Working at the command line

AT YOUR COMMAND

Beyond all the splash screens, screen savers, and vivid rock-star wallpaper is the simple yet powerful Bash shell. **BY BRUCE BYFIELD**

Many desktop users approach the command line as though armed with a magic spell. They have a command – complete with options – to type or paste to get the desired results, but they are unclear what else might be going on. This approach is understandable; however, if you take the time to understand something of the structure of the command line, you can increase your control over your computing.

By default, most distributions run Bash (the Bourne Again Shell). Bash is a command-line interpreter – a program that runs macros and other utilities. These macros and utilities are the commands that you enter at the prompt. They include those built into Bash, such as *cd*, and many others that are external, including most of the commands that you run. However, from the end user's perspective, the difference between internal and external commands is unimportant.

Like other shells, Bash can run interactively or non-interactively. When acting as a login shell for your account, Bash runs non-interactively, reading instructions from the *.bashprofile* file in your home directory. In many cases, commands give you the option to create a file and run it non-interactively.

Most of the time, though, Bash runs as an interactive shell, meaning that you can enter commands and scripts using the keyboard, and Bash processes your input and displays output. You can also fine-tune how Bash runs with a set of options similar to any commands. These can be entered in a KDE Konsole profile or in a script that you run when opening a command line.

One of Bash's most common options is *-r*, which places Bash in restrictive mode. In restrictive mode, some actions, like using the *cd* command or changing environment variables, are disabled. Some administrators place Bash in restrictive mode in the hope of limiting the damage that rash users can cause on a network, but, more often, restricted shells are used to sandbox – that is, run a command in isolation for test purposes. The option *--debugger* is also used to log debugging information.

Getting Around at the Prompt

In the old days, the command prompt was the primary means of interacting with Linux, but most contemporary Linux systems open up in some form of graphical user interface. To reach the command prompt on a GUI-based Linux system, you'll need to open a terminal window. Systems that use the Gnome desktop environment typically include the Gnome Terminal application. On Ubuntu, for instance, you'll find the Terminal app in *Applications* | *Accessories* (Figure 2). KDE-based openSUSE systems, on the other hand, include the Konsole terminal program, which you will find in the **System** menu. Several other terminal programs are also available for Linux systems. Consult your vendor documentation for more on finding your way to a command prompt.

While in the shell, forget about your mouse, although you can copy and paste, as the Edit menu reveals. Communicate with your system through the keyboard; type a line, then press Enter. Of course, modern tools like Konsole or the

Tweak Your Bash

You can modify how Bash operates with its built-in commands. For instance, the *umask* command changes the default permissions used when creating a file, whereas the *alias* command can be used to change the name used to run a specific command – for example, my Debian system comes with *ls --color=auto* aliased to *ls*, so that directories and different file types are all colored.

Another way to modify Bash is through the *shopt* built-in (Figure 1). The *shopt* command includes a number of interesting, if seldom used, possibilities. For example, *shopt -s cdspell* enables Bash to correct minor misspellings in its default directories when you use the *cd* command. Similarly, *shopt -s checkjobs* lists any stopped jobs that remain when you close the shell.

These few examples of what you can do with Bash should be sufficient to show that Bash is far from the passive recipient of your commands. Instead, like the commands that it runs, Bash is full of options and can be customized to suit your needs. You'll learn more about customizing the Bash environment later in this issue.

Gnome Terminal are not terminals in the old sense but are actually *terminal emulators*. You can close or minimize the terminal window as you would any other window on your Linux system.

This handbook assumes you have some basic knowledge of how to move around in the Bash shell. If you are looking for a very basic crash course, a few simple commands will help you get familiar with the command prompt.

Most likely, the terminal will open in your home directory. Type *ls* to list the contents of the directory.

You can use the *cd* ("change directory") command to move to another directory. You'll also need to mention the path to the target directory:

```
$ cd /home/berney/Music
```

Most Bash shells let you use a period (.) in the path to represent the current directory. In other words, a user named *berney* could move from his home directory to the *Music* subdirectory by typing:

```
$ cd ./Music
```

A double period means "go up one level in the directory path," so if *berney* wanted to go from the */home/berney/Music* directory back to his home directory (*/home/berney*), he could type:

```
$ cd ..
```

Many systems also use the tilde character (~) to represent the home directory, so no matter where you are, you can always return to your home directory with:

```
$ cd ~
```

If you start to get lost when you are navigating around in the directory structure, you can always enter the *pwd* command ("Print Working Directory") to display the name of the current directory.

To create a new directory, enter the *mkdir* command and give the name of the new directory:

```
$ mkdir /home/berney/Music/Beatles
```

Or, if *berney* were already in his *Music* directory, he could just type:

```
$ mkdir ./Beatles
```

The *cp* command lets you copy files. The syntax is as follows:

```
cp source_filename destination_filename
```

The default is to look in the current directory, however, you can include a path with the source or destination to copy from or to a different directory. Of course, you must have the necessary permissions to access the directory.

To delete a file, use the *rm* command,

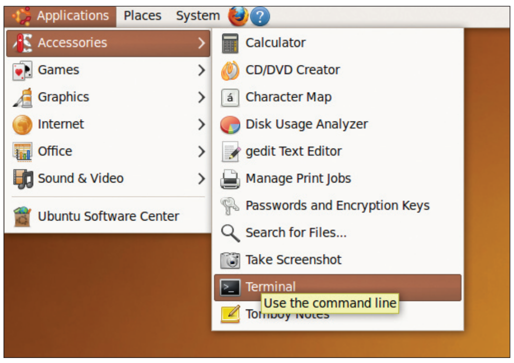
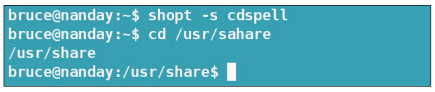


Figure 2: Starting up a terminal window in Ubuntu.



```
bruce@nanday:~$ shopt -s cdspell
bruce@nanday:~$ cd /usr/sahare
/usr/share
bruce@nanday:/usr/share$ 
```

Figure 1: Shopt is a command built into Bash that provides many interesting features. Here, the cdspell option automatically corrects errors when you type directory names.

and to delete a directory, the *rm -r* or *rmdir* command. (Needless to say, be careful how you use these commands.)

A summary of these basic commands appears in Table 1. Each of these commands includes additional options that you can enter at the command line. As you will learn later in this article, you can type *man* or *info*, followed by the command, for information on syntax and usage. For instance, to learn the various options for the *mkdir* command, you would enter:

```
man mkdir
```

In later articles, you will learn about more Bash commands for modifying text, managing users, overseeing processes, and troubleshooting networks.

History

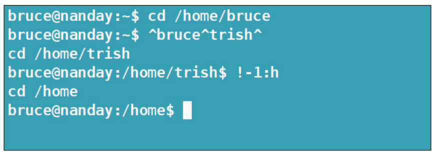
If you are doing repetitive commands in Bash, you can save time by using the history for the current account. Stored in the *bash_history* file in your home directory is a list of commands that you have run, numbered with *1* as the oldest. You can use the arrow keys to move up and down or use the plain command *history* to see a complete list of what is stored in your history.

If you are somewhat more adventurous, you can use a number of shortcuts to run a previous command in the history. *!number* runs the command with that number. Similarly, *!-number* sets the number of previous commands to revert to, and *!string* runs the first command that includes that string.

When you are either very certain of what you are doing or willing to live dangerously, you can enter *^string1^ string2^* to repeat the last command but replace the first string of characters with the second. Another trick is to add *:h* to remove the last element of the path in the command or *:t* to remove the first element. However, if you are uncertain of the results, you can add *:p* to print the

Table 1: Some Basic Bash Commands

ls	Lists contents of the current directory
cd	Change directory
pwd	Show current working directory
mkdir	Make directory
cp	Copy file(s)
rm	Remove file(s)
rmdir	Remove directory



```
bruce@nanday:~$ cd /home/bruce
bruce@nanday:~$ ^bruce^trish^
cd /home/trish
bruce@nanday:/home/trish$ !-1:h
cd /home
bruce@nanday:/home$ █
```

Figure 3: You can use several keyboard shortcuts to run commands in the history with slight changes. Here, the string "bruce" is replaced with "trish" in the first case, then only the head of the path is preserved in the second.

Table 2: Man Page Sections

Section	Description
1	General commands
2	System calls
3	C library functions
4	Special files (usually devices found in */dev*) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellanea
8	System administration commands and daemons

command that you find but not run it (Figure 3).

Documentation

Bash and the individual commands associated with it add up to a lot to learn. Fortunately, you don't have to remember everything. Like other Unix-type systems, GNU/Linux includes a number of different help systems.

The most basic form of help is the man page (Figure 4). Man pages are divided into eight sections (see Table 2), but most of the time, you only need to type the command *man* followed by the command, file, or concept about which you want information.

However, some topics have entries in several sections. To go to the specific section, place the number of the section between the *man* command and the topic. Thus, *man man* takes you to the basic page about the *man* command in section 1, but *man 7 man* takes you to a section about the collection of macros used to create man pages. Either way, when you are finished reading, press Ctrl+Z followed by Ctrl+C to return to the command line.

When you are doing deeper research, consider using *apropos* followed by a topic to receive a list of all the applicable man pages. The one drawback to *apropos* is that, unless you are very specific, you could get dozens of pages, only a few of which are relevant to you.

By contrast, if all you need is a brief snippet of information, use *whatis* followed by the command. For example, if you enter *whatis fdisk*, you receive the line *fdisk (8) -- Partition table manipulator for Linux*. The (8) refers to the man section where detailed information is available. Similarly, if you need to identify a file type, use *type* then the file.

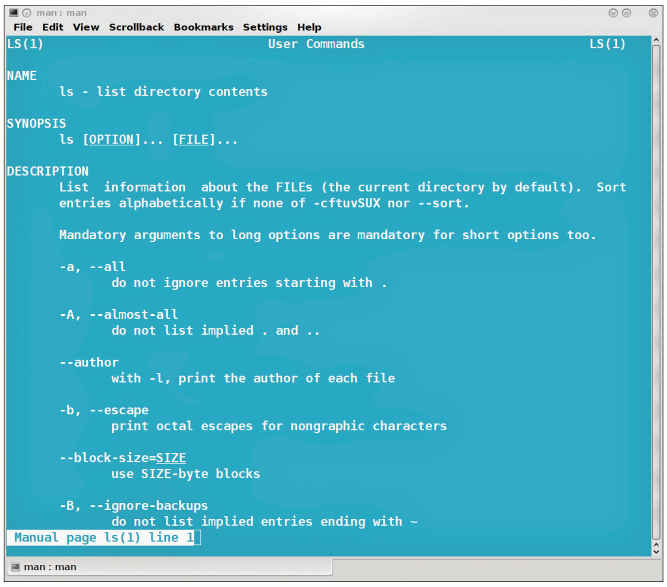
For several decades, man pages have been the standard help form. However, more than a decade ago, the GNU Project made *info* its official help format. But, instead of replacing *man*, *info* has simply become an alternative (Figure 5). Although some man pages today stress that the full help file is only available through *info*, in practice, many developers simply maintain both *info* and *man*, focusing on the command structure in the man pages and on basic instruction in the info pages. Still, it can never hurt to check both in the hope of finding the most complete information.

Digging Deeper

As experts will be quick to point out, these comments are only the barest outline of subjects that have filled entire books. Read on for more on working in the Bash shell. If you want additional information, a good place to start is the man pages. Another important reference is the online Bash Reference Manual [1]. Read this material with a Bash shell open next to the text, so that you can try commands as you learn about them. ■

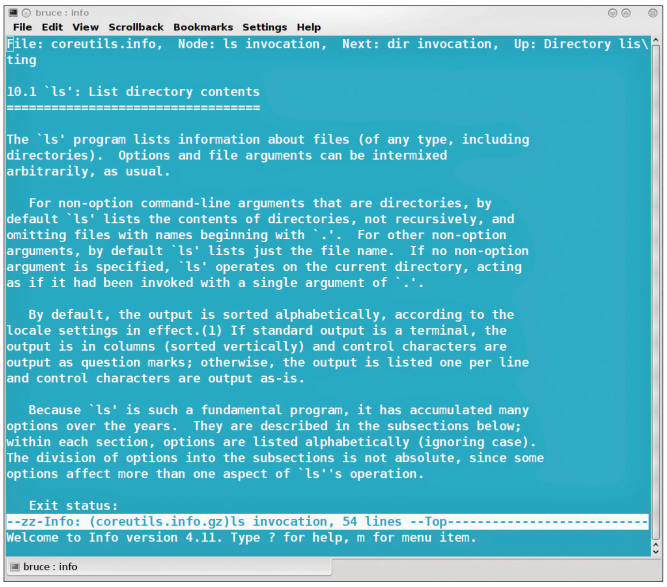
INFO

[1] Bash Reference Manual: *http://www.gnu.org/software/bash/manual/bashref.html*



```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about the FILEs (the current directory by default).  Sort
       entries alphabetically if none of -cftuvSUX nor --sort.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print octal escapes for nongraphic characters

       --block-size=SIZE
              use SIZE-byte blocks

       -B, --ignore-backups
              do not list implied entries ending with ~
 Manual page ls(1) line 1
```

Figure 4: The man page for the list command.



```
File: coreutils.info,  Node: ls invocation,  Next: dir invocation,  Up: Directory lis\
ting

10.1 `ls': List directory contents
==================================

The `ls' program lists information about files (of any type, including
directories).  Options and file arguments can be intermixed
arbitrarily, as usual.

   For non-option command-line arguments that are directories, by
default `ls' lists the contents of directories, not recursively, and
omitting files with names beginning with `.'.  For other non-option
arguments, by default `ls' lists just the file name.  If no non-option
argument is specified, `ls' operates on the current directory, acting
as if it had been invoked with a single argument of `.'.

   By default, the output is sorted alphabetically, according to the
locale settings in effect.(1) If standard output is a terminal, the
output is in columns (sorted vertically) and control characters are
output as question marks; otherwise, the output is listed one per line
and control characters are output as-is.

   Because `ls' is such a fundamental program, it has accumulated many
options over the years.  They are described in the subsections below;
within each section, options are listed alphabetically (ignoring case).
The division of options into the subsections is not absolute, since some
options affect more than one aspect of `ls''s operation.

   Exit status:
--zz-Info: (coreutils.info.gz)ls invocation, 54 lines --Top--------------------------
Welcome to Info version 4.11. Type ? for help, m for menu item.
```

Figure 5: The info page for the ls command.