
System Administration Toolkit: Get the most out of bash

Skill Level: Intermediate

[Martin Brown \(mc@mcslp.com\)](mailto:mc@mcslp.com)
Freelance Writer
Consultant

12 Dec 2006

Ease your system administration tasks by taking advantage of key parts of the Bourne-again shell (bash) and its features. Bash is a popular alternative to the original Bourne and Korn shells. It provides an impressive range of additional functionality that includes improvements to the scripting environment, extensive aliasing techniques, and improved methods for automatically completing different commands, files, and paths.

About this series

The typical UNIX® administrator has a key range of utilities, tricks, and systems he or she uses regularly to aid in the process of administration. There are key utilities, command-line chains, and scripts that are used to simplify different processes. Some of these tools come with the operating system, but a majority of the tricks come through years of experience and a desire to ease the system administrator's life. The focus of this series is on getting the most from the available tools across a range of different UNIX environments, including methods of simplifying administration in a heterogeneous environment.

Bash background

Shells under UNIX and Linux® typically fall into one of two categories based on the original shells included with the earliest versions of UNIX. The two types are the Bourne shell and the C shell, the latter being distinctive because its format and structure is like that of the C programming language.

The Bourne shell is easier to use and understand than the C shell, but it is less practical for the more complex script programming that you might want to achieve within the shell programming environment. The Korn shell provides ease of use of the Bourne shell and added extensions for job control (allowing you to easily manage multiple background jobs), command-line editing and history, and added elements of the C shell to make programming easier.

The Bourne-again shell (bash) is an open source project that combines the principles of the Bourne shell, programming environment of the C shell, extended functionality of the Korn shell, and a number of extensions of its own to provide a rich environment not only for programming shell scripts, but also as an interactive shell environment for controlling and interacting with your machine.

Command-line editing and key bindings

The main command prompt within bash provides both the ability to edit the command line and a history function, remembering individual command lines so that you can execute them again.

The editing functionality means that you can go forward and backward through the command line currently displayed to make changes and correct typos. In bash's standard form, you can use the cursor keys for basic movement. More extensive commands, such as going backward and forward by words, are controlled by the Readline library that supports both vi and emacs bindings by default. To set the editing mode, specify your preferred mode either on the command line or in a bootstrap file: `$ set editing-mode emacs`.

For example, using the emacs editing mode, the following key bindings are in effect:

- Control-A -- This key binding takes you to the beginning of the line.
- Control-E -- This key binding takes you to the end of the line.
- Control-K -- This key binding deletes everything to the end of the line.
- Meta-B -- This key binding goes back by one word.
- Meta-F -- This key binding goes forward by one word.
- Meta-D -- This key binding deletes the current word.

You can in fact bind any key or combination you like to a particular operation using the internal `bind` bash command. To start, you can get a list of the available commands by using the `-P` option (see [Listing 1](#)).

Listing 1. Using the -P option to get a list of available commands

```
$ bind -P
abort can be found on "\C-g", "\C-x\C-g", "\M-\C-g".
accept-line can be found on "\C-j", "\C-m".
alias-expand-line is not bound to any keys
arrow-key-prefix is not bound to any keys
backward-byte is not bound to any keys
...
yank can be found on "\C-y".
yank-last-arg can be found on "\M-.", "\M-_".
yank-nth-arg can be found on "\M-\C-y".
yank-pop can be found on "\M-y".
```

The `\C` refers to the control key. The `\M` sequence refers to the 'meta' key (special on some keyboards, or usually the **Alt** key or the **Escape** key).

To set a binding, you must specify the key sequence and the command to be executed, separated by a colon, with the key sequence escaped by a double quote (in extreme circumstances, you might need to escape this again with a single quote). For example, to change **Control-B** to go backwards word by word, use `$ bind "\C-b":backward-word`.

You can even use the binding to execute a shell command (for example, to run an application). To do this, you must add the `-x` option, and this is an example of where the escaping of both is required. For example, to set **Control-E** to run emacs, you would use the following: `$ bind -x '"\C-e":emacs`.

To have key bindings in bash enabled every time, you can either set the information in the `.inputrc` file (which then affects all Readline-enabled applications), or you can place specific bash bindings in your startup scripts, which will be covered later in this article.

Command history

Every command that you enter is recorded so that you go back to the command and either run it again verbatim, or edit it and run the edited version. You can go backwards and forwards through your command history (in reverse chronological order) using **Control-P** (previous command) and **Control-N** (next). You can only go next if you have already gone back through your previous commands.

If you know the contents of the command but can't remember where it was and don't want to go through the command list by hand, you can use **Control-R** to do a reverse intelligent search. This goes back to the first command (in reverse order) that matches the first letter you typed; letters typed successively match more and

more specifically. Once you've found the command you want, press **Return** to run it verbatim. To edit the command you've found, use the cursor keys (or key bindings) to move to the point you want to edit.

Prompt customization

All shells enable you to customize the prompt presented to you on the command line, usually by setting the value of the `PS1` variable. Normally this is limited to being able to set a static string or, in case of the Korn shell, you can normally set a dynamic value, such as the current directory.

Bash extends this functionality even further to allow the inclusion of username, hostname, and even hidden characters (for example, the escapes required for colorization or setting the title of windows and Xterms). The string specified is evaluated each time just before the prompt is printed so that it's always up to date.

Custom expansions of specific dynamic values are achieved with a series of backslash characters. For example, to set a typical prompt consisting of the username and last part of the current directory, use the following: `PS1="\u \W$"`.

A more typical solution is to display the username, hostname, directory, and the dollar or hash sign, depending on the UID of the current user. The latter option displays a hash sign when the effective user ID is zero (root): `PS1='\u@\h:\W \ $'`.

The full list of the available expansions is shown in [Table 1](#).

Table 1. List of available expansions

Escape character	Description
<code>\t</code>	<code>\t</code> is the time, in HH:MM:SS format.
<code>\d</code>	<code>\d</code> is the date, in Dayname Month Day format (for example, Fri Oct 13).
<code>\n</code>	<code>\n</code> is a new line.
<code>\s</code>	<code>\s</code> is the name of the shell (typically bash, or '-bash' if the shell is a login shell).
<code>\w</code>	<code>\w</code> is the full path of the current directory.
<code>\W</code>	<code>\W</code> is the final part of the current directory (in other words, 'mc' when in '/home/mc').
<code>\u</code>	<code>\u</code> is your username.
<code>\h</code>	<code>\h</code> is the hostname.
<code>\#</code>	<code>\#</code> is the command number of this command (the

	value is incremented for each line).
\!	\! is the unique history number for this command.
\nnn	\nnn is the character that is identified by the octal nnn.
\\$	\\$ is the # character if the effective UID is zero, otherwise it is the \$ character.
\\	\\ is a backslash.
\[\[begins an escape sequence; generally used for terminal control characters for colorization and titles.
\]	\] ends an escape sequence.

The escape sequence characters can be used to send terminal control sequences, which in turn can be used to set colors and terminal titles. Because bash updates this information each time the prompt is shown, this automatically updates window titles accordingly. For example, use the following sequence in [Listing 2](#) to set the terminal window and Xterm title to user@hostname:fullpath, and set the prompt to hostname:endpath with an appropriate termination character (dollar or hash).

Listing 2. Setting the terminal title and the prompt

```
PS1=" \[\033]0;\u@\h: \w\007\]\[\033[31m\]\h \[\033[34m\]\w \$ \[\033[00m\]"
```

In addition, the hostname is colored in red when it is root, green when it is a normal user, and the remainder of the prompt is colored in cyan.

File and directory completion

To help speed up your shell interaction, bash includes pathname completion -- that is, you can get bash to either fill in the remaining parts of a pathname or provide you with a list of potential expansions.

For example, if you look at the current directory, there are a range of different subdirectories (see [Listing 3](#)).

Listing 3. Subdirectories in the current directory

```
$ ls -f
back/   cheffyhack/  edin/      logstomerge/  mysql-binlogs/  svn/
build/  cvs/         install/   lost+found/   mysqlsizer      vmware/
calc/   dbdumps/    logs/      my.cnf        statmon/        webs/
```

To change into the `edin` directory, you could type: `$ cd edin`. Or, using completion within `bash`, you could type: `$ cd e` and then press the **TAB** key. By default, this will attempt to expand the pathname beginning with 'e'; the result in this directory should be: `$ cd edin`.

If there is more than one match, then the first press of **TAB** fails; pressing **TAB** a second time returns a list of matching paths (see [Listing 4](#)).

Listing 4. Returning a list of matching paths

```
$ cd my
my.cnf          mysql-binlogs/ mysqlsizer
```

The expansion continues to work, replacing what it can with unique components, either until there are either no further expansions, or only one. For example, consider the following sequence shown in [Listing 5](#).

Listing 5. Continuing the expansion

```
$ cd my <TAB>
my.cnf          mysql-binlogs/ mysqlsizer
$ cd mys <TAB>
$ cd mysql <TAB> <TAB>
mysql-binlogs/ mysqlsizer
$ cd mysqls <TAB>
$ cd mysqlsizer
```

If the pathname is a directory, then you can continue to expand each part of the pathname.

Aliases

Aliases are a simple mechanism that enable you to provide an expansion for a given sequence. Because the alias is an expansion and not a replacement, it enables you to continue adding options and arguments to an aliased command.

For example, it's common to set an alias (`ll`), which expands to `ls -l`, for a long listing of a directory or list of files. You can create the alias using the following command line: `alias ll='ls -l'`.

To use, just type the alias at the prompt: `$ ll`.

Because the alias is just that, an alias to the real command, you can add further options, for example, to list all the files in the directory, use: `$ ll -a`, which effectively expand to: `$ ls -l -a`.

Aliases are an effective way to run commands quickly, or run your favorite commands with the command-line options you commonly use.

Using the directory stack

The directory stack enables you to store one or more directories into a temporary area, and then bring them back again. The list of directories acts like a stack -- you push a directory onto the top of the stack, and pop a directory off the top again to get it back.

To push a directory onto the stack, use `pushd` and the directory you want to place on the stack -- for example, you can push the current directory, which can be identified by the single period (see [Listing 6](#)).

Listing 6. Pushing the current directory

```
$ pushd .  
/export/data /export/data
```

Bash responds with the directory you pushed and a list of all the directories currently in the stack separated by a space. You can see this more clearly by changing the directory and pushing it again (see [Listing 7](#)).

Listing 7. Pushing a different directory

```
$ cd /etc  
$ pushd .  
/etc /etc /export/data
```

Using `popd` takes the last directory added to the stack off and changes to that directory (see [Listing 8](#)).

Listing 8. Using popd

```
$ cd /usr/local/bin  
$ popd  
/etc /export/data  
$ pwd
```

The `popd` command returns the list of directories before changing to the new directory.

The directory functionality is most often used when you want to temporarily change your directory and then go back to the original directory. To make this kind of situation easier, `bash` is often configured with two aliases, `pu` and `po`, as shown in [Listing 9](#).

Listing 9. Pu and po aliases

```
$ alias pu='pushd .'
```

```
$ alias po='popd'
```

For example, imagine you are working in a directory and suddenly realize you need to create a tar archive in another directory (see [Listing 10](#)).

Listing 10. Use pu and po to create a tar archive

```
$ pwd
/usr/local/bin
$ pu
/usr/local/bin /usr/local/bin
$ cd /export/data
$ tar cf webs.tar ./webs
$ po
$ pwd
/usr/local/bin
```

Although the directory stack system supports multiple directories, it frequently gets used for only one or two while you temporarily pop somewhere else.

Bash run time configuration files

When logging in, `bash` supports the standard `.profile` file (as used by Bourne shell), in addition to its own specific `.bash_profile`. The file to use is chosen, as follows:

- If `~/.bash_profile` exists, use it, otherwise:
- If `~/.bash_login` exists, use it, otherwise:
- If `~/.profile` exists, use it.

For `bash` shells not started as a login shell (for example, when using Xterm or another application that starts a sub shell, `bash` looks for the `~/.bashrc` file, if it exists.

Since the contents of `.bashrc` and `.bash_profile` are unlikely to change, it is not uncommon to find the `.bash_profile` containing the following in [Listing 11](#).

Listing 11. `.bash_profile` contents

```
if [ -f ~/.bashrc ];
then
    source ~/.bashrc
fi
```

Beyond these rules, the contents of the files is entirely up to you. Because so much customization is available within the bash environment, it is also not uncommon to find that `.bashrc/.bash_profile` contents are simply wrappers that source a number of other `~/.bash_*` scripts. For example, you might split up your configuration into:

- `~/.bash_aliases` -- to store all your custom aliases and functions
- `~/.bash_path` -- to store all your path specifications
- `~/.bash_vars` -- to store all your bash variables

These are then sourced within `.bashrc` (see [Listing 12](#)).

Listing 12. `.bashrc` contents

```
if [ -f ~/.bash_path ]; then
    source ~/.bash_path
fi

if [ -f ~/.bash_vars ]; then
    source ~/.bash_vars
fi

if [ -f ~/.bash_aliases ]; then
    source ~/.bash_aliases
fi
```

You should be careful to ensure that the files are sourced in the right order.

Scripting improvements

The last area in which bash provides some considerable improvement is in the scripting functionality. One of the commonly quoted limitations of most shells is that they have loose variable typing, no support for arrays of data, and no built-in functionality to perform basic math or expressions. All of these have been resolved within bash to different degrees.

Variables within bash can be declared before they are used, and the declaration can include a type. For example, to declare a variable as an integer type (and therefore to always be identified as a valid number), use: `$ declare -i myint`. To set a value at the same time, use: `$ declare -i myint=235`.

To perform basic arithmetic, you can embed the expression into `$(())` (see [Listing 13](#)).

Listing 13. Embedding the expression to perform basic arithmetic

```
$ echo $((4+3*12))
40
```

You can also include variables (see [Listing 14](#)).

Listing 14. Including variables

```
$ echo $((myint+3*12))
63
```

To declare a variable as an array type, use `$ declare -a myarray`.

You can add values by specifying a parenthesized list of values: `$ declare -a myarray=(tom dick harry)`.

To get a value out of the array, specify the array reference (see [Listing 15](#)).

Listing 15. Specifying the array reference

```
$ echo ${myarray[1]}
dick
```

You can use the same system to populate the array, for example, from a list of files (see [Listing 16](#)).

Listing 16. Populating the array from a list of files

```
$ declare -a files=`ls`
$ echo $files
back/ build/ calc/ cheffyhack/ cvs/ dbdumps/ edin/ install/ \
logs/ logstomerge/ lost+found/ my.
cnf mysql-binlogs/ mysqlsizer statmon/ svn/ vmware/ webs/
```

And you can use the entire contents of the array by using the `@` symbol as the

element specification (see [Listing 17](#)).

Listing 17. Using the @ symbol as the element specification

```
for file in ${files[@]}
do
    echo $file
done
```

This manipulation and variable support makes many aspects of programming with a shell script significantly easier.

Summary

Bash provides a number of significant improvements over the traditional Bourne, Korn, and C shells. The bulk of these extensions improve the experience for the interactive user -- that is, the user who is using bash as their main method of interacting with their system. Some systems become a habit, for example, pathname completion and aliases, while others are vital on individual occasions. The directory stack functionality is a good example; it becomes invaluable when you want a quick method of visiting a directory without losing your train of thought.

Whatever elements you use within the bash shell, you will find a rich environment and an almost limitless array of customizations that you can use to improve your interaction and environment.

Share this...

[Digg
this
story](#)

[Post
to
del.icio.us](#)

[Slashdot
it!](#)

Resources

Learn

- [System Administration Toolkit](#): Check out other parts in this series.
- ["Making UNIX and Linux work together"](#) (developerWorks, April 2006): This article is a guide to getting traditional UNIX distributions and Linux working together.
- [Solaris to Linux Migration: A Guide for System Administrators](#): Different systems use different tools, and this IBM Redbook helps you identify some key tools.
- ["Exploring the Linux memory model"](#) (developerWorks, January 2006): Understand how Linux uses memory, swap space, and exchanges pages and processes between the two.
- ["Working in the bash shell"](#) (developerWorks, May 2006): This tutorial provides a guide for using the Bourne Again Shell for all your work, including methods for customizing and extending your environment.
- [Bash](#): Bash is an alternative shell to the standard Bourne shell with similar syntax, but an expanded range of features, including aliasing, job control and auto-completion of file and directory names.
- [AIX® and UNIX articles](#): Check out other articles written by Martin Brown.
- Search the AIX and UNIX library by topic:
 - [System administration](#)
 - [Application development](#)
 - [Performance](#)
 - [Porting](#)
 - [Security](#)
 - [Tips](#)
 - [Tools and utilities](#)
 - [Java technology](#)
 - [Linux](#)
 - [Open source](#)
- [AIX and UNIX](#): The AIX and UNIX developerWorks zone provides a wealth of

information relating to all aspects of AIX systems administration and expanding your UNIX skills.

- [New to AIX and UNIX](#): Visit the New to AIX and UNIX page to learn more about AIX and UNIX.
- [AIX 5L™ Wiki](#): A collaborative environment for technical information related to AIX.
- [Safari bookstore](#): Visit this e-reference library to find specific technical resources.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Get products and technologies

- [IBM trial software](#): Build your next development project with software for download directly from developerWorks.

Discuss

- Participate in the [developerWorks blogs](#) and get involved in the developerWorks community.
- Participate in the AIX and UNIX forums:
 - [AIX 5L -- technical forum](#)
 - [AIX for Developers Forum](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant](#)
 - [Performance Tools -- technical](#)
 - [Virtualization -- technical](#)
 - [More AIX and UNIX forums](#)

About the author

Martin Brown

Martin Brown has been a professional writer for more than seven years. He is the author of numerous books and articles across a range of topics. His expertise spans myriad development languages and platforms -- Perl, Python, Java™, JavaScript,

Basic, Pascal, Modula-2, C, C++, Rebol, Gawk, Shellsript, Windows®, Solaris, Linux, BeOS, Mac OS X and more -- as well as Web programming, systems management, and integration. He is a Subject Matter Expert (SME) for Microsoft® and regular contributor to ServerWatch.com, LinuxToday.com, and IBM developerWorks. He is also a regular blogger at Computerworld, The Apple Blog, and other sites. You can contact him through [his Web site](#).

Trademarks

IBM, AIX, and AIX 5L are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.