## 5 – Working With Commands

Up to this point, we have seen a series of mysterious commands, each with its own mys- terious options and arguments. In this chapter, we will attempt to remove some of that mystery and even create some of our own commands. The commands introduced in this chapter are:

* type – Indicate how a command name is interpreted
* which – Display which executable program will be executed
* help – Get help for shell builtins
* man – Display a command's manual page
* apropos – Display a list of appropriate commands
* info – Display a command's info entry
* whatis – Display a very brief description of a command
* alias – Create an alias for a command

### What Exactly Are Commands?

A command can be one of four different things:

1. **An executable program** like all those files we saw in /usr/bin. Within this category, programs can be *compiled binaries* such as programs written in C and C++, or programs written in *scripting languages* such as the shell, perl, python, ruby, etc.
2. **A command built into the shell itself**. bash supports a number of commands in- ternally called *shell builtins*. The cd command, for example, is a shell builtin.
3. **A shell function.** These are miniature shell scripts incorporated into the *environ- ment*. We will cover configuring the environment and writing shell functions in later chapters, but for now, just be aware that they exist.
4. **An alias.** Commands that we can define ourselves, built from other commands.

Identifying Commands

### Identifying Commands

It is often useful to know exactly which of the four kinds of commands is being used and Linux provides a couple of ways to find out.

#### type – Display A Command's Type

The type command is a shell builtin that displays the kind of command the shell will execute, given a particular command name. It works like this:

**type *command***

where “command” is the name of the command you want to examine. Here are some ex- amples:

[me@linuxbox ~]$ **type type** type is a shell builtin [me@linuxbox ~]$ **type ls**

ls is aliased to `ls --color=tty' [me@linuxbox ~]$ **type cp**

cp is /bin/cp

Here we see the results for three different commands. Notice that the one for ls (taken from a Fedora system) and how the ls command is actually an alias for the ls command with the “-- color=tty” option added. Now we know why the output from ls is displayed in color!

#### which – Display An Executable's Location

Sometimes there is more than one version of an executable program installed on a sys- tem. While this is not very common on desktop systems, it's not unusual on large servers. To determine the exact location of a given executable, the which command is used:

[me@linuxbox ~]$ **which ls**

/bin/ls

which only works for executable programs, not builtins nor aliases that are substitutes for actual executable programs. When we try to use which on a shell builtin, for exam- ple, cd, we either get no response or an error message:

[me@linuxbox ~]$ **which cd**

/usr/bin/which: no cd in (/opt/jre1.6.0\_03/bin:/usr/lib/qt- 3.3/bin:/usr/kerberos/bin:/opt/jre1.6.0\_03/bin:/usr/lib/ccache:/usr/l ocal/bin:/usr/bin:/bin:/home/me/bin)

which is a fancy way of saying “command not found.”

### Getting A Command's Documentation

With this knowledge of what a command is, we can now search for the documentation available for each kind of command.

#### help – Get Help For Shell Builtins

bash has a built-in help facility available for each of the shell builtins. To use it, type “help” followed by the name of the shell builtin. For example:

[me@linuxbox ~]$ **help cd**

cd: cd [-L|[-P [-e]]] [dir]

Change the shell working directory.

Change the current directory to DIR. The default DIR is the value of the HOME shell variable.

The variable CDPATH defines the search path for the directory containing DIR. Alternative directory names in CDPATH are separated by a colon (:). A null directory name is the same as the current directory. If DIR begins with a slash (/), then CDPATH is not used.

If the directory is not found, and the shell option `cdable\_vars' is set, the word is assumed to be a variable name. If that variable has a value, its value is used for DIR.

Options:

-L force symbolic links to be followed

-P use the physical directory structure without following symbolic links

-e if the -P option is supplied, and the current working directory cannot be determined successfully, exit with a non-zero status

The default is to follow symbolic links, as if `-L' were specified. Exit Status:

Returns 0 if the directory is changed, and if $PWD is set

successfully when -P is used; non-zero otherwise.

**A note on notation:** When square brackets appear in the description of a command's syn- tax, they indicate optional items. A vertical bar character indicates mutually exclusive items. In the case of the cd command above:

cd [-L|[-P[-e]]] [dir]

This notation says that the command cd may be followed optionally by either a “-L” or a “-P” and further, if the “-P” option is specified the “-e” option may also be included fol- lowed by the optional argument “dir”.

While the output of help for the cd commands is concise and accurate, it is by no means tutorial and as we can see, it also seems to mention a lot of things we haven't talked about yet! Don't worry. We'll get there.

#### --help – Display Usage Information

Many executable programs support a “--help” option that displays a description of the command's supported syntax and options. For example:

[me@linuxbox ~]$ **mkdir --help**

Usage: mkdir [OPTION] DIRECTORY...

Create the DIRECTORY(ies), if they do not already exist.

-Z, --context=CONTEXT (SELinux) set security context to CONTEXT Mandatory arguments to long options are mandatory for short options too.

-m, --mode=MODE set file mode (as in chmod), not a=rwx – umask

-p, --parents no error if existing, make parent directories as needed

-v, --verbose print a message for each created directory

--help display this help and exit

--version output version information and exit

Report bugs to [<bug-](mailto:bug-coreutils@gnu.org)[coreutils@gnu.org>.](mailto:coreutils@gnu.org)

Some programs don't support the “--help” option, but try it anyway. Often it results in an error message that will reveal the same usage information.

#### man – Display A Program's Manual Page

Most executable programs intended for command line use provide a formal piece of doc- umentation called a *manual* or *man page*. A special paging program called man is used to view them. It is used like this:

**man *program***

where “program” is the name of the command to view.

Man pages vary somewhat in format but generally contain a title, a synopsis of the com- mand's syntax, a description of the command's purpose, and a listing and description of each of the command's options. Man pages, however, do not usually include examples, and are intended as a reference, not a tutorial. As an example, let's try viewing the man page for the ls command:

[me@linuxbox ~]$ **man ls**

On most Linux systems, man uses less to display the manual page, so all of the familiar

less commands work while displaying the page.

The “manual” that man displays is broken into sections and not only covers user com- mands but also system administration commands, programming interfaces, file formats and more. The table below describes the layout of the manual:

*Table 5-1: Man Page Organization*

**Section Contents**

1. User commands
2. Programming interfaces kernel system calls
3. Programming interfaces to the C library
4. Special files such as device nodes and drivers
5. File formats
6. Games and amusements such as screen savers
7. Miscellaneous
8. System administration commands

Sometimes we need to look in a specific section of the manual to find what we are look- ing for. This is particularly true if we are looking for a file format that is also the name of a command. Without specifying a section number, we will always get the first instance of a match, probably in section 1. To specify a section number, we use man like this:

**man *section search\_term***

For example:

[me@linuxbox ~]$ **man 5 passwd**

This will display the man page describing the file format of the /etc/passwd file.

#### apropos – Display Appropriate Commands

It is also possible to search the list of man pages for possible matches based on a search term. It's very crude but sometimes helpful. Here is an example of a search for man pages using the search term “floppy”:

|  |  |  |
| --- | --- | --- |
| fdformat | (8) | - Low-level formats a floppy disk |
| floppy | (8) | - format floppy disks |
| gfloppy | (1) | - a simple floppy formatter for the GNOME |
| mbadblocks | (1) | - tests a floppy disk, and marks the bad |
|  |  | blocks in the FAT |
| mformat | (1) | - add an MSDOS filesystem to a low-level |
|  |  | formatted floppy disk |

The first field in each line of output is the name of the man page, the second field shows the section. Note that the man command with the “-k” option performs the exact same function as apropos.

[me@linuxbox ~]$ **apropos floppy**

create\_floppy\_devices (8) - udev callout to create all possible

floppy device based on the CMOS type

#### whatis – Display A Very Brief Description Of A Command

The whatis program displays the name and a one line description of a man page match- ing a specified keyword:

[me@linuxbox ~]$ **whatis ls**

ls (1) - list directory contents

**The Most Brutal Man Page Of Them All**

As we have seen, the manual pages supplied with Linux and other Unix-like sys- tems are intended as reference documentation and not as tutorials. Many man pages are hard to read, but I think that the grand prize for difficulty has got to go to the man page for bash. As I was doing my research for this book, I gave it careful review to ensure that I was covering most of its topics. When printed, it's over 80 pages long and extremely dense, and its structure makes absolutely no sense to a new user.

On the other hand, it is very accurate and concise, as well as being extremely complete. So check it out if you dare and look forward to the day when you can read it and it all makes sense.

#### info – Display A Program's Info Entry

The GNU Project provides an alternative to man pages for their programs, called “info.” Info pages are displayed with a reader program named, appropriately enough, info. Info pages are *hyperlinked* much like web pages. Here is a sample:

File: coreutils.info, Node: ls invocation, Next: dir invocation, Up: Directory listing

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 filename. 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

--zz-Info: (coreutils.info.gz)ls invocation, 63 lines --Top----------

The info program reads *info files*, which are tree structured into individual *nodes*, each containing a single topic. Info files contain hyperlinks that can move you from node to node. A hyperlink can be identified by its leading asterisk, and is activated by placing the cursor upon it and pressing the enter key.

To invoke info, type “info” followed optionally by the name of a program. Below is a table of commands used to control the reader while displaying an info page:

*Table 5-2: info Commands*

**Command Action**

? Display command help

PgUp or Backspace Display previous page

PgDn or Space Display next page

n Next - Display the next node

p Previous - Display the previous node

u Up - Display the parent node of the currently displayed node, usually a menu.

Enter Follow the hyperlink at the cursor location

q Quit

Most of the command line programs we have discussed so far are part of the GNU Project's “coreutils” package, so typing:

[me@linuxbox ~]$ **info coreutils**

will display a menu page with hyperlinks to each program contained in the coreutils package.

#### README And Other Program Documentation Files

Many software packages installed on your system have documentation files residing in the /usr/share/doc directory. Most of these are stored in plain text format and can

be viewed with less. Some of the files are in HTML format and can be viewed with a web browser. We may encounter some files ending with a “.gz” extension. This indicates that they have been compressed with the gzip compression program. The gzip package includes a special version of less called zless that will display the contents of gzip- compressed text files.

### Creating Your Own Commands With alias

Now for our very first experience with programming! We will create a command of our own using the alias command. But before we start, we need to reveal a small com- mand line trick. It's possible to put more than one command on a line by separating each command with a semicolon character. It works like this:

*command1*; *command2*; *command3*...

Here's the example we will use:

[me@linuxbox ~]$ **cd /usr; ls; cd -**

bin games kerberos lib64 local share tmp etc include lib libexec sbin src

/home/me [me@linuxbox ~]$

As we can see, we have combined three commands on one line. First we change directory to /usr then list the directory and finally return to the original directory (by using 'cd

-') so we end up where we started. Now let's turn this sequence into a new command us- ing alias. The first thing we have to do is dream up a name for our new command. Let's try “test”. Before we do that, it would be a good idea to find out if the name “test” is already being used. To find out, we can use the type command again:

[me@linuxbox ~]$ **type test**

test is a shell builtin

Oops! The name “test” is already taken. Let's try “foo”:

[me@linuxbox ~]$ **type foo**

bash: type: foo: not found

Creating Your Own Commands With alias Great! “foo” is not taken. So let's create our alias:

[me@linuxbox ~]$ **alias foo=**'**cd /usr; ls; cd -'**

Notice the structure of this command:

**alias *name*='*string*'**

After the command “alias” we give alias a name followed immediately (no whitespace al- lowed) by an equals sign, followed immediately by a quoted string containing the mean- ing to be assigned to the name. After we define our alias, it can be used anywhere the shell would expect a command. Let's try it:

[me@linuxbox ~]$ **foo**

bin games kerberos lib64 local share tmp etc include lib libexec sbin src

/home/me [me@linuxbox ~]$

We can also use the type command again to see our alias:

[me@linuxbox ~]$ **type foo**

foo is aliased to `cd /usr; ls ; cd -'

To remove an alias, the unalias command is used, like so:

[me@linuxbox ~]$ **unalias foo** [me@linuxbox ~]$ **type foo** bash: type: foo: not found

While we purposefully avoided naming our alias with an existing command name, it is not uncommon to do so. This is often done to apply a commonly desired option to each invocation of a common command. For instance, we saw earlier how the ls command is often aliased to add color support:

[me@linuxbox ~]$ **type ls**

ls is aliased to `ls --color=tty'

To see all the aliases defined in the environment, use the alias command without argu- ments. Here are some of the aliases defined by default on a Fedora system. Try and figure out what they all do:

[me@linuxbox ~]$ **alias**

alias l.='ls -d .\* --color=tty' alias ll='ls -l --color=tty' alias ls='ls --color=tty'

There is one tiny problem with defining aliases on the command line. They vanish when your shell session ends. In a later chapter, we will see how to add our own aliases to the files that establish the environment each time we log on, but for now, enjoy the fact that we have taken our first, albeit tiny, step into the world of shell programming!

### Summing Up

Now that we have learned how to find the documentation for commands, go and look up the documentation for all the commands we have encountered so far. Study what addi- tional options are available and try them out!

### Further Reading

There are many online sources of documentation for Linux and the command line. Here are some of the best:

* The *Bash Reference Manual* is a reference guide to the bash shell. It’s still a ref- erence work but contains examples and is easier to read than the bash man page. <http://www.gnu.org/software/bash/manual/bashref.html>
* The *Bash FAQ* contains answers to frequently asked questions regarding bash. This list is aimed at intermediate to advanced users, but contains a lot of good in- formation.

<http://mywiki.wooledge.org/BashFAQ>

* The GNU Project provides extensive documentation for its programs, which form the core of the Linux command line experience. You can see a complete list here: <http://www.gnu.org/manual/manual.html>
* Wikipedia has an interesting article on man pages: <http://en.wikipedia.org/wiki/Man_page>





















