FOURTH EDITION

"First Sobell taught people how to use Linux... now he teaches you the power of Linux. A must-have book for anyone who wants to take Linux to the next level."

—**Jon "maddog" Hall,** Executive Director, Linux International

A Practical Guide to Linux® Commands, Editors, and Shell Programming

Discover the Power of Linux— Covers macOS, too!

- Learn from hundreds of realistic, high-quality examples, and become a true command-line guru
- Covers MariaDB, DNF, and Python 3
- 300+ page reference section covers 102 utilities, including macOS commands

For use with all popular versions of Linux, including Ubuntu,™ Fedora,™ openSUSE,™ Red Hat,® Debian, Mageia, Mint, Arch, CentOS, and macOS

Mark G. Sobell

coauthored by **Matthew Helmke**

Utility Index

A light page number such as 456 indicates a brief mention. Page numbers followed by the letter **t** refer to tables.

Symbols

: (null) 486, 498
. (dot) 291, 493
[[...]] 506
@ 396, 398, 400, 418
% 418

A

alias 352, 387, 418
alloc 418
apropos 35
apt-get 1060–1064
ash *see* dash
aspell 449, 739–742
at 743–746
atq 743–746
atrm 743–746
atrun (macOS) 744
autoconf 771
awk *see* mawk

B

basename 463
bash *see* bash in the Main index (page 1141)
batch 743–746
bg 151, 306, 418
bind 350
bindkey 393, 394, 418
builtins 419, 504t
bunzip2 65, 750–751
busybox 747–749
bzcat 65, 750–751
bzip2 64, 750–751
bzip2recover 65, 750–751

C

cal 752
cat 52, 138, 141, 145, 436, 710, 753–754
cd 94, 117, 323, 419, 755–756
chdir 419
chgrp 121, 757–758
chmod 102–104, 295, 756–763
chmod (macOS) 1074–1076
chown 764–765
chsh 287, 381
cmp 766–767
col 900
comm 768–769
compress 65, 86, 859, 997, 997
configure 770–771
cp 53, 97, 131, 772–775
cpio 776–780, 823, 826, 860
cron 781–783
crontab 781–783
curlftpfs 983–985
cut 369, 784–785

D

dash 287
date 62, 329, 330, 331, 383, 472, 787–789
dd 711, 790–792
declare 315–316, 487
df 793–794
diff 59, 795–799
diff3 796
dircolors 888
dirs 307, 389, 419
disktool (macOS) 801
diskutil (macOS) 800–802
ditto (macOS) 803–804
dmesg 805
dos2unix 63
dscl (macOS) 806–808
du 809–811

E

e2fsck 835
echo 61, 154, 381, 419, 457, 458, 476, 812–813
ed 56, 166, 795
egrep 853, 854, 1044
elvis 166
emacs 222–278
env 358, 483
eval 358, 419, 500
ex 166, 175
exec 416, 419, 465, 493–496
exit 382, 407, 419, 434, 477
expand 814–815
export 316, 481–482
expr 816–819

F

false 302
fc 338–340
fg 151, 305, 419
fgrep 853, 854
file 60, 332, 820–821
filetest 411, 419
find 390, 442, 822–827
finger 72, 828–829
fmt 831–832
free 74
fromdos 63
fsck 833–837
fsck (macOS) 801
ftp 838–843
fusermount 981–983

G

g++ 850–854
gawk 447, 636–668, 1044
gcc 846–850
gdb 847
getfacl 108–112
GetFileInfo (macOS) 851–852
getopts 501–503
getty 333
glob 420
grep 56, 148, 450, 462, 531, 827, 853–857
gunzip 66, 858–860, 997
gzip 66, 858–860, 997

H

hash 336
hashstat 420
head 57, 861–862
history 336, 337, 384, 420
hostname 53

I

info 36–38, 158
ispell *see* aspell

J

jobs 30, 152, 305, 420
join 863–865

K

kill 30, 152, 424, 496, 499, 866–867, 949
killall 868–869

L

launchctl (macOS) 745, 870–871
ldd 928
less 34, 53, 148, 436, 873–876
let 370, 505
lftp 715
limit 420
ln 113, 116, 878–880
local 357, 488
locale 328–330
locate 70
log 420
login 333, 420
logout 382, 407, 420
lpq 56, 881–883

Continued on inside back cover

You can customize the prompt displayed by **PS1**. For example, the assignment

```
$ PS1="[\u@\h \W \!]$ "
```

displays the prompt

[user@host directory event]$

where ***user*** is the username, ***host*** is the hostname up to the first period, ***directory*** is the basename of the working directory, and ***event*** is the event number (page 337) of the current command.

If you are working on more than one system, it can be helpful to incorporate the system name into your prompt. The first example that follows changes the prompt to the name of the local host, a SPACE, and a dollar sign (or, if the user is running with **root** privileges, a hashmark), followed by a SPACE. A SPACE at the end of the prompt makes commands you enter following the prompt easier to read. The second example changes the prompt to the time followed by the name of the user. The third example changes the prompt to the one used in this book (a hashmark for a user running with **root** privileges and a dollar sign otherwise):

```
$ PS1='\h \$ '
guava $

$ PS1='\@ \u $ '
09:44 PM max $

$ PS1='\$ '
$
```

Table 8-4 describes some of the symbols you can use in **PS1**. See Table 9-4 on page 403 for the corresponding tcsh symbols. For a complete list of special characters you can use in the prompt strings, open the bash man page and search for the third occurrence of **PROMPTING** (enter the command **/PROMPTING** followed by a RETURN and then press **n** two times).

Table 8-4 **PS1** symbols

Symbol	Display in prompt
\$	**#** if the user is running with **root** privileges; otherwise, **$**
\w	Pathname of the working directory
\W	Basename of the working directory
\!	Current event (history) number (page 341)
\d	Date in Weekday Month Date format
\h	Machine hostname, without the domain
\H	Full machine hostname, including the domain
\u	Username of the current user

Table 8-4 **PS1** symbols (continued)

Symbol	Display in prompt
\@	Current time of day in 12-hour, AM/PM format
\T	Current time of day in 12-hour HH:MM:SS format
\A	Current time of day in 24-hour HH:MM format
\t	Current time of day in 24-hour HH:MM:SS format

PS2: User Prompt (Secondary)

The **PS2** variable holds the secondary prompt (**prompt2** under tcsh). On the first line of the next example, an unclosed quoted string follows echo. The shell assumes the command is not finished and on the second line displays the default secondary prompt (>). This prompt indicates the shell is waiting for the user to continue the command line. The shell waits until it receives the quotation mark that closes the string and then executes the command:

```
$ echo "demonstration of prompt string
> 2"
demonstration of prompt string
2
```

The next command changes the secondary prompt to **Input =>** followed by a SPACE. On the line with who, a pipe symbol (|) implies the command line is continued (page 512) and causes bash to display the new secondary prompt. The command **grep sam** (followed by a RETURN) completes the command; grep displays its output.

```
$ PS2="Input => "
$ who |
Input => grep sam
sam      tty1        2018-05-01 10:37 (:0)
```

PS3: Menu Prompt

The **PS3** variable holds the menu prompt (**prompt3** in tcsh) for the **select** control structure (page 461).

PS4: Debugging Prompt

The **PS4** variable holds the bash debugging symbol (page 443; not in tcsh).

IFS: Separates Input Fields (Word Splitting)

The **IFS** (Internal Field Separator) shell variable (not in tcsh) specifies the characters you can use to separate arguments on a command line. It has the default value of SPACE-TAB-NEWLINE. Regardless of the value of **IFS**, you can always use one or more SPACE or TAB characters to separate arguments on the command line, provided these characters are not quoted or escaped. When you assign character values to **IFS**, these characters can also separate fields—but only if they undergo expansion. This type of interpretation of the command line is called *word splitting* and is discussed on page 372.

Be careful when changing IFS

caution Changing **IFS** has a variety of side effects, so work cautiously. You might find it useful to save the value of **IFS** before changing it. You can then easily restore the original value if a change yields unexpected results. Alternatively, you can fork a new shell using a **bash** command before experimenting with **IFS**; if you run into trouble, you can **exit** back to the old shell, where **IFS** is working properly.

The following example demonstrates how setting **IFS** can affect the interpretation of a command line:

```
$ a=w:x:y:z

$ cat $a
cat: w:x:y:z: No such file or directory
$ IFS=":"

$ cat $a
cat: w: No such file or directory
cat: x: No such file or directory
cat: y: No such file or directory
cat: z: No such file or directory
```

The first time cat is called, the shell expands the variable **a**, interpreting the string **w:x:y:z** as a single word to be used as the argument to cat. The cat utility cannot find a file named **w:x:y:z** and reports an error for that filename. After **IFS** is set to a colon (**:**), the shell expands the variable **a** into four words, each of which is an argument to cat. Now cat reports errors for four files: **w**, **x**, **y**, and **z**. Word splitting based on the colon (**:**) takes place only *after* the variable **a** is expanded.

The shell splits all *expanded* words on a command line according to the separating characters found in **IFS**. When there is no expansion, there is no splitting. Consider the following commands:

```
$ IFS="p"
$ export VAR
```

Although **IFS** is set to **p**, the **p** on the **export** command line is not expanded, so the word **export** is not split.

The following example uses variable expansion in an attempt to produce an **export** command:

```
$ IFS="p"
$ aa=export
$ echo $aa
ex ort
```

This time expansion occurs, so the **p** in the token **export** is interpreted as a separator (as the echo command shows). Next, when you try to use the value of the **aa** variable to export the **VAR** variable, the shell parses the **$aa VAR** command line as **ex ort VAR**. The effect is that the command line starts the ex editor with two filenames: **ort** and **VAR**.

```
$ $aa VAR
2 files to edit
"ort" [New File]
Entering Ex mode.  Type "visual" to go to Normal mode.
:q
E173: 1 more file to edit
:q
$
```

If **IFS** is unset, bash uses its default value (SPACE-TAB-NEWLINE). If **IFS** is null, bash does not split words.

Multiple separator characters

tip Although the shell treats sequences of multiple SPACE or TAB characters as a single separator, it treats *each occurrence* of another field-separator character as a separator.

CDPATH: Broadens the Scope of cd

The **CDPATH** variable (**cdpath** under tcsh) allows you to use a simple filename as an argument to the cd builtin to change the working directory to a directory other than a child of the working directory. If you typically work in several directories, this variable can speed things up and save you the tedium of using cd with longer pathnames to switch among them.

When **CDPATH** is not set and you specify a simple filename as an argument to cd, cd searches the working directory for a subdirectory with the same name as the argument. If the subdirectory does not exist, cd displays an error message. When **CDPATH** is set, cd searches for an appropriately named subdirectory in the directories in the **CDPATH** list. If it finds one, that directory becomes the working directory. With **CDPATH** set, you can use cd and a simple filename to change the working directory to a child of any of the directories listed in **CDPATH**.

The **CDPATH** variable takes on the value of a colon-separated list of directory pathnames (similar to the **PATH** variable). It is usually set in the **~/.bash_profile** startup file with a command line such as the following:

```
export CDPATH=$HOME:$HOME/literature
```

This command causes cd to search your home directory, the **literature** directory, and then the working directory when you give a cd command. If you do not include the working directory in **CDPATH**, cd searches the working directory if the search of all the other directories in **CDPATH** fails. If you want cd to search the working directory first, include a colon (**:**) as the first entry in **CDPATH**:

```
export CDPATH=:$HOME:$HOME/literature
```

If the argument to the cd builtin is anything other than a simple filename (i.e., if the argument contains a slash [**/**]), the shell does not consult **CDPATH**.

Keyword Variables: A Summary

Table 8-5 lists the bash keyword variables. See page 402 for information on tcsh variables.

Table 8-5 bash keyword variables

Variable	Value
BASH_ENV	The pathname of the startup file for noninteractive shells (page 289)
CDPATH	The cd search path (page 323)
COLUMNS	The width of the display used by **select** (page 460)
HISTFILE	The pathname of the file that holds the history list (default: **~/.bash_history**; page 336)
HISTFILESIZE	The maximum number of entries saved in **HISTFILE** (default: 1,000–2,000; page 336)
HISTSIZE	The maximum number of entries saved in the history list (default: 1,000; page 336)
HOME	The pathname of the user's home directory (page 317); used as the default argument for cd and in tilde expansion (page 91)
IFS	Internal Field Separator (page 321); used for word splitting (page 372)
INPUTRC	The pathname of the Readline startup file (default: **~/.inputrc**; page 349)
LANG	The locale category when that category is not specifically set using one of the **LC_** variables (page 327)
LC_	A group of variables that specify locale categories including **LC_ALL**, **LC_COLLATE**, **LC_CTYPE**, **LC_MESSAGES**, and **LC_NUMERIC**; use the locale builtin (page 328) to display a more complete list including values
LINES	The height of the display used by **select** (page 460)
MAIL	The pathname of the file that holds a user's mail (page 319)
MAILCHECK	How often, in seconds, bash checks for mail (default: 60; page 319)
MAILPATH	A colon-separated list of file pathnames that bash checks for mail in (page 319)
OLDPWD	The pathname of the previous working directory
PATH	A colon-separated list of directory pathnames that bash looks for commands in (page 318)
PROMPT_COMMAND	A command that bash executes just before it displays the primary prompt

Table 8-5 bash keyword variables (continued)

Variable	Value
PS1	Prompt String 1; the primary prompt (page 319)
PS2	Prompt String 2; the secondary prompt (page 321)
PS3	The prompt issued by **select** (page 460)
PS4	The bash debugging symbol (page 443)
PWD	The pathname of the working directory
REPLY	Holds the line that read accepts (page 490); also used by **select** (page 460)

Special Characters

Table 8-6 lists most of the characters that are special to the bash and tcsh shells.

Table 8-6 Shell special characters

Character	Use
NEWLINE	A control operator that initiates execution of a command (page 300)
;	A control operator that separates commands (page 300)
( )	A control operator that groups commands (page 302) for execution by a subshell; these characters are also used to identify a function (page 356)
(( ))	Evaluates an arithmetic expression (page 505)
&	A control operator that executes a command in the background (pages 150 and 300)
|	A control operator that sends standard output of the preceding command to standard input of the following command (pipeline; page 300)
|&	A control operator that sends standard output and standard error of the preceding command to standard input of the following command (page 293)
>	Redirects standard output (page 140)
>>	Appends standard output (page 144)
<	Redirects standard input (page 142)
<<	Here document (page 462)
*	Matches any string of zero or more characters in an ambiguous file reference (page 154)