CHAPTER

3

MASTERING EDITORS

After reading this chapter and completing the exercises, you will be able to:

- Explain the basics of UNIX/Linux files, including ASCII, binary, and executable files
- Understand the types of editors
- Create and edit files using the vi editor
- Create and edit files using the Emacs editor

The ability to create and modify the contents of files is a fundamental skill not only in producing documents such as memos, reports, and letters, but also in writing programs and customizing system configuration files. All operating systems, including UNIX/Linux, provide one or more editors that enable you to work with the contents of files.

This chapter introduces two important UNIX/Linux editors, which you use throughout the rest of this book. The vi editor provides basic editing functions and is often preferred by UNIX/Linux administrators and programmers for its simplicity. The Emacs editor offers more sophisticated editing capabilities for writing all kinds of documents as well as programs. Both of these popular editors can be started from the command line and are included in most versions of UNIX/Linux.

Understanding UNIX/Linux Files

Almost everything you create in UNIX/Linux is stored in a file. All information stored in files is in the form of binary digits. A binary digit, called a **bit** for short, is in one of two states. The states are 1 (on) and 0 (off). They can indicate, for example, the presence or absence of voltage in an electronic circuit. Because the computer consists of electronic circuits that are either in an on or off state, binary numbers are perfectly suited to report these states. The exclusive use of 0s and 1s as a way to communicate with the computer is known as **machine language**. The earliest programmers had to write their programs using machine language, a tedious and time-consuming process.

ASCII Text Files

To make information stored in files accessible, computer designers established a standard method for translating binary numbers into plain English. This standard uses a string of eight binary digits, called a **byte**, which is the abbreviation for binary term. A byte can be configured into fixed patterns of bits, and these patterns can be interpreted as an alphabetic character, decimal number, punctuation mark, or a special character, such as &, *, or @. Each byte, or code, has been standardized into a set of bit patterns known as ASCII. **ASCII** stands for the American Standard Code for Information Interchange. Computer files containing nothing but printable characters are called **text files**, and files that contain nonprintable characters, such as machine instructions, are called **binary files**. The ASCII character set represents 256 characters. Figure 3-1 lists the printable and nonprintable ASCII characters.

Many nonprintable ASCII characters are available. Some examples are characters used to control printers, such as the escape (ESC) character to show the start of a printing command, a form feed (FF) character, and a line feed (LF) character. If you use Microsoft Word or OpenOffice.org Writer, you are familiar with other nonprinting characters, such as the paragraph symbol. You can view nonprinting characters in Microsoft Word or OpenOffice.org Writer by clicking the Show/Hide ¶ or Nonprinting Characters button (paragraph symbol) on the Standard toolbar.

Some operating systems also support **Unicode**. Unicode offers up to 65,536 characters, although not all of the possible characters are currently defined. Unicode was developed because the 256 characters in ASCII are not enough for some languages, such as Chinese, that use more than 256 characters. Visit *www.unicode.org* to learn more about Unicode.

Binary Files

Computers are not limited to processing ASCII codes. To work with graphic information, such as icons, illustrations, and other images, binary files can include strings of bits representing white and black dots, in which each black dot represents a 1 and each white dot

Printing Characters (Punctuation Characters)

Dec	*Octal*	*Hex*	*ASCII*
32	040	20	(Space)
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-
46	056	2E	.
47	057	2F	/

(Decimal Numbers—Print)

Dec	*Octal*	*Hex*	*ASCII*
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9

(Special Characters—Print)

Dec	*Octal*	*Hex*	*ASCII*
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	080	40	@

Printing Characters (Alphabet—Uppercase)

Dec	*Octal*	*Hex*	*ASCII*
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z

Printing Characters (Alphabet—Lowercase)

Dec	*Octal*	*Hex*	*ASCII*
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z

Nonprinting Characters (Abridged)

Control Characters

Dec	*Octal*	*Hex*	*ASCII*
0	000	00	^@ (Null)
7	007	07	Bell
8	010	08	Backspace
9	011	09	Tab
10	012	0A	Line Feed, Newline
11	013	0B	Vertical tab
12	014	0C	Form feed
13	015	0D	Carriage return

Figure 3-1 ASCII characters

represents a 0. Graphics files include bit patterns—rows and columns of dots called a **bitmap**—that must be translated by graphics software, commonly called a graphics viewer, which transforms a complex array of bits into an image.

Executable Program Files

Many programmers develop source code for their programs by writing text files; then, they compile these files to convert them into executable program files. **Compiling** is a process of translating a program file into machine-readable language. Programmers and users also develop

scripts, which are files containing commands. Scripts are typically not compiled into machine code prior to running, but are executed through an interpreter. At the time the script is run, the interpreter looks at each line and converts the commands on each line into actions taken by the computer. Scripts are interpreted program files that are executable. You learn more about writing program code, scripts, compilers, and interpreters later in this book.

Compiled and interpreted files that can be run are called **executable program files** (or sometimes just **executables**). These files can be run from the command line.

Using Editors

An **editor** is a program for creating and modifying files containing source code, text, data, memos, reports, and other information. A **text editor** is like a simplified word-processing program; you can use a text editor to create and edit documents, but many text editors do not allow you to format text using boldface text, centered text, or other text-enhancing features.

Editors let you create and edit ASCII files. UNIX/Linux normally include the two editors vi and Emacs. They are **screen editors**, because they display the text you are editing one screen at a time and let you move around the screen to change and add text. Both are text editors as well, because they work like simple word processors. You can also use a line editor to edit text files. A **line editor** lets you work with only one line or group of lines at a time. Although line editors do not let you see the context of your editing, they are useful for general tasks, such as searching, replacing, and copying blocks of text. In UNIX/Linux, however, most users prefer vi or Emacs to using a simple line editor, which is another reason why vi and Emacs are included with UNIX/Linux systems.

The vi and Emacs editors do not offer the same functionality as GUI-based editors such as Microsoft Word (although Emacs now has lots of functionality and "snap-ins" for extra functions like editing Web documents). Also, UNIX/Linux systems can use more sophisticated GUI editors, such as OpenOffice.org Writer, gedit in the GNOME desktop, and KEdit in the KDE desktop. However, both vi and Emacs are typically preferred for system, configuration, and programming activities, because they are quickly initiated from the command line and offer a simple, direct way to perform critical editing tasks.

Using the vi Editor

The vi editor is so called because it is visual—it immediately displays on screen the changes you make to text. It is also a **modal editor**; that is, it works in three modes: insert mode, command mode, and extended (ex) command set mode. **Insert mode**, which lets you enter

text, is accessed by typing the letter "i" after the vi editor is started. **Command mode**, which is started by pressing Esc, lets you enter commands to perform editing tasks, such as moving through the file and deleting text. **Ex mode** employs an extended set of commands that were initially used in an early UNIX editor called ex. You can access this mode by pressing Esc to enter command mode, and then typing a colon (:) to enter extended commands at the bottom of the screen.

You can simulate a line editor using vi by starting the vi editor with the *-e* option (*vi -e filename*), which places vi exclusively in ex mode. Also, when you open vi, it is set up by default to edit a text file. You can edit a binary file by using the *-b* option with vi.

To use the vi editor, it is important to master the following tasks:

- Creating a file
- Inserting, editing, and deleting text
- Searching and replacing text
- Adding text from other files
- Copying, cutting, and pasting text
- Printing a file
- Saving a file
- Exiting a file

Different versions of the vi editor are included in different versions of UNIX/Linux. The commands described in this chapter generally apply to most UNIX/Linux vi editor versions. However, they particularly apply to the vi editor in Fedora, Red Hat Enterprise Linux, and SUSE, which is technically called the vim (vi improved) editor.

Creating a New File in the vi Editor

To create a new file in the vi editor:

1. Access the command line.
2. Enter *vi* plus the name of the file you want to create, such as *vi data*.

These steps open the vi editor and enable you to begin entering text in the file you specify. Remember, though, at this point the file is in memory and is not permanently saved to the disk until you issue the command to save it.

As you enter text, the line containing the cursor is the current line. Lines containing a tilde (~) are not part of the file; they indicate lines on the screen only, not lines of text in the file. (See Figure 3-2.)

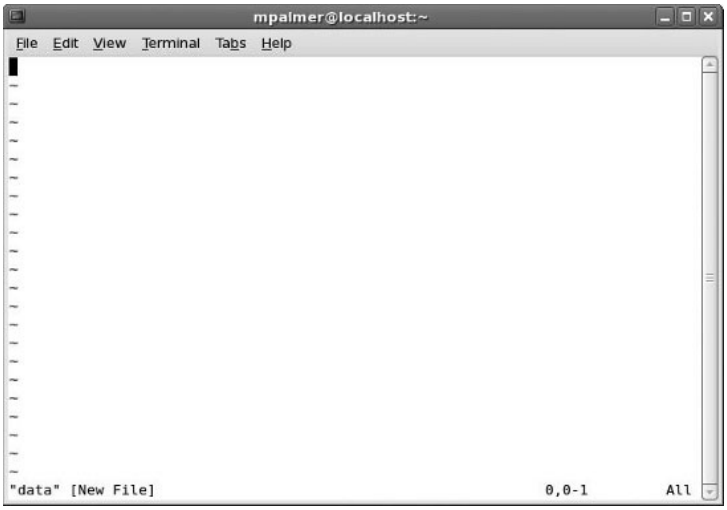

Figure 3-2 Creating a new file in the vi editor

Hands-on Project 3-1 enables you to create a new file using the vi editor.

Sometimes you might open the vi editor without specifying the file name with the vi command on the command line. You can save the file and specify a file name at any time by pressing Esc, typing *:w filename*, and pressing Enter.

Inserting Text

When you start the vi editor, you're in command mode. This means that the editor interprets anything you type on the keyboard as a command. Before you can insert text in your new file, you must use the *i* (insert) command. In insert mode, every character you type appears on the screen. You can return to command mode at any time by pressing the Esc key.

Try Hands-on Project 3-2 to insert text in the vi editor.

Repeating a Change

The vi editor offers features that can save you time. One such feature is the ability to replicate any changes you make. When you are in command mode, you can use a period (.) to repeat the most recent change you made. This is called the repeat command and it can save you time when typing the same or similar text. Hands-on Project 3-3 uses the repeat command.

Moving the Cursor

When you want to move the cursor to a different line or to a specific position on the same line, use command mode (press Esc). In command mode, you can move forward or back one

word, move up or down a line, go to the beginning of the file, and so on. Table 3-1 summarizes useful cursor-movement commands. You can practice moving the cursor in Hands-on Project 3-4.

Table 3-1 vi editor's cursor movement keys

Key	Movement
h or left arrow	Left one character position
l or right arrow	Right one character position
k or up arrow	Up one line
j or down arrow	Down one line
H	Upper-left corner of the screen
L	Last line on the screen
G	Beginning of the last line
nG	The line specified by a number, *n*
W	Forward one word
b	Back one word
0 (zero)	Beginning of the current line
$	End of the current line
Ctrl+u	Up one-half screen
Ctrl+d	Down one-half screen
Ctrl+f or *Page Down*	Forward one screen
Ctrl+b or *Page Up*	Back one screen

Remember that the Ctrl key combinations and the letter keys shown in Table 3-1 are designed to work in command mode. The arrow keys, which are used for moving around text, work in both command and insert mode. Try Hands-on Project 3-4 to practice using vi in command mode to move around in a file.

Using the letter keys to move the cursor can be traced to the time when UNIX/Linux used teletype terminals that had no arrow keys. Designers of vi chose the letter keys because of their relative position on the keyboard.

Deleting Text

The vi editor employs several commands for deleting text when you are in command mode. For example, to delete the text at the cursor, type *x*. Use *dd* in command mode to delete the current line. Use *dw* to delete a word or to delete from the middle of a word to the end of the word. To delete more than one character, combine the delete commands with the cursor movement commands you learned in the preceding section. Table 3-2 summarizes the most common delete commands.

Table 3-2 vi editor's delete commands

Command	Purpose
x	Delete the character at the cursor.
dd	Delete the current line (putting it in a buffer so it can also be pasted back into the file).
dw	Delete the word starting at the cursor. If the cursor is in the middle of the word, delete from the cursor to the end of the word.
d$	Delete from the cursor to the end of the line.
d0	Delete from the cursor to the start of the line.

The command to delete a line, *dd*, actually places deleted lines in a buffer. You can then use the command *p* to paste deleted (cut) lines elsewhere in the text. (Position the cursor where you want to paste the information.) To copy and paste text, use the "yank" command, *yy*, to copy the lines. After yanking the lines you want to paste elsewhere, move the cursor, and type *p* to paste the text in the current location. You learn more about the *p* and *yy* commands in later sections of this chapter.

Hands-on Project 3-5 gives you the opportunity to practice using delete commands.

Undoing a Command

If you complete a command and then realize you want to reverse its effects, you can use the undo (*u*) command. For example, if you delete a few lines from a file by mistake, type *u* to restore the text.

Searching for a Pattern

You can search forward for a pattern of characters by typing a forward slash (/), typing the pattern you are seeking, and then pressing Enter. Programmers often call this a "string search." For example, suppose you want to know how many times you used the word "insure" in a file. First, go to the top of the file, type */insure*, and press Enter to find the first instance of insure. To find more instances, type *n* while you are in command mode.

When placed after the forward slash, several special characters are supported by vi when searching for a pattern. For example, the special characters \> are used to search for the next word that ends with a specific string. If you enter */te*\> you find the next word that ends with "te," such as "write" or "byte." The characters \< search for the next word that begins with a specific string, such as using /\<*top* to find the next word that begins with "top," which might be "topology," for example. The ^ special character searches for the next line that begins with a specific pattern. For instance, */^However* finds the next line that starts with "However." Use a period as a wildcard to match characters. For example, */m.re* finds "more" and "mere," and */s..n* finds "seen," "soon," and "sign." Also, use brackets *[ ]* to find any of the characters between the brackets, such as */theat[er]* to find "theater" or "theatre," and */pas[st]* to find "pass" or "past." Finally, use *$* to find a line that ends with a specific character. For instance */!$* finds a line that ends with an exclamation point "!". You can type *n* after searching with any of these special

characters to find the next pattern that matches. Table 3-3 summarizes the special characters used to match a pattern.

Table 3-3 Special characters used to match a pattern

Special Character*	Purpose
\>	Searches for the next word that ends with a specific string.
\<	Searches for the next word that begins with a specific string.
.	Acts as a wildcard for one character.
[]	Finds the characters between the brackets.
$	Searches for the line that ends with a specific character.
*All of these special characters must be preceded with a slash (/) from the command mode.	

Hands-on Project 3-6 provides experience in pattern matching.

If you are in an editing session and want to review information about the file status, press Ctrl+g or Ctrl+G (you can use uppercase G or lowercase g). The status line at the bottom of the screen displays information, including line-oriented commands and error messages.

Searching and Replacing

Suppose you want to change all occurrences of "insure" in the file you are editing to "ensure." Instead of searching for "insure," and then deleting it and inserting "ensure," you can search and replace with one command. The commands you learned so far are screen-oriented. Commands that can perform more than one action (searching and replacing) are line-oriented commands and they operate in ex mode.

Screen-oriented commands execute at the location of the cursor. You do not need to tell the computer where to perform the operation because it takes place relative to the cursor. **Line-oriented commands**, on the other hand, require you to specify an exact location (an address) for the operation. Screen-oriented commands are easy to type, and their changes appear on the screen. Typing line-oriented commands is more complicated, but they can execute independently of the cursor and in more than one place in a file, saving you time and keystrokes.

A colon (:) precedes all line-oriented commands. It acts as a prompt on the status line, which is the bottom line on the screen in the vi editor. You enter line-oriented commands on the status line, and press Enter when you complete the command.

In this chapter, all instructions for line-oriented commands include the colon as part of the command.

For example, to replace all occurrences of "insure" with "ensure," you first enter command mode (press Esc), type *:1,$s/insure/ensure/g*, and press Enter. This command means access the ex mode (:), beginning with the first line (*1*) to the end of the file (*$*), search for "insure," and replace it with "ensure" (*s/insure/ensure/*) everywhere it occurs on each line (g).

Try Hands-on Project 3-7 to use a line-oriented command for searching and replacing.

Saving a File and Exiting vi

As you edit a file, periodically saving your changes is a good idea. This is especially true if your computer is not on an uninterruptible power supply (UPS). A UPS is a device that provides immediate battery power to equipment during a power failure or brownout.

You can save a file in several ways. One way is to enter command mode and type *:w* to save the file without exiting. (See Figure 3-3.) If you are involved in a relatively long editing session, consider using this command every 10 minutes or so to periodically save your work. If you want to save your changes and exit right away, use *:wq* or *ZZ* from command mode. You can also use *:x* to save and exit. You should always save the file before you exit vi; otherwise, you will lose your changes. Hands-on Project 3-8 enables you to save your changes and exit an editing session.

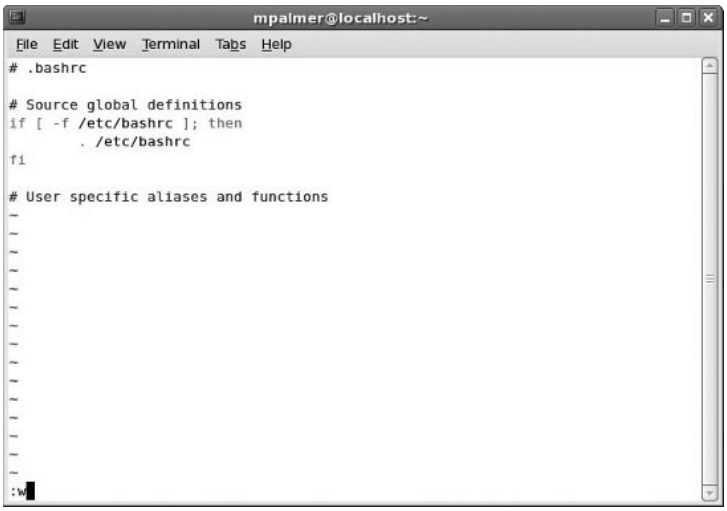

Figure 3-3 Saving without exiting

Adding Text from Another File

Sometimes, the text you want to include in one file is already part of another file. For example, suppose you already have a text file that lists customer accounts and you have another file called customerinfo that contains customer information. You want to copy the customer accounts text into the customerinfo file and make further changes to the

customerinfo file. It is much easier to use the vi command to copy the text from the accounts file into the customerinfo file than it is to retype all of the text. To copy the entire contents of one file into another file: (1) use the vi editor to edit the file you want to copy into; and (2) use the command *:r filename*, where *filename* is the name of the file that contains the information you want to copy. Hands-on Project 3-9 enables you to copy from one file into another.

Leaving vi Temporarily

If you want to execute other UNIX/Linux commands while you work with vi, you can launch a shell or execute other commands from within vi. For example, suppose you're working on a text or program file and you want to leave to check the calendar for the current month. To view the calendar from command mode, type *:!cal* (a colon, an exclamation point, and the command) and press Enter. This action executes the *cal* command, and when you press Enter again, you go back into your vi editing session. Using *:!* plus a command-line command enables you to start a new shell, run the command, and then go back into the vi editor.

When you want to run several command-line commands in a different shell without first closing your vi session, use the *Ctrl+z* option to display the command line. (See Figure 3-4.) When you finish executing commands, type *fg* to go back into your vi editing session.


NOTE

Using *Ctrl+z* in this context is really a function of the Bash shell, which in this example leaves the vi editor running in the background and takes you to the shell command line. When you enter *fg*, this is a shell command that brings the job you left (the vi editing session) back to the foreground.

Hands-on Project 3-10 enables you to use the *:!* and *Ctrl+z* commands from a vi editing session.


TIP

You can set up a script file (a file of commands) that automatically runs when you launch vi. The file is called .exrc and is a hidden file located in your home directory. This file can be used to automatically set up your vi environment. Programmers, for example, who want to view line numbers in every editing session might create an .exrc file and include the *set number* command in the file. To learn more about scripts, see Chapters 6 and 7 ("Introduction to Shell Script Programming" and "Advanced Shell Programming").

Changing Your Display While Editing

Besides using the vi editing commands, you can also set options in vi to control editing parameters, such as using a line number display. Turn on line numbering when you want to work with a range of lines, for example, when you're deleting or cutting and pasting blocks of text. Then, you can refer to the line numbers to specify the text.

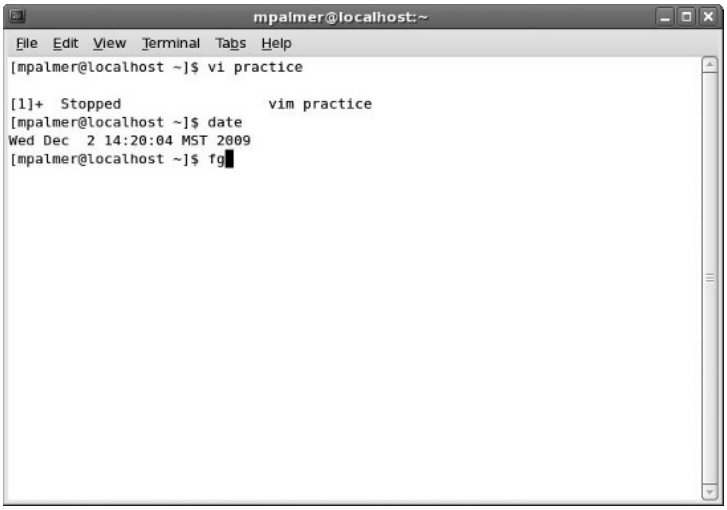

Figure 3-4 Accessing a shell command line from the vi editor

To turn on line numbering, use the *:set number* command. Then, if you want to delete lines 4 through 6, for example, it is easy to determine that these are the lines you intend to delete, and you simply use the *:4,6d* command to delete them. Try Hands-on Project 3-11 to turn on line numbering and then refer to it for deleting text.

Copying or Cutting and Pasting

You can use the *yy* command in vi to copy a specified number of lines from a file and place them on the clipboard. To cut the lines from the file and store them on the clipboard, use the *dd* command. After you use the *dd* or *yy* commands, you can use the *p* command to paste the contents in the clipboard to another location in the file. These commands are handy if you want to copy text you already typed and paste it in another location. Hands-on Project 3-12 enables you to cut text and paste it in another location within the same file.

Printing Text Files

Sometimes you want to print a file before you exit the vi editor. You can use the *lpr* (line print) shell command to print a file from vi. Type *!lpr* and then type the name of the file you want to print. Hands-on Project 3-13 enables you to print a file on which you are working in the vi editor.

You can also specify which printer you want to use via the *-P printer* option, where *printer* is the name of the printer you want to use. For example, you might have two printers, lp1 and lp2. To print the file accounts to lp2, enter *:!lpr -P lp2 accounts* and press Enter.

Syntax **lpr** [-option] [*filename*]

Dissection

- The argument consists of the name of the file to print.
- Options include:

  -P specifies the destination printer; you include the name of the printer just after the option.

  -# specifies the number of copies to print (up to 100 copies).

  -r deletes a print file after it is printed (saving disk space).

Canceling an Editing Session

If necessary, you can cancel an editing session and discard all the changes you made in case you change your mind. Another option is to save only the changes you made since last using the *:w* command to save a file without exiting vi. In Hands-on Project 3-14, you exit a file without saving your last change.

Getting Help in vi

You can get help in using vi at any time you are in this editor. To access the online help documentation while you are editing a file, use the *help* command. You can access help documentation after you start the vi editor by pressing Esc, then typing a colon (:), and then *help*. You access the help documentation in Hands-on Project 3-14.

You can also view documentation about vi using the *man vi* command. While you are in vi, press Esc, type *:!man vi*, and press Enter (type *q* and press Enter to go back into your editing session). Or, when you are at the shell command line and not in a vi session, type *man vi* and press Enter.

Using the Emacs Editor

Emacs is another popular UNIX/Linux text editor. Unlike vi, Emacs is not modal. It does not switch from command mode to insert mode. This means that you can type a command without verifying that you are in the proper mode. Although Emacs is more complex than vi, it is more consistent. For example, you can enter most commands by pressing Alt or Ctrl key combinations.

Emacs also supports a sophisticated macro language. A **macro** is a set of commands that automates a complex task. Think of a macro as a "superinstruction." Emacs has a powerful command syntax and is extensible. Its packaged set of customized macros lets you read electronic mail and news and edit the contents of directories. You can start learning Emacs by learning its