第十四章 自定制shell提示符

In this chapter we will look at a seemingly trivial detail — our shell prompt. This examination will reveal some of the inner workings of the shell and the terminal emulator program itself.

在这一章中，我们将会看一下表面上看来很琐碎的细节－shell 提示符。但这会揭示一些内部 shell 和 终端仿真器的工作方式。

Like so many things in Linux, the shell prompt is highly configurable, and while we have pretty much taken it for granted, the prompt is a really useful device once we learn how to control it.

和 Linux 内的许多程序一样，shell 提示符是可高度配置的，虽然我们把它相当多地看作是理所当然的， 但是我们一旦学会了怎样控制它，shell 提示符是一个真正有用的设备。

Anatomy Of A Prompt

**解剖一个提示符**

Our default prompt looks something like this:

我们默认的提示符看起来像这样：

[me@linuxbox ~]$

Notice that it contains our user name, our host name and our current working directory, but how did it get that way? Very simply, it turns out. The prompt is defined by an environment variable named PS1 (short for “prompt string one”). We can view the contents of PS1 with the echo command:

注意它包含我们的用户名，主机名和当前工作目录，但是它又是怎样得到这些东西的呢？ 结果证明非常简单。提示符是由一个环境变量定义的，叫做 PS1（是“prompt string one” 的简写）。我们可以通过 echo 命令来查看 PS1的内容。

[me@linuxbox ~]$ echo $PS1

[\u@\h \W]\$

Note: Don’t worry if your results are not exactly the same as the example above. Every Linux distribution defines the prompt string a little differently, some quite exotically.

注意：如果你 shell 提示符的内容和上例不是一模一样，也不必担心。每个 Linux 发行版 定义的提示符稍微有点不同，其中一些相当异乎寻常。

From the results, we can see that PS1 contains a few of the characters we see in our prompt such as the brackets, the at-sign, and the dollar sign, but the rest are a mystery. The astute among us will recognize these as backslash-escaped special characters like those we saw in Chapter 8. Here is a partial list of the characters that the shell treats specially in the prompt string:

从输出结果中，我们看到那个 PS1 环境变量包含一些这样的字符，比方说中括号，@符号，和美元符号， 但是剩余部分就是个谜。我们中一些机敏的人会把这些看作是由反斜杠转义的特殊字符，就像我们 在第八章中看到的一样。这里是一部分字符列表，在提示符中 shell 会特殊对待这些字符：

|  |  |
| --- | --- |
| *Table 14-1: Escape Codes Used In Shell Prompts* | |
| Sequence | Value Displayed |
| \a | ASCII bell. This makes the computer beep when it is encountered. |
| \d | Current date in day, month, date format. For example, “Mon May 26." |
| \h | Host name of the local machine minus the trailing domain name. |
| \H | Full host name. |
| \j | Number of jobs running in the current shell session. |
| \l | Name of the current terminal device. |
| \n | A newline character. |
| \r | A carriage return. |
| \s | Name of the shell program. |
| \t | Current time in 24 hour hours:minutes:seconds format. |
| \T | Current time in 12 hour format. |
| \@ | Current time in 12 hour AM/PM format. |
| \A | Current time in 24 hour hours:minutes format. |
| \u | User name of the current user. |
| \v | Version number of the shell. |
| \V | Version and release numbers of the shell. |
| \w | Name of the current working directory. |
| \W | Last part of the current working directory name. |
| \! | History number of the current command. |
| \# | Number of commands entered into this shell session. |
| \$ | This displays a “$” character unless you have superuser privileges. In that case, it displays a “#” instead. |
| \[ | Signals the start of a series of one or more non-printing characters. This is used to embed non-printing control characters which manipulate the terminal emulator in some way, such as moving the cursor or changing text colors. |
| \] | Signals the end of a non-printing character sequence. |

|  |  |
| --- | --- |
| *表14-1: Shell 提示符中用到的转义字符* | |
| 序列 | 显示值 |
| \a | 以 ASCII 格式编码的铃声 . 当遇到这个转义序列时，计算机会发出嗡嗡的响声。 |
| \d | 以日，月，天格式来表示当前日期。例如，“Mon May 26.” |
| \h | 本地机的主机名，但不带末尾的域名。 |
| \H | 完整的主机名。 |
| \j | 运行在当前 shell 会话中的工作数。 |
| \l | 当前终端设备名。 |
| \n | 一个换行符。 |
| \r | 一个回车符。 |
| \s | shell 程序名。 |
| \t | 以24小时制，hours:minutes:seconds 的格式表示当前时间. |
| \T | 以12小时制表示当前时间。 |
| \@ | 以12小时制，AM/PM 格式来表示当前时间。 |
| \A | 以24小时制，hours:minutes 格式表示当前时间。 |
| \u | 当前用户名。 |
| \v | shell 程序的版本号。 |
| \V | Version and release numbers of the shell. |
| \w | 当前工作目录名。 |
| \W | 当前工作目录名的最后部分。 |
| \! | 当前命令的历史号。 |
| \# | 当前 shell 会话中的命令数。 |
| \$ | 这会显示一个"$"字符，除非你拥有超级用户权限。在那种情况下， 它会显示一个"#"字符。 |
| \[ | 标志着一系列一个或多个非打印字符的开始。这被用来嵌入非打印 的控制字符，这些字符以某种方式来操作终端仿真器，比方说移动光标或者是更改文本颜色。 |
| \] | 标志着非打印字符序列结束。 |

Trying Some Alternate Prompt Designs

**试试一些可替代的提示符设计**

With this list of special characters, we can change the prompt to see the effect. First, we’ll back up the existing string so we can restore it later. To do this, we will copy the existing string into another shell variable that we create ourselves:

参照这个特殊字符列表，我们可以更改提示符来看一下效果。首先， 我们把原来提示符字符串的内容备份一下，以备之后恢复原貌。为了完成备份， 我们把已有的字符串复制到另一个 shell 变量中，这个变量是我们自己创造的。

[me@linuxbox ~]$ ps1\_old="$PS1"

We create a new variable called ps1\_old and assign the value of PS1 to it. We can verify that the string has been copied with the echo command:

我们新创建了一个叫做 ps1\_old 的变量，并把变量 PS1的值赋 ps1\_old。通过 echo 命令可以证明 我们的确复制了 PS1的值。

[me@linuxbox ~]$ echo $ps1\_old

[\u@\h \W]\$

We can restore the original prompt at any time during our terminal session by simply reversing the process:

在终端会话中，我们能在任一时间复原提示符，只要简单地反向操作就可以了。

[me@linuxbox ~]$ PS1="$ps1\_old"

Now that we are ready to proceed, let’s see what happens if we have an empty prompt string:

现在，我们准备开始，让我们看看如果有一个空的字符串会发生什么：

[me@linuxbox ~]$ PS1=

If we assign nothing to the prompt string, we get nothing. No prompt string at all! The prompt is still there, but displays nothing, just as we asked it to. Since this is kind of disconcerting to look at, we’ll replace it with a minimal prompt:

如果我们没有给提示字符串赋值，那么我们什么也得不到。根本没有提示字符串！提示符仍然在那里， 但是什么也不显示，正如我们所要求的那样。我们将用一个最小的提示符来代替它：

PS1="\$ "

That’s better. At least now we can see what we are doing. Notice the trailing space within the double quotes. This provides the space between the dollar sign and the cursor when the prompt is displayed.

这样要好一些。至少能看到我们在做什么。注意双引号中末尾的空格。当提示符显示的时候， 这个空格把美元符号和光标分离开。

Let’s add a bell to our prompt:

在提示符中添加一个响铃：

$ PS1="\a\$ "

Now we should hear a beep each time the prompt is displayed. This could get annoying, but it might be useful if we needed notification when an especially long-running command has been executed.

现在每次提示符显示的时候，我们应该能听到嗡嗡声。这会变得很烦人，但是它可能会 很有用，特别是当一个需要运行很长时间的命令执行完后，我们要得到通知。

Next, let’s try to make an informative prompt with some host name and time-of-day information:

下一步，让我们试着创建一个信息丰富的提示符，包含主机名和当天时间的信息。

$ PS1="\A \h \$ "

17:33 linuxbox $

Try out the other sequences listed in the table above and see if you can come up with a brilliant new prompt.

试试其他上表中列出的转义序列，看看你能否想出精彩的新提示符。

Adding Color

**添加颜色**

Most terminal emulator programs respond to certain non-printing character sequences to control such things as character attributes (like color, bold text and the dreaded blinking text) and cursor position. We’ll cover cursor position in a little bit, but first we’ll look at color.

大多数终端仿真器程序支持一定的非打印字符序列来控制，比方说字符属性（像颜色，黑体和可怕的闪烁） 和光标位置。我们会更深入地讨论光标位置，但首先我们要看一下字体颜色。

Terminal Confusion

混乱的终端时代

Back in ancient times, when terminals were hooked to remote computers, there were many competing brands of terminals and they all worked differently. They had different keyboards and they all had different ways of interpreting control information. Unix and Unix-like systems have two rather complex subsystems to deal with the babel of terminal control (called termcap and terminfo). If you look in the deepest recesses of your terminal emulator settings you may find a setting for the type of terminal emulation.

回溯到终端连接到远端计算机的时代，有许多竞争的终端品牌，它们各自工作不同。 它们有着不同的键盘，以不同的方式来解释控制信息。Unix 和类 Unix 的系统有两个 相当复杂的子系统来处理终端控制领域的混乱局面（称为 termcap 和 terminfo）。如果你 查看一下终端仿真器最底层的属性设置，可能会找到一个关于终端仿真器类型的设置。

In an effort to make terminals speak some sort of common language, the American National Standards Institute (ANSI) developed a standard set of character sequences to control video terminals. Old time DOS users will remember the ANSI.SYS file that was used to enable interpretation of these codes.

为了努力使所有的终端都讲某种通用语言，美国国家标准委员会（ANSI）制定了 一套标准的字符序列集合来控制视频终端。原先 DOS 用户会记得 ANSI.SYS 文件， 这是一个用来使这些编码解释生效的文件。

Character color is controlled by sending the terminal emulator an ANSI escape code embedded in the stream of characters to be displayed. The control code does not “print out” on the display, rather it is interpreted by the terminal as an instruction. As we saw in the table above, the [ and ] sequences are used to encapsulate non-printing characters. An ANSI escape code begins with an octal 033 (the code generated by the escape key) followed by an optional character attribute followed by an instruction. For example, the code to set the text color to normal (attribute = 0), black text is:

字符颜色是由发送到终端仿真器的一个嵌入到了要显示的字符流中的 ANSI 转义编码来控制的。 这个控制编码不会“打印”到屏幕上，而是被终端解释为一个指令。正如我们在上表看到的字符序列， 这个 [ 和 ] 序列被用来封装这些非打印字符。一个 ANSI 转义编码以一个八进制033（这个编码是由 退出按键产生的）开头，其后跟着一个可选的字符属性，在之后是一个指令。例如，把文本颜色 设为正常（attribute = 0），黑色文本的编码如下：

\033[0;30m

Here is a table of available text colors. Notice that the colors are divided into two groups, differentiated by the application of the bold character attribute (1) which creates the appearance of “light” colors:

这里是一个可用的文本颜色列表。注意这些颜色被分为两组，由应用程序粗体字符属性（1） 分化开来，这个属性可以描绘出“浅”色文本。

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 14-2: Escape Sequences Used To Set Text Colors* | | | |
| Sequence | Text Color | Sequence | Text Color |
| \033[0;30m | Black | \033[1;30m | Dark Gray |
| \033[0;31m | Red | \033[1;31m | Light Red |
| \033[0;32m | Green | \033[1;32m | Light Green |
| \033[0;33m | Brown | \033[1;33m | Yellow |
| \033[0;34m | Blue | \033[1;34m | Light Blue |
| \033[0;35m | Purple | \033[1;35m | Light Purple |
| \033[0;36m | Cyan | \033[1;36m | Light Cyan |
| \033[0;37m | Light Gray | \033[1;37m | White |

|  |  |  |  |
| --- | --- | --- | --- |
| *表14-2: 用转义序列来设置文本颜色* | | | |
| 序列 | 文本颜色 | 序列 | 文本颜色 |
| \033[0;30m | 黑色 | \033[1;30m | 深灰色 |
| \033[0;31m | 红色 | \033[1;31m | 浅红色 |
| \033[0;32m | 绿色 | \033[1;32m | 浅绿色 |
| \033[0;33m | 棕色 | \033[1;33m | 黄色 |
| \033[0;34m | 蓝色 | \033[1;34m | 浅蓝色 |
| \033[0;35m | 粉红 | \033[1;35m | 浅粉色 |
| \033[0;36m | 青色 | \033[1;36m | 浅青色 |
| \033[0;37m | 浅灰色 | \033[1;37m | 白色 |

Let’s try to make a red prompt. We’ll insert the escape code at the beginning:

让我们试着制作一个红色提示符。我们将在开头加入转义编码：

<me@linuxbox ~>$ PS1='\[\033[0;31m\]<\u@\h \W>\$'

<me@linuxbox ~>$

That works, but notice that all the text that we type after the prompt is also red. To fix this, we will add another escape code to the end of the prompt that tells the terminal emulator to return to the previous color:

我们的提示符生效了，但是注意我们在提示符之后输入的文本也是红色的。为了修改这个问题， 我们将添加另一个转义编码到这个提示符的末尾来告诉终端仿真器恢复到原来的颜色。

<me@linuxbox ~>$ PS1='\[\033[0;31m\]<\u@\h \W>\$\[\033[0m\]'

<me@linuxbox ~>$

That’s better!

这看起来要好些！

It’s also possible to set the text background color using the codes listed below. The background colors do not support the bold attribute.

也有可能要设置文本的背景颜色，使用下面列出的转义编码。这个背景颜色不支持黑体属性。

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 14-3: Escape Sequences Used To Set Background Color* | | | |
| \033[0;40m | Blue | \033[1;44m | Black |
| \033[0;41m | Red | \033[1;45m | Purple |
| \033[0;42m | Green | \033[1;46m | Cyan |
| \033[0;43m | Brown | \033[1;47m | Light Gray |

|  |  |  |  |
| --- | --- | --- | --- |
| *表14-3: 用转义序列来设置背景颜色* | | | |
| \033[0;40m | 蓝色 | \033[1;44m | 黑色 |
| \033[0;41m | 红色 | \033[1;45m | 粉红 |
| \033[0;42m | 绿色 | \033[1;46m | 青色 |
| \033[0;43m | 棕色 | \033[1;47m | 浅灰色 |

We can create a prompt with a red background by applying a simple change to the first escape code:

我们可以创建一个带有红色背景的提示符，只是对第一个转义编码做个简单的修改。

<me@linuxbox ~>$ PS1='\[\033[0;41m\]<\u@\h \W>\$\[\033[0m\] '

<me@linuxbox ~>$

Try out the color codes and see what you can create!

试试这些颜色编码，看看你能定制出怎样的提示符！

Note: Besides the normal (0) and bold (1) character attributes, text may also be given underscore (4), blinking (5), and inverse (7) attributes as well. In the interests of good taste, many terminal emulators refuse to honor the blinking attribute, however.

注意：除了正常的 (0) 和黑体 (1) 字符属性之外，文本也可以具有下划线 (4)，闪烁 (5)， 和反向 (7) 属性。为了拥有好品味，然而，许多终端仿真器拒绝使用这个闪烁属性。

Moving The Cursor

**移动光标**

Escape codes can be used to position the cursor. This is commonly used to provide a clock or some other kind of information at a different location on the screen such as an upper corner each time the prompt is drawn. Here is a list of the escape codes that position the cursor:

转义编码也可以用来定位光标。这些编码被普遍地用来，每次当提示符出现的时候，会在屏幕的不同位置 比如说上面一个角落，显示一个时钟或者其它一些信息。这里是一系列用来定位光标的转义编码：

|  |  |
| --- | --- |
| *Table 14-4: Cursor Movement Escape Sequences* | |
| Escape Code | Action |
| \033[l;cH | Move the cursor to line l and column c. |
| \033[nA | Move the cursor up n lines. |
| \033[nB | Move the cursor down n lines. |
| \033[nC | Move the cursor forward n characters. |
| \033[nD | Move the cursor backward n characters. |
| \033[2J | Clear the screen and move the cursor to the upper left corner (line 0, column 0). |
| \033[K | Clear from the cursor position to the end of the current line. |
| \033[s | Store the current cursor position. |
| \033[u | Recall the stored cursor position. |

|  |  |
| --- | --- |
| *表14-4: 光标移动转义序列* | |
| 转义编码 | 行动 |
| \033[l;cH | 把光标移到第 l 行，第 c 列。 |
| \033[nA | 把光标向上移动 n 行。 |
| \033[nB | 把光标向下移动 n 行。 |
| \033[nC | 把光标向前移动 n 个字符。 |
| \033[nD | 把光标向后移动 n 个字符。 |
| \033[2J | 清空屏幕，把光标移到左上角（第零行，第零列）。 |
| \033[K | 清空从光标位置到当前行末的内容。 |
| \033[s | 存储当前光标位置。 |
| \033[u | 唤醒之前存储的光标位置。 |

Using the codes above, we’ll construct a prompt that draws a red bar at the top of the screen containing a clock (rendered in yellow text) each time the prompt is displayed. The code for the prompt is this formidable looking string:

使用上面的编码，我们将构建一个提示符，每次当这个提示符出现的时候，会在屏幕的上方画出一个 包含时钟（由黄色文本渲染）的红色长条。提示符的编码就是这个看起来令人敬畏的字符串：

PS1='\[\033[s\033[0;0H\033[0;41m\033[K\033[1;33m\t\033[0m\033[u\]

<\u@\h \W>\$ '

Let’s take a look at each part of the string to see what it does:

让我们分别看一下这个字符串的每一部分所表示的意思：

|  |  |
| --- | --- |
| Squence | Action |
| \[ | Begins a non-printing character sequence. The real purpose of this is to allow bash to correctly calculate the size of the visible prompt. Without this, command line editing features will improperly position the cursor. |
| \033[s | Store the cursor position. This is needed to return to the prompt location after the bar and clock have been drawn at the top of the screen. Be aware that some terminal emulators do not honor this code. |
| \033[0;0H | Move the cursor to the upper left corner, which is line zero, column zero. |
| \033[0;41m | Set the background color to red. |
| \033[K | Clear from the current cursor location (the top left corner) to the end of the line. Since the background color is now red, the line is cleared to that color creating our bar. Note that clearing to the end of the line does not change the cursor position, which remains at the upper left corner. |
| \033[1;33m | Set the text color to yellow. |
| \t | Display the current time. While this is a “printing” element, we still include it in the non-printing portion of the prompt, since we don't want bash to include the clock when calculating the true size of the displayed prompt. |
| \033[0m | Turn off color. This affects both the text and background. |
| \033[u | Restore the cursor position saved earlier. |
| \] | End non-printing characters sequence. |
| <\u@\h \W>\$ | Prompt string. |

|  |  |
| --- | --- |
| 序列 | 行动 |
| \[ | 开始一个非打印字符序列。其真正的目的是为了让 bash 能够正确地计算提示符的大小。如果没有这个转义字符的话，命令行编辑 功能会弄错光标的位置。 |
| \033[s | 存储光标位置。这个用来使光标能回到原来提示符的位置， 当长条和时钟显示到屏幕上方之后。当心一些 终端仿真器不推崇这个编码。 |
| \033[0;0H | 把光标移到屏幕左上角，也就是第零行，第零列的位置。 |
| \033[0;41m | 把背景设置为红色。 |
| \033[K | 清空从当前光标位置到行末的内容。因为现在 背景颜色是红色，则被清空行背景成为红色，以此来创建长条。注意虽然一直清空到行末， 但是不改变光标位置，它仍然在屏幕左上角。 |
| \033[1;33m | 把文本颜色设为黄色。 |
| \t | 显示当前时间。虽然这是一个可“打印”的元素，但我们仍把它包含在提示符的非打印部分， 因为我们不想 bash 在计算可见提示符的真正大小时包括这个时钟在内。 |
| \033[0m | 关闭颜色设置。这对文本和背景都起作用。 |
| \033[u | 恢复到之前保存过的光标位置处。 |
| \] | 结束非打印字符序列。 |
| <\u@\h \W>\$ | 提示符字符串。 |

Saving The Prompt

**保存提示符**

Obviously, we don’t want to be typing that monster all the time, so we’ll want to store our prompt someplace. We can make the prompt permanent by adding it to our .bashrc file. To do so, add these two lines to the file:

显然地，我们不想总是敲入那个怪物，所以我们将要把这个提示符存储在某个地方。通过把它 添加到我们的.bashrc 文件，可以使这个提示符永久存在。为了达到目的，把下面这两行添加到.bashrc 文件中。

PS1='\[\033[s\033[0;0H\033[0;41m\033[K\033[1;33m\t\033[0m\033[u\]<\u@\h \W>\$ '

export PS1

Summing Up

**总结归纳**

Believe it or not, there is much more that can be done with prompts involving shell functions and scripts that we haven’t covered here, but this is a good start. Not everyone will care enough to change the prompt, since the default prompt is usually satisfactory. But for those of us who like to tinker, the shell provides the opportunity for many hours of trivial fun.

不管你信不信，还有许多事情可以由提示符来完成，涉及到我们在这里没有论及的 shell 函数和脚本， 但这是一个好的开始。并不是每个人都会花心思来更改提示符，因为通常默认的提示符就很让人满意。 但是对于我们这些喜欢思考的人们来说，shell 却提供了许多制造琐碎乐趣的机会。