www.photocase.com

Bash tricks

SHELL POLISHING

A few basic tricks can liven up the command line and add a dash of color to your console. **BY HEIKE JURZIK**

The shell is the command line interpreter that interfaces between the user and the system. Among other things it interprets commands, wildcards and variables, links commands, and passes program output in to other tools or to files. Besides the Bourne Shell (Sh), Korn Shell (Ksh), C Shell (Csh), and Z Shell (Zsh), the Bourne Again Shell, or Bash for short, plays a prominent role on Unix-style systems – it has long established itself as the standard shell on Linux. Working with Bash is a lot more fun if you modify the prompt to suit your personal preferences, if you are familiar with the many keyboard shortcuts, and if you add more functionality by defining your own aliases and environmental variables.

Attention!

Commands are typically entered at the shell prompt. A typical prompt looks like this:

```
huhn@huhnix:~>
```

The prompt tells you your user ID, the name of the computer you are working on, and the current working directory (this is the home directory for the current user in our example, denoted by the tilde character). If you are working as the system administrator, root, the prompt will look slightly different – on most systems you can tell you are root by the pound sign in the prompt (#):

```
huhnix:~ #
```

Users can modify the prompt to display the date and time, the computer uptime, or to use different colors – a few steps are all it takes to set up your individual working environment. The environmental variable responsible for the appearance of the prompt is *PS1* (see the "Everything is Variable" box). If you would like to display the current time (in 24-hour format) in front of the user and computer names, followed by the current working directory, you can assign the variable temporarily by doing this:

```
$ export PS1="[\t]➥
\u@\h:\w> "
[19:11:06] huhn@huhnix:~>
```

The escape sequences used here are *\t* (time in 24-hour format), *\u* (username of the logged in user), *\h* (hostname up to the first dot) and *\w* (current working directory). Special escape sequences give you the ability to color the prompt, assigning green to a normal user prompt, and red to the administrator's prompt, for example.

The Bash manpage has a full list of escape sequences; and you might also like to check out the Bash Prompt Howto [1].

Good Memory

Bash has a good memory, storing user input in the history (a file called *~/.bash_history*), and letting users recycle commands on screen. Pressing the [Up arrow] key displays the last com-

Figure 1: Everything is variable – "env" outputs a complete list of Bash environmental variables.

mand to be typed on your screen. Pressing the key multiple times takes you farther back in time, and pressing the [Down arrow] key takes you closer to the present day. You can modify commands that you recycle in this way, and run them again by pressing Enter.

Of course, Bash's memory is only as good as you tell it to be in the *HISTSIZE* environmental variable (see the "Everything is Variable" box). You can ask the shell to tell you how many slots it has reserved for commands:

```
$ echo $HISTSIZE
500
```

When the shell hits this threshold, the commands at the top of the list start to disappear to make way for new ones. To avoid wearing out your [Up arrow] key, Bash includes a search function. If you press the keyboard shortcut [Ctrl-R] or [Ctrl-S] (see the "Bash Keyboard Shortcuts" box), you can search forwards or backwards through the history for a command. The prompt changes to reflect this, for example, pressing [Ctrl-R] puts the following prompt on your screen:

```
(reverse-i-search)`':
```

GLOSSARY

Builtin: Builtins are not independent executables, but integrated shell commands. Builtins do not have their own manpage, but are typically described in the Bash documentation (man bash).

Type the command (or part of the command) you are looking for at the colon. Bash will complete commands while you are typing so that just a few letters might locate the command you are looking for. To run a command you have found using this approach, just press Enter. Alternatively, keep pressing [Ctrl] + [R] until you find what you are looking for. Pressing [Esc] lets you modify the command before running it.

Context

The functions we have looked at thus far help you find commands, but taken completely out of context. To display more complex processes on screen, you can either view the history file (*~/.bash_history*), or run the shell **builtin** fc ("Fix Command"). In combination with the -l flag, this tool outputs the last 17 commands by default. You can add a number to change the default. For example, *fc -l -5* outputs the last five commands.

The *script* tool provides an alternative approach to logging a shell session. You could type *script MyLogfile* before launching into an extended shell hack, to record any commands you enter and the output they create. You can quit the practical logger by pressing [Ctrl] + [D].

Bash Configuration Files

Bash comes with a collection of start files that modify the behavior of the shell (and other programs). To add versatility to this chaos, most distributions do their own thing here, and assign different priorities to configuration files, or include yet more files.

Using SSH to log in at a virtual console launches a login shell, which is prefaced by a minus sign in the process list:

```
$ ps auxwww | grep bash
huhnix    3778  0.0  0.3  3056➥
1652 tty1     Ss+  May28   0:00➥
-bash
huhnix    4193  0.0  0.3  3068➥
1672 pts/1    Ss   May28   0:00➥
/bin/bash
```

The login shell first checks the user's home directory for a *~/.bash_profile*. If this file is missing, bash looks for *~/.bash_login*, and then for */etc/profile* or *~/.profile*.

Interactive Shells

In addition to this, there are interactive shells – such as the ones launched from within other shells. These include Xterms or other console programs on graphical user interfaces. Preferences for these shells are stored in the private *~/.bashrc*, or in the system global */etc/bash.bashrc* file, where you also add the alias definitions and environmental variables discussed in this article.

Whenever you modify your own *~/.bashrc*, you either need to launch a new shell, or enter *source ~/.bashrc* to parse the changes. You can shorten the command by entering *~/.bashrc*.

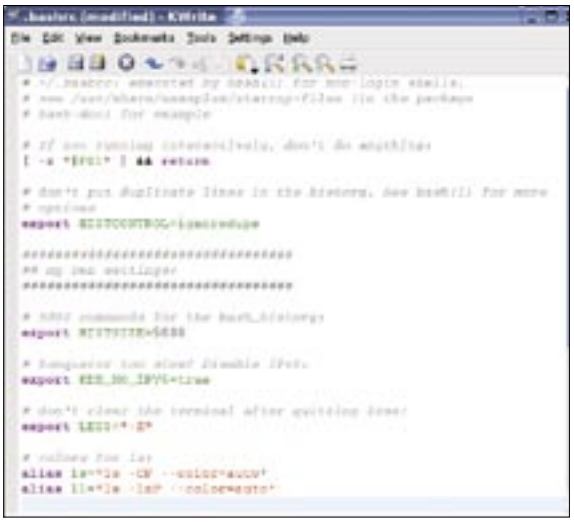

Figure 2: Environmental variables give users the ability to modify the behavior of Bash and many other programs.

The alias builtin is another practical feature that lets users define commands they use regularly, along with their typical choice of options, and assign a shorter and more intuitive name to the whole kitten caboodle. Most distributions define a few default aliases – to find out which aliases your version of Bash knows, type the alias command at the prompt:

```
$ alias
alias ls='ls ➥
--color=auto'
```

To define an alias for the current shell session, just type the *alias* command, followed by the name for the new shortcut, an equals sign, and the command (in quotes), as in:

```
alias ll='ls -l'
```

The *unalias* command lets you drop any shortcuts you no longer need. For example, *unalias ll* would ditch the alias we just defined. To make an alias persistent, you need to add the definition to your *~/.bashrc*, and parse the file.

Table 1: Bash Keyboard Shortcuts

[Ctrl-A]	Go back to the start of the input.
[Ctrl-E]	Jump to the end of the input.
[Esc-B]	Go back one word.
[Esc-F]	Go forward one word.
[Ctrl-B]	Go back one character.
[Ctrl-F]	Go forward one character.
[Ctrl-K]	Delete from the cursor position to the end of the input.
[Ctrl-U]	Delete from the cursor position to the start of the line.
[Ctrl-W]	Delete the word to the left of the cursor.
[Ctrl-T]	Swap the two characters to the left and below the cursor.
[Esc-T]	Swap the two previous words.
[Ctrl-L]	Clear the content of the terminal window.

Everything is Variable

Environmental variables let you extend the shell's feature scope, and modify the behavior of certain other programs. The system comes with a set of default variables, such as the *HISTSIZE* example (in the "Good Memory" section). You can use the *echo* command to output the value of the variable, remembering to preface the variable with a dollar sign:

```
echo $HISTSIZE
```

If you are interested in a complete list of all Bash environmental variables, you can type *env* at the command line to output the list (Figure 1).

You can run the *export* command to set a variable. For example, if you want to run a program in a different language just this once, temporarily set the *LC_ALL* variable, and add the executable name after a blank. The

```
LC_ALL=fr_FR firefox
```

command would launch the popular browser in French for a change. You can also use the *unset* command to disable a variable set for the current shell, as in:

```
unset LS_COLORS
```

To set a variable persistently, add the *export* command to *~/.bashrc*. Figure 2 shows a commented Bash setup file with various examples of useful variables. ■

INFO

[1] Bash Prompt HOWTO: *http://www.tldp.org/HOWTO/Bash-Prompt-HOWTO/*