

The Complete Reference

Covers all
UNIX variants—
including Linux,
Solaris, HP-UX,
AIX, FreeBSD,
and Mac OS X

UNIX

Second Edition

The one must-have book on UNIX for
beginners through experienced users

Set up, administer, and manage UNIX on all
types of computers

Get complete details on GNOME, KDE, and CDE
shell programming, Perl and Python, and many
other UNIX tools and development resources

Kenneth Rosen
Douglas Host
Rachel Klee
James Farber
Richard Rosinski



Osborne

UNIX: The Complete Reference, Second Edition

Kenneth H. Rosen

Douglas A. Host

Rachel Klee

James Farber

Richard Rosinski



New York Chicago San Francisco
Lisbon London Madrid Mexico City
Milan New Delhi San Juan
Seoul Singapore Sydney Toronto

McGraw-Hill books are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. For more information, please write to the Director of Special Sales, Professional Publishing, McGraw-Hill, Two Penn Plaza, New York, NY 10121-2298. Or contact your local bookstore.

UNIX: The Complete Reference, Second Edition

Copyright © 2007 by The McGraw-Hill Companies. All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of publisher, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

1234567890 DOC DOC 019876

ISBN-13: 978-0-07-226336-7

ISBN-10: 0-07-226336-9

Sponsoring Editor

Jane Brownlow

Editorial Supervisor

Patty Mon

Project Manager

Samik Roy Chowdhury

Acquisitions Coordinator

Jennifer Housh

Technical Editor

Nalneesh Gaur

Copy Editor

Bob Campbell

Proofreader

Megha Beniwal

Indexer

Valerie Robbins

Production Supervisor

Jean Bodeaux

Composition

International Typesetting
and Composition

Illustration

International Typesetting
and Composition

Cover Designer

Jeff Weeks

Information has been obtained by McGraw-Hill from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, McGraw-Hill, or others, McGraw-Hill does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

runs the **troff** command in the background on *big_file*, sends its output to *output*, and discards error messages. In **cs**h or **tc**sh, this would look like

```
$ (troff big_file > output) >& /dev/null &
```

Logging Off with Active Jobs

If you run a command that takes a very long time, you may want to log out before it finishes. Ordinarily, if you log out while a background job is running it will be terminated. However, you can use the **nohup** (*no hang up*) command to run a job that will continue running even if you log out. For example,

```
$ nohup find / -name "lost_file" -print > lostfound 2>&1 &
```

in the Bourne-compatible shells, or

```
$ nohup find / -name "lost_file" -print >& lostfound &
```

in the C shells, allow **find** to continue even after you quit. This command starts looking in the root directory of the file system for any files named *lost_file*. Any pathnames that are found are put in the file named *lostfound*, along with any error messages. The whole thing is run in the background, to allow you to enter other commands or log out.

When you use **nohup**, you should be sure to redirect both standard output and standard error to files, so that when you log back in you can find out what happened. If you do not specify output files, **nohup** automatically sends command output, including standard error, to the file *nohup.out*.

Job Control

Because the UNIX System provides the capability to run commands in the background, you sometimes have two or more commands running at once. There is always one job in the foreground. This may be the shell, when it is prompting you for input, or it may be any other command to which your keyboard input is connected, such as a text editor. In addition, there may be several jobs running in the background at any given time.

Job control is a crucial feature of the modern UNIX shells that was first introduced in **cs**h and is also found in **tc**sh, **ksh**, and **bash**. The commands and syntax are for the most part identical in all four shells. The job control commands allow you to terminate a background job (**kill** it), suspend a background job temporarily (**stop** it), resume a suspended job in the background, move a background job to the foreground, and suspend a foreground job.

The **jobs** command displays a list of all your jobs, such as

```
$ jobs
[1] +  Running          find /home/jmf -print > files &
[2]   Stopped          vi filesplit.tcl
[3] -  Stopped          grep supv * | awk -f fixes > data &
```

The output shows your current foreground and background jobs, as well as jobs that are stopped or suspended. In this example, there are three jobs. The number at the beginning of each line is the *job ID*. Job 1 (the **find** command) is running in the background. Jobs 2 and 3

are both stopped. The plus sign (+) indicates the current job (the most recently started or restarted); minus (-) indicates the one before that.

You can suspend your current foreground job by typing `CTRL-Z`. This halts the program and returns you to your shell. For example, if you are running a command that is taking a long time, type `CTRL-Z` to suspend it so that you can do something else. The job will essentially be paused—that is, it will not do anything until you resume the job, but you can resume it at any time.

Suppose you ran the `find` command, and forgot to tell it to run in the background. You can use `CTRL-Z` to suspend it, but now you need to tell it to resume. The command `bg` will cause a job to run in the background. If the job is stopped, it will resume. By default, `bg` acts on the current job. To resume an older job, refer to it by the job ID:

```
$ jobs
[1] + Stopped          find /home/jmf -print > files
$ bg %1
[1] + find /home/jmf -print > files &
```

You use the % sign to introduce the job identifier, so %1 refers to job 1.

Similarly, the command `fg` causes an existing job to run in the foreground. This can be used to resume a suspended job, or to move a background job to the foreground.

You can terminate any of your background or suspended jobs with the `kill` command. For example,

```
$ kill %2
```

terminates job number 2. Once a job is killed, it is gone—it can't be resumed.

In addition to the job ID number, you can use the name of the command to tell the shell which job to kill. For instance,

```
$ kill %troff
```

kills a `troff` job running in the background. If you have two or more `troff` commands running, this will kill the most recent one.

The `stop` command halts execution of a background job but doesn't terminate it, just like `CTRL-Z` for foreground jobs. The command sequence

```
$ stop %find
$ fg %find
```

stops the `find` command that is running in the background and then resumes executing it in the foreground. The `stop` command is supported by `csh`, `tcsh`, and `ksh`, but not by `bash`. In `bash`, you would use the following command line instead:

```
$ kill -s STOP %find
```

Table 4-2 summarizes the shell job control commands

Command	Effect
<code>jobs</code>	List all jobs
<code>Ctrl-Z</code>	Suspend current (foreground) process
<code>bg %n</code>	Resume stopped job in background
<code>fg %n</code>	Resume job in foreground
<code>stop %n</code>	Suspend background job
	In bash , use kill -s STOP %n
<code>kill %n</code>	Terminate job

TABLE 4-2 Job Control Commands

Configuring the Shell

When your login shell starts up, it looks for certain files in your home directory. These files contain commands that can be used to configure your working environment. The particular files it looks for depend on which shell you are using:

- **sh** runs the commands in a configuration file called *.profile*.
- **ksh** also uses the *.profile* file. In addition, you can set a variable in your *.profile* to cause it to read the commands in a second file. The variable is called **ENV**. By convention, that file is often called *.kshrc*.
- **bash** uses the file *.bash_profile*. If that file does not exist, it will look for the file *.profile*, instead. The *.bash_profile* often contains a line that causes **bash** to run the commands in a second file, *.bashrc*. When you log out of **bash**, it will run the commands in *.bash_logout*.
- **csh** looks for a file called *.login*. It will also run commands in *.cshrc*. When you log out of the C shell, it will run the commands in the file *.logout*.
- **tcsh** uses the *.login* file as well. It also looks for *.tshrc*. If that file does not exist, it will look for *.cshrc* instead. Like the C shell, **tcsh** will run the *.logout* file when you log out.

This may all sound very confusing, but these configuration files all work in pretty much the same way. Each of these files is actually an example of a shell script. They contain commands or instructions for the shell. The commands include settings that allow you to customize your environment.

The first file that the shell reads (*.profile*, *.bash_profile*, or *.login*) contains variables or settings that you want to be in effect throughout your login session. The section "Shell Variables" later in this chapter describes these variables. The file might also include commands you want to run at login, such as `who` (to show the list