13 – Customizing the Prompt

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.

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.

Anatomy of a Prompt

Our default prompt looks something like this:

```
[me@linuxbox ~]$
```

Notice that it contains our username, our hostname, 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 1"). We can view the contents of `PS1` with the `echo` command.

```
[me@linuxbox ~]$ echo $PS1
[\u@\h \W]\$
```

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

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 7, "Seeing the World as the Shell Sees It." Table 13-1 provides a

partial list of the characters that the `bash` treats specially in the prompt string.

Table 13-1: 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`	Hostname of the local machine minus the trailing domain name.
`\H`	Full hostname.
`\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`	Username 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 during this shell session.
`\$`	This displays a “$” character unless we 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 that 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.

Trying Some Alternative Prompt Designs

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

```
[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 by using the `echo` command.

```
[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.

```
[me@linuxbox ~]$ PS1="$ps1_old"
```

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

```
[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 do. Since this is kind of disconcerting to look at, we'll replace it with a minimal prompt.

```
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.

Let's add a bell to our prompt.

```
$ PS1="\[\a\]\$ "
```

Now we should hear a beep each time the prompt is displayed, though some systems disable this "feature." This could get annoying, but it might be useful if we needed notification when an especially long-running command has been executed. Note that we included the `\[` and `\]` sequences. Since the ASCII bell (`\a`) does not "print," that is, it does not move the cursor, we need to tell `bash` so it can correctly determine the length of the prompt.

Next, let's try to make an informative prompt with some hostname and time-of-day information.

```
$ PS1="\A \h \$ "
17:33 linuxbox $
```

Adding time-of-day to our prompt will be useful if we need to keep track of when we perform certain tasks. Finally, we'll make a new prompt that is similar to our original.

```
17:37 linuxbox $ PS1="<\u@\h \W>\$ "
<me@linuxbox ~>$
```

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

Adding Color

Most terminal emulator programs respond to certain non-printing character sequences to control such things as character attributes (such as 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.

Terminal Confusion

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.

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.

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 `Esc` 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 as follows:

```
\033[0;30m
```

Table 13-2 lists the 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.

Table 13- 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

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

```
<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 will also display in 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.

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

That's better!

It's also possible to set the text background color using the codes listed Table 13-3. The background colors do not support the bold attribute.

Table 13-3: Escape Sequences Used to Set Background Color

Sequence	Background Color	Sequence	Background Color
\033[0;40m	Black	\033[0;44m	Blue
\033[0;41m	Red	\033[0;45m	Purple
\033[0;42m	Green	\033[0;46m	Cyan
\033[0;43m	Brown	\033[0;47m	Light gray

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

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

Try the color codes and see what you can create!

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

Moving the Cursor

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 in an upper corner each time the prompt is drawn. Table 13-4 lists the escape codes that position the cursor.

Table 13-4: Cursor Movement Escape Sequences

Escape Code	Action
`\033[`*l*`;`*c*`H`	Move the cursor to line *l* and column *c*
`\033[`*n*`A`	Move the cursor up *n* lines
`\033[`*n*`B`	Move the cursor down *n* lines
`\033[`*n*`C`	Move the cursor forward *n* characters
`\033[`*n*`D`	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

Using the codes in Table 13-4, 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:

```
PS1="\[\033[s\033[0;0H\033[0;41m\033[K\033[1;33m\t\033[0m\033[u\]
<\u@\h \W>\$ "
```

Table 13-5 outlines what each part of the string does.

Table 13-5: Breakdown of Complex Prompt String

Sequence	Action
`\[`	Begin a non-printing character sequence. The purpose of this is to allow `bash` to properly calculate the size of the visible prompt. Without an accurate calculation, command line editing features cannot position the cursor correctly.

`\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 recognize this code.*
`\033[0;0H`	Move the cursor to the upper-left corner, which is line 0, column 0.
`\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 in 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 the background.
`\033[u`	Restore the cursor position saved earlier.
`\]`	End the non-printing characters sequence.
`<\u@\h \W>\$`	Prompt string.

Saving the Prompt

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:

```
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

Believe it or not, there is much more that can be done with prompts involving shell func-

tions 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 casual fun.

Further Reading

- The *Bash Prompt HOWTO* from the Linux Documentation Project provides a pretty complete discussion of what the shell prompt can be made to do. It is available at:
  http://tldp.org/HOWTO/Bash-Prompt-HOWTO/
- Wikipedia has a good article on the ANSI Escape Codes:
  http://en.wikipedia.org/wiki/ANSI_escape_code