术-开放文档体系结构（ODA）和交换格式：字符内容体系结构[20]给出了一个似乎不太受支持的替代版本：

```
ESC[ … 38:2:<Color-Space-ID>:<r>:<g>:<b>:<unused>:<CS tolerance>:<Color-Space: 0="CIELUV"; 1="CIELAB">m选择RGB前景色
ESC[ … 48:2:<Color-Space-ID>:<r>:<g>:<b>:<unused>:<CS tolerance>:<Color-Space: 0="CIELUV"; 1="CIELAB">m选择RGB背景色
```

请注意，这里使用了保留的“:”字符来分隔子选项，这可能是在实际实现中造成混淆的始作俑者。它还使用“3”作为第二个参数来指定使用青-品红-黄方案的方案，“4”用于青-品红-黄-黑的方案，后者使用上面标记为“unused”（“未使用”）的位置作为黑色组件。

还要注意，许多识别“:”作为分隔符的实现错误地忽视了色彩空间标识符参数，并因此改变了其余部分的位置。

示例

`CSI 2 J` — 清除屏幕、（在某些设备上）把光标置于1,1位置（左上角）。

`CSI 32 m` — 使文字呈绿色。在MS-DOS上，一般绿色是暗淡的绿色，可以用`CSI 1 m`启用粗体使其变成明亮的绿色，或者将两者合并为`CSI 32 ; 1 m`。MS-DOS ANSI.SYS用粗体状态使字符变亮，闪烁状态（通过`INT 10, AX 1003h, BL 00h`)使背景色变成明亮模式。MS-DOS ANSI.SYS并不直接支持SGR代码90–97和100–107。

`CSI 0 ; 6 8 ; "DIR" ; 13 p` — 重新分配F10键的功能为发送字符串“DIR”和回车符到键盘缓存中，在DOS命令行里会显示当前目录的内容（仅MS-DOS ANSI.SYS）。这种序列有时用于“ANSI炸弹”。这是一个私用编码（如字母p所示），用非标准的扩展使其包含一个字符串参数。如果按标准，会认为字母D是序列的末尾。

`CSI s` — 保存光标的位置。用序列`CSI u`会把光标重置回这个位置。假设当前的光标位置是7（y）、10（x）。序列`CSI s`会保存这两个数值。现在可以把光标移动到其他位置，比如用序列`CSI 20 ; 3 H`或`CSI 20 ; 3 f`把光标移动到20（y）、3（x）。现在如果用序列`CSI u`，光标会回到7（y）、10（x）。某些终端需要使用DEC序列`ESC 7`/`ESC 8`，这得到了更广泛的支持。

使用Shell脚本的示例

ANSI转移代码常常用于UNIX和类UNIX终端，以提供语法高亮功能。例如，在兼容的终端上，以下ls命令按类型对文件和目录的名称进行颜色编码。

```
ls --color
```

用户可以在脚本中使用转义码，将其作为标准输出或标准错误输出的一部分。例如，下面的GNU sed命令通过反显“WARN”开头的单词的行，以及使用暗红色背景色和亮黄色前景色显示以“ERR”开头的单词（字母大小写被忽略）的行来修饰make命令的输出。突出显示了设置ANSI代码的部分。[21]

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

以下Bash函数会使终端闪烁（通过交替发送反相和正常显示模式代码），直到用户按下任意键[22]。这个函数可以用于当一个冗长的命令终止时提醒用户，用法如`make; flasher`[23]。

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

下面这个命令可以重置控制台，类似现代Linux系统的`reset`命令。然而，即使在较早的Linux系统和其他（非Linux）UNIX变体上，也应该能起作用。

```
printf \\033c
```

参见

- 控制字符

脚注

1. 屏幕显示可以通过从底部绘制整个新屏幕的内容来替代，滚动上一屏幕以充分擦除所有旧文本内容。用户会看到滚动，硬件光标会留在最底部。 一些早期的批处理文件以这种方式实现了基本的“全屏”显示。
2. 在启动PC并将其保留在文本模式下时使用的典型颜色，使用16个条目的颜色表。EGA/VGA图形模式中的颜色不同。
3. 从Windows XP起
4. 以上颜色名称来自X11 rgb.txt颜色数据库，用“light”作为明亮颜色的前缀。
5. 用于虚拟终端，来自/etc/vtrgb。
6. 在基于CGA兼容硬件（如在DOS上运行的ANSI.SYS）的终端上，正常强度的前景色呈现为橙色。 CGA RGBI显示器包含一些硬件，通过减少绿色成分将深黄色修改为橙色/棕色。

参考资料

1. Historical version of ECMA-48 (PDF). [2018-01-12]. （原始内容存档 (PDF)于2016-11-09）.
2. Williams, Paul. Digital's Video Terminals. VT100.net. 2006 [2011-08-17]. （原始内容存档于2012-07-23）.
3. Heathkit Company. Heathkit Catalog 1979. Heathkit Company. 1979 [2011-11-04]. （原始内容存档于2012-01-13）.
4. Withdrawn FIPS Listed by Number (PDF). [2018-01-12]. （原始内容存档 (PDF)于2016-06-14）.
5. Amiga Printer Command Definitions. Commodore. [2013-07-10]. （原始内容存档于2014-07-14）.
6. "Using C-Kermit" (https://books.google.com/books?id=Z0ejBQAAQBAJ&pg=PA88), p. 88.
7. Mefford, Michael. ANSI.com: Download It Here. PC Magazine. 1989-02-07 [2011-08-10]. （原始内容存档于2012-01-24）.
8. Kegel, Dan; Auer, Eric. Nansi and NNansi – ANSI Drivers for MS-DOS. Dan Kegel's Web Hostel. 1999-02-28 [2011-08-10]. （原始内容存档于2018-05-19）.
9. Hood, Jason. Process ANSI escape sequences for Windows console programs. Jason Hood's Home page. 2005 [2013-05-09]. （原始内容存档于2014-07-29）.
10. colorama 0.2.5 :. Python Package Index. [2013-08-17]. （原始内容存档于2013-10-22）.
11. Grehan, Oisin. Windows 10 TH2 (v1511) Console Host Enhancements. 2016-02-04 [2016-02-10]. （原始内容存档于2016-02-09）.
12. Standard ECMA-48: Control Functions for Coded Character Sets Fifth. Ecma International. June 1991 [2018-01-12]. （原始内容存档于2008-09-18）.
13. XTerm Control Sequences. invisible-island.net. 2014-01-13 [2014-04-13]. （原始内容存档于2007-03-12）.
14. Changed from 0,0,205 in July 2004 Patch #192 – 2004/7/12 – XFree86 4.4.99.9. [2018-01-12]. （原始内容存档于2001-12-22）.

15. Changed from 0,0,255 in July 2004 Patch #192 – 2004/7/12 – XFree86 4.4.99.9. [2018-01-12]. （原始内容存档于2001-12-22）.
16. README.moreColors. KDE. 2010-04-22.
17. T.416 Information technology - Open Document Architecture (ODA) and interchange format: Character content architectures. [2018-01-12]. （原始内容存档于2016-03-03）.
18. color-spaces.pl (a copy of 256colors2.pl from xterm dated 1999-07-11). KDE. 2006-12-06.
19. libvte's bug report and patches. GNOME Bugzilla. 2014-04-04 [2016-06-05]. （原始内容存档于2015-09-03）.
20. T.416 Information technology - Open Document Architecture (ODA) and interchange format: Character content architectures. [2018-01-12]. （原始内容存档于2016-03-03）.
21. Chapter 9. System tips. debian.org. [2018-01-12]. （原始内容存档于2018-01-29）.
22. VT100.net: Digital VT100 User Guide. [2015-01-19]. （原始内容存档于2015-04-15）.
23. bash – How to get a notification when my commands are done – Ask Different. [2015-01-19]. （原始内容存档于2015-01-19）.

外部链接

- Standard ECMA-48, Control Functions For Coded Character Sets (http://www.ecma-international.org/publications/standards/Ecma-048.htm)页面存档备份 (https://web.archive.org/web/20080918044558/http://www.ecma-international.org/publications/standards/Ecma-048.htm)，存于互联网档案馆. (*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/)Archive.is的存檔 (https://archive.is/20130222130813/http://vt100.net/docs/)，存档日期2013-02-22
- ANSI.SYS -- ansi terminal emulation escape sequences. [2018-01-12]. （原始内容存档于2006-02-06）.
- Xterm / Escape Sequences (http://invisible-island.net/xterm/ctlseqs/ctlseqs.html)页面存档备份 (https://web.archive.org/web/20070312045032/http://invisible-island.net/xterm/ctlseqs/ctlseqs.html)，存于互联网档案馆
- AIXterm / Escape Sequences (http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds1%2Faixterm.htm)
- 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)页面存档备份 (https://web.archive.org/web/20110515004938/http://bjh21.me.uk/all-escapes/all-escapes.txt)，存于互联网档案馆
- ANSI Escape Sequences (http://ascii-table.com/ansi-escape-sequences.php)页面存档备份 (https://web.archive.org/web/20110525032501/http://ascii-table.com/ansi-escape-sequences.php)，存于互联网档案馆
- 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)页面存档备份 (https://web.archive.org/web/20160303224432/https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.416-199303-I!!PDF-E&type=items)，存于互联网档案馆