The vi Editor: First Look

Chapter 4 is the first of two chapters that discuss the UNIX operating system vi editor (*vi* is pronounced *vee-eye*); Chapter 6 is the second. Chapter 4 starts with an explanation of editors in general and then discusses the types of editors and their applications. After a brief description of the editors supported by UNIX, the chapter introduces the vi editor. The rest of the chapter presents the basic commands necessary to do a simple editing job in the vi editor. Basic concepts and operations with the vi editor are explained:

- The different modes of the vi editor
- Memory buffers
- The process of opening a file for editing
- Saving a file
- Quitting vi

In This Chapter

4.1 WHAT IS AN EDITOR?

Editing a text file is one of the most frequently performed computer operations. In fact, most of the things you will want to do will require some sort of file editing sooner or later.

An editor (text editor) is a tool that facilitates the creation of files or modification of existing files. Files may contain notes, memos, program source code, and so on. An editor is a watered down, vanilla sort of word processor. It does not have the typographical features (bold, center, underline, etc.) that word processors do.

At least one editor program is supplied with the software of every operating system. There are two general types of editors:

- Line editors
- Full screen editors

The Line Editor In a *line editor,* most of the changes are applied to only one line or group of lines at a time. To make a change, you must first specify the line number in the text, and then you must specify the change itself. Line editors are usually difficult to use because you cannot see the scope and context of your editing task. Line editors are good for global operations like searching, replacing, and copying large blocks of text in your file.

The Screen Editor A *screen editor* displays the text that you are editing one screen at a time and allows you to move the cursor around the screen and make changes. Any changes you make are applied to the file, and you get an immediate feedback on the screen. You can easily view the rest of the text one screen at a time. Screen editors are more user friendly than line editors and are preferred for everyday editing jobs.

4.1.1 UNIX-Supported Editors

The UNIX operating system supports a number of both line and screen editors, so you can create or alter a file easily and efficiently. To name a few, *emacs* and *ex* are line editors, and *vi* is a screen editor under UNIX.

A line editor called *ed* is an old editor supplied with the early versions of the UNIX operating systems. Currently, the ex family of editors is supplied with most UNIX operating systems. The ex editor originally provided a display facility that showed and let you work with a screen of text instead of one line at a time. In order to use this capability, you had to give ex the **vi** (for *visual*) command. However, the use of the visual mode of the ex editor became so popular that the developers of the ex editor now provide a separate vi editor. This means you can use vi without having to start the ex editor.

The Text Formatter The UNIX editors are not text-formatting programs. They do not provide functions like centering a line or setting up margins, things that are readily available with any word processing program. To format text, UNIX supports utilities like *nroff* and *troff.*

See page x for an explanation of the icons used to highlight information in this chapter.

Text formatters are usually used to prepare documents. Input to these programs are text files that you create using editors like the vi editor. Output from text formatters is paginated, and you can display the formatted text on the screen or send it to the printer.

Table 4.1 shows some of the UNIX operating system editors and their categories.

Table 4.1
Some of the Editors Supported by UNIX

Editor	Category
ed	The original line editor
ex	A more sophisticated superset of the ed editor
vi	A visual screen editor
emacs	A public domain screen editor

4.2 THE vi EDITOR

The vi editor is a screen-oriented text editor available on most UNIX operating systems, and it provides some of the flexibility and ease of a word processing program. Because vi is based on the line editor ex, it is possible to use ex commands from within vi. When you use vi, changes to your file are reflected on the terminal screen, and the position of the cursor on the screen indicates the position within your file. More than 100 vi commands are available, providing many capabilities—and also the challenge of learning them! Do not panic. Only a few of the commands are necessary to do a simple editing task.

Two versions of vi, the *view editor* and the *vedit editor,* are tailored for specific tasks or users. These versions are the same as the vi editor except that certain flags (options) are preset. Not every system has these versions on it. By the end of the two vi chapters in this book, you will have learned to customize the vi environment according to your own needs.

The view Editor The *view editor* is the version of the vi editor with the read-only flag set in it. The view editor is useful when you only want to look at the contents of your file and not modify it. The view editor prevents you from inadvertently changing your file, and so your file remains intact.

The vedit Editor The *vedit editor* is a version of the vi editor with several flags set. The vedit editor is intended for beginners. Its flag settings make it easier to learn how to use vi.

4.2.1 The vi Modes of Operation

The vi editor has two basic modes of operation: command (edit) mode and text input mode. Some commands are applicable only to the command mode, and others work only in the text input mode. During a vi editing session, you change from one mode of vi to the other to do the editing job.

Command Mode When you start the vi editor, it comes up in the command mode. In command mode, key entries (any key or sequence of keys you press) are interpreted as commands. The keys are not echoed on the screen, but the commands associated with the pressed key or keys are carried out. While the vi editor is in the command mode, you can delete lines, search for a word, move the cursor on the screen, and perform a number of other useful operations by pressing a key or sequence of keys.

Certain commands in the command mode start with the colon **(:)**, forward slash **(/)**, or question mark **(?)**. The vi editor displays these commands on the last line on the screen. To signal the end of the command line for these types of commands, you press **[Return]**.

Text Input Mode In the text input mode, the keyboard becomes your typewriter. vi displays any key or sequence of the keys that you press, and input is interpreted not as commands but as text that you want to write into your file.

Status Line The bottom line on the screen, usually line 24, is used by vi to give you some feedback about your editing operation. Error messages and other informative messages are displayed on the status line. vi also displays the commands that start with **:**, **/**, or **?** on line 24.

4.3 BASIC vi EDITOR COMMANDS

The basic editing job usually involves the following operations:

- Creating a new file or modifying an existing file (open file operation)
- Entering text
- Deleting text
- Searching text
- Changing text
- Saving the file and quitting the editing session (close file operation)

The following sections walk you through a few editing sessions, explore the vi editor's way of working, and show you some of the necessary editing commands in both text input mode and command mode.

Relying solely on the textbook to learn the vi editor (or any other editor) is not a good idea. It is highly recommended that, after an initial reading of this chapter, you practice the examples and terminal session exercises on your system.

Assumptions In order not to repeat the setup of examples in every section, the general assumptions that are applicable to all examples are summarized here:

- The *current line* is the line that the cursor is on.
- The double underline (=) indicates the position of the cursor on the line.
- The `myfirst` file is used to show the operations of different keys and commands. When necessary, examples are illustrated on screens. If there are multiple screens, the top one shows the current state of the text, and the subsequent ones show the changes on the screen after the specified keys or commands are applied to the text.
- To save space, only the relevant part of the screen is shown.
- The examples are not continuous; that is, the editing changes are not carried over from one example to another.

4.3.1 Access to the vi Editor

As with any other software program, the first step with vi is to learn how to start and end the program. This section shows how to invoke the vi editor to create a small text file and then save the file.

Starting vi

To start the vi editor, type **vi**, press **[Spacebar]**, type the name of the file (in this example, **myfirst**), and then press **[Return]**.

If a `myfirst` file already exists, vi displays the first page (23 lines) of the file on the screen. If `myfirst` is a new file, vi clears the screen, shows the vi blank screen, and positions the cursor at the upper-left corner of the screen.The screen is filled with tildes *(~)* in the first column. The vi editor is in the command mode and is ready to accept your commands. Figure 4.1 shows the vi editor blank screen.

Figure 4.1
The vi Editor Blank Screen

```
~
~
~
~
"myfirst" [new file] 0 lines, 0 characters
```

1. *The status line shows the filename and the fact that it is a new file.*
2. *To save space, screens shown in this book are not shown in full size (24 lines).*

In order to input your text, you must have the vi editor in the text input mode. Make sure **[Caps Lock]** is off, and then press **i** (for insert). vi does not display the letter *i* but enters into the text input mode. Now type the lines shown in Figure 4.2. The vi editor displays everything you type on the screen. Use **[Backspace]** (or **[Ctrl-h]**) to erase characters, and press **[Return]** at the end of each line to go to the next line. Do not be overly concerned about the syntax and spelling of the text. The goal here is to create a file and save it.

Figure 4.2
The vi Editor Display Screen

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
~
~
~
~
"myfirst" [new file]
```

Make sure [Caps Lock] is off because uppercase and lowercase letters have different meanings in the command mode.

If your file does not fill the whole screen, the vi editor fills the first column of the remaining lines with tildes (~).

Ending vi

In order to save a file that you have created or edited with vi, you must first place vi in command mode. To do so, press **[Esc]**. If the sound on your terminal is activated, you hear a beep that indicates vi is in the command mode. The **save file** and **quit** vi commands are both commands that start with a colon **(:)**. Press **:** to put the cursor on the last line of the terminal screen. Then type **wq** (for *write* and *quit*) and press **[Return]**. The vi editor saves your file (which you called "myfirst") and passes control back to the shell. The shell displays the dollar sign prompt. The **$** signals that you are out of the vi editor and back to the shell and the system is ready for your next command. Figure 4.3 shows the vi editor display after you enter **: wq**.

Figure 4.3
The vi Editor Screen After the **wq** Command

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
~
~
:wq
"myfirst" [new file] 3L, 112C written
$_
```

The vi editor feedback appears on the last line of the screen. It shows the filename followed by the number of lines and number of characters in the file.

4.3.2 Cursor Movement Keys: First Look

In order to accomplish the next terminal sessions, you need to be able to change the mode of the vi editor to command mode or text input mode and to be able to position the cursor on a specific location on the screen. The following section explains the subset of the keys in command mode that are needed to do these operations.

When you start the vi editor, it enters the command mode. Pressing the **[Esc]** key will place vi in command mode regardless of which mode vi was in. Table 4.2 is a partial list of the cursor movement keys. These keys are used to position the cursor. Later in this chapter these and additional cursor movement keys will be explained and practiced.

Table 4.2
Partial List of the Cursor Movement Keys

Key	Operation
h or **[Left Arrow]**	Moves the cursor position one space to the left.
j or **[Down Arrow]**	Moves the cursor position one line down.
k or **[Up Arrow]**	Moves the cursor position one line up.
l or **[Right Arrow]**	Moves the cursor position one space to the right.

1. **Remember, vi must be in command mode when you want to use the cursor movement keys. If in doubt, press the [Esc] key before using these keys.**
2. **Make sure [Caps Lock] is off, because uppercase and lowercase letters have different meanings in the command mode.**
3. **While the vi editor is in the command mode, most commands are initiated as soon as you press the key. There is no need to use [Return] to indicate the end of a command line.**

4.3.3 Text Input Mode

You must be in the text input mode in order to type text into your file. However, there are several different commands you can use to change to text input mode, and each change mode key enters the vi text input mode in a slightly different manner. The location of the text in your file depends on the position of the cursor on the screen and the key you choose to place vi in the text input mode.

Table 4.3 summarizes the keys that change the vi editor from the command mode to the text input mode. For example, let's use `myfirst`, the file created in the previous section. Suppose you have it on the screen and want to enter **999** in your file. (There is nothing special about 999. It is merely being used to illustrate the location of the text you enter relative to the position of the cursor on the screen. You can type anything you

Table 4.3
The vi Change Mode Keys

Key	Operation
i	Inserts the text you enter before the character that the cursor is on.
I	Places the text you enter at the beginning of the current line.
a	Appends the text you enter after the character that the cursor is on.
A	Places the text you enter after the last character of the current line.
o	Opens a blank line below the current line and places the cursor at the beginning of the new line.
O	Opens a blank line above the current line and places the cursor at the beginning of the new line.

wish.) Also assume the cursor is on the letter *m* of the word *most*, as indicated by the double underline. Depending on what key you choose to enter the text input mode, the 999 that you type will be put in a different location in your file.

Normally, when you enter into text input mode by using any of the change mode keys, vi does not provide feedback or a confirmation message to indicate that you are indeed in text input mode. The vi editor can be tailored to provide feedback and indicate which of the vi modes is in use. If words such as *insert*, *open*, *append*, and so on appear on the screen indicating the mode vi is in, then the vi editor on your system has been tailored to show feedback. You will learn how to tailor the vi editor to your preference in Chapter 6.

1. *The cursor position is identified with a double underline (_).*
2. *The current line is a line that has the cursor placed anywhere in it.*

For the terminal sessions in the following sections, you need to open the `myfirst` file. Open the `myfirst` file by using the following command:

```
$ vi myfirst [Return]  . . . . . Open myfirst file
```

If you have not created a `myfirst` file in the previous section, you can create it now by typing the text from Figure 4.1. Before starting each terminal session, make sure the cursor is on the designated letter. Use the arrow keys on the keyboard (not the ones on the number pad) to place the cursor on the desired letter. Your UNIX system might not recognize the arrow keys for moving the cursor. In this case, use the **h**, **j**, **k**, or **l** keys to move and position the cursor.

Also, for each of the following terminal sessions, you have to close the file and end vi without saving the changes in the file. You want to use the original `myfirst` file for your next terminal sessions. Use the following key sequence to close the `myfirst` file:

- Press **[Esc]**. Just to make sure vi is in command mode.
- Press : to place the prompt on the status line at the bottom of the screen.
- Press **q! [Return].** This command quits vi without saving the changes. The **[!]** key indicates you are sure you want to abandon the changes.

These steps for opening and closing the `myfirst` *file at the beginning and end of each part of the vi terminal sessions might seem excessive, but the goal is to have the original* `myfirst` *file for each exercise and to make the exercises independent from each other. If you are comfortable having the results of each of your practice sessions in the same file, then you can skip these steps opening and closing the file for each exercise.*

Inserting Text: Using i or I

Pressing either **i** or **I** places the vi editor in text input mode. The choice you make, however, puts the text you enter in a different place in the file. Pressing **i** places the text you enter before the cursor position, but pressing **I** places your text at the beginning of the current line.

Do the following things to experiment with **i**:

- Type **vi myfile [Return]**. This opens the `myfile` file.
- Press **[Esc]**. Just to make sure vi is in the command mode.
- Use the cursor movement keys to place the cursor on the letter *m* of *most.*

- Press **i**. vi enters the text input mode.
- Press **9** three times. 999 appears before the *m.*

The cursor remains on the letter *m,* and the vi editor remains in the text input mode until you press **[Esc]** to return to the command mode.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by 999most of the UNIX operating systems.
```

Do the following things to experiment with **I:**

- Press **[Esc]** to change vi to the command mode.
- Use the cursor movement keys to place the cursor on the letter *s* of *supported.*
- Press **I.** vi enters the text input mode and moves the cursor to the beginning of the current line.
- Press **9** three times. 999 appears at the beginning of the current line, and the cursor moves to the *T.*

```
The vi history
The vi editor is an interactive text editor that is
supported by 999most of the UNIX operating systems.
```

```
The vi history
999The vi editor is an interactive text editor that is
supported by 999most of the UNIX operating systems.
```

The cursor remains on the *T,* and the vi editor remains in the text input mode until you press **[Esc]** to return to the command mode.

- Press **[Esc]** to change vi to the command mode.
- Type **:q! [Return]**. This quits vi without saving your changes.

Adding Text: Using a or A

Pressing either **a** or **A** places the vi editor in text input mode. The choice you make, however, puts the text you enter in a different place in the file. Pressing **a** places the text you enter after the cursor position at the time you pressed **a**, and pressing **A** adds the text you enter to the end of the current line.

Do the following things to experiment with **a:**

- ❏ Type **vi myfile [Return]**. This opens the `myfile` file.
- ❏ Press **[Esc]** just to make sure vi is in the command mode.
- ❏ Use the cursor movement keys to place the cursor on the letter *m* of *most*.
- ❏ Press **a**. vi enters the text input mode, and the cursor moves to the *o*, the next letter on the right.
- ❏ Press **9** three times. 999 appears after the *m*.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by m999ost of the UNIX operating systems.
```

The cursor remains on the o, and the vi editor is in the text input mode until you press **[Esc]** to return to the command mode.

Do the following things to experiment with **A**:

- ❏ Press **[Esc]** to change vi to the command mode.
- ❏ Use the cursor movement keys to place the cursor on the letter *o* of *most.*
- ❏ Press **A**. The vi editor enters the text input mode, and the cursor moves to the end of the current line.
- ❏ Press **9** three times. 999 appears after the . (period), the last character on the current line.

```
The vi history
The vi editor is an interactive text editor that is
supported by m999ost of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by m999ost of the UNIX operating systems.999
```

The cursor moves to the end of the line, and the vi editor remains in the text input mode until you press **[Esc]** to return to the command mode.

- ❏ Press **[Esc]** to change vi to the command mode.
- ❏ Type **:q! [Return]**. This quits vi without saving your changes.

Opening a Line: Using o or O

Pressing either **o** or **O** places the vi editor in text input mode. Pressing **o** opens a blank line below the current line, and pressing **O** opens a blank line above the current line.

Do the following things to experiment with **o**:

- Type **vi myfile [Return]**. This opens the `myfile` file.
- Press **[Esc]** just to make sure vi is in the command mode.
- Use the cursor movement keys to place the cursor on the letter *i* of *is*.
- Press **o**. The vi editor enters the text input mode, opens a line below the current line, and moves the cursor to the beginning of the new line.
- Press **9** three times. 999 appears on the new line.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
999
supported by most of the UNIX operating system.
```

The cursor moves to the end of the new line, and the vi editor remains in the text input mode until you press **[Esc]** to return to the command mode.

Do the following things to experiment with **O**:

- Press **[Esc]** to change vi to the command mode.
- Use the cursor movement keys to place the cursor on the letter *i* of *is*.
- Press **O**. The vi editor enters the text input mode, opens a line above the current line, and moves the cursor to the beginning of the new line.
- Press **9** three times. 999 appears on the new line.

```
The vi history
The vi editor is an interactive text editor that is
999
supported by most of the UNIX operating systems.
```

```
The vi history
999
The vi editor is an interactive text editor that is
999
supported by most of the UNIX operating systems.
```

The cursor moves to the end of the new line, and the vi editor remains in the text input mode until you press **[Esc]** to return to the command mode.

❏ Press **[Esc]** to change vi to the command mode.

❏ Type **:q! [Return]**. This quits vi without saving your changes.

Avoid the use of the cursor keys (arrow keys) in the text input mode. On some systems, the cursor keys are interpreted as regular ASCII characters, and their ASCII codes will be inserted into your file if they are used in text input mode.

Using [Spacebar], [Tab], [Backspace], and [Return]

While in the text input mode, the vi editor displays the letters you type on the screen, but not all keys on the keyboard produce a displayable character. For example, when you press **[Return]** you intend to move the cursor to the next line, and you do not expect to see a **[Return]** symbol on the screen. As you know by now, depending on the vi editor's mode, keys have different meanings. Consider the use of these keys in the text input mode.

[Spacebar] and [Tab] Pressing **[Spacebar]** always produces a space character before the cursor position. Pressing **[Tab]** usually produces eight spaces. The tab size is changeable and can be set for any number of spaces. (See Chapter 6.)

[Backspace] Pressing **[Backspace]** moves the cursor one character to the left over the currently typed text.

[Return] Pressing **[Return]** when you are in text input mode always opens a new line. Depending on the position of the cursor on the current line, it opens a line above or below the current line:

- If the cursor is at the end of the line or there is no text to the right of it, then pressing **[Return]** opens an empty line below the current line.
- If the cursor is on the very first character of the current line, then pressing **[Return]** opens an empty line above the current line.
- If the cursor is somewhere on the line with text to the right of it, then pressing **[Return]** splits the line, and the text to the right of the cursor moves to the new line.

This explanation applies only when the vi editor is in the text input mode.

Do the following things to experiment with the **[Spacebar]**, **[Tab]**, and **[Return]** keys:

❏ Type **vi myfile [Return]**. This opens the `myfile` file.

❏ Press **i**. vi enters the text mode.

❏ Type something and practice using the **[Spacebar]** key. Observe the results.

❏ Practice using the **[Tab]** key and observe the results.

❏ Practice using the **[Return]** key and observe the results.

❏ Press **[Esc]** to change vi to the command mode.

❏ Type **:q! [Return]**. This quits vi without saving your changes.

4.3.4 Command Mode

When you start the vi editor, it enters the command mode. If vi is in the text input mode and you want to change to command mode, press **[Esc]**. If you press **[Esc]** while the vi editor is in the command mode, nothing happens, and vi remains in the command mode.

Cursor Movement Keys

In order to delete text, correct text, or insert text, you need to move the cursor to a specific location on the screen. While in command mode, you use the arrow keys (cursor control keys) to move the cursor around on the screen. On some terminals, the arrow keys do not work as explained or are not available. In these cases, you can use the **h**, **j**, **k**, and **l** letter keys to move the cursor left, down, up, and right, respectively. Pressing any of the cursor movement keys moves the cursor position one space, one word, or one line at a time.

Table 4.4 summarizes the cursor movement keys and their applications. Each key application is explained in the examples and the terminal sessions. Figure 4.4 shows the effect of these cursor movement keys on the screen.

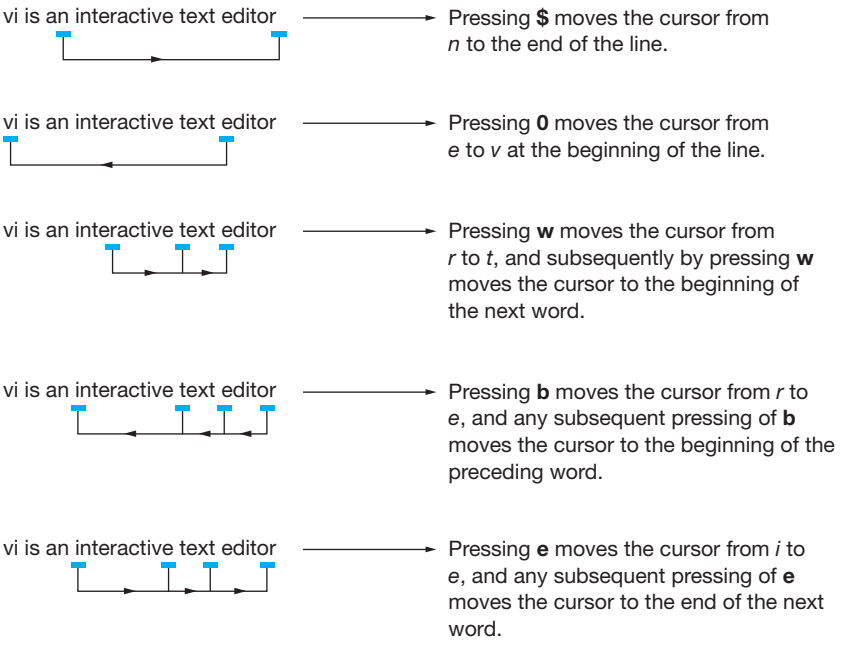
Table 4.4
vi's Cursor Movement Keys

Key	Operation
h or **[Left Arrow]**	Moves the cursor position one space to the left.
j or **[Down Arrow]**	Moves the cursor position one line down.
k or **[Up Arrow]**	Moves the cursor position one line up.
l or **[Right Arrow]**	Moves the cursor position one space to the right.
$	Moves the cursor position to the end of the current line.
w	Moves the cursor position forward one word.
b	Moves the cursor position back one word.
e	Moves the cursor position to the end of the word.
0 (zero)	Moves the cursor position to the beginning of the current line.
[Return]	Moves the cursor position to the beginning of the next line.
[Spacebar]	Moves the cursor position one space to the right.
[Backspace]	Moves the cursor position one space to the left.

j and k The **j** or **[Down Arrow]** key moves the cursor one line down to the same position as it was on the previous line. The **k** or **[Up Arrow]** key moves the cursor one line up. When there is no line below the current line (end of the file), or the current line is the first line (top of the file), then you hear a beep, and the cursor remains on the current line. These keys do not cause the text to wrap around.

Figure 4.4

The Movement of the Cursor on the Screen



h and l Each time you press the **h** or **[Left Arrow]** key, the cursor moves one character to the left until it is on the first character of the current line. Then it beeps, indicating it cannot move farther to the left. The **l** (lowercase *L*) or **[Right Arrow]** key moves the cursor to the right in a similar manner. These keys do not cause text to wrap around.

$ and 0 Pressing **$** moves the cursor to the end of the current line. This key cannot be pressed more than once: When the cursor is at the end of the line, pressing **$** does not change anything. Pressing **0** (zero) moves the cursor to the beginning of the current line in a similar manner.

w, b, and e Each time you press **w**, the cursor moves to the beginning of the next word. Pressing **b** moves the cursor left (back) to the beginning of the preceding word, and pressing **e** moves the cursor to the end of the word. These keys cause the text to wrap around and, if necessary, move the cursor to the next line.

[Return] Each time you press **[Return]** in command mode, the cursor moves to the beginning of the next line below the current line until you reach the end of the file.

Do the following things to experiment with the cursor movement keys:

- ❑ Type **vi myfile [Return]**. Opens `myfile` file.
- ❑ Move the cursor to the left using **h** or **[Left Arrow]**. Observe that the cursor moves over each character and stops at the beginning of the line.
- ❑ Move the cursor to the right using **l** (ell) or **[Right Arrow]**. Observe that the cursor moves over each character and stops at the end of the line.
- ❑ Move the cursor to the top of the file using the **k** or **[Up Arrow]**. Observe that the cursor moves over each line and stops on the first line of the file.
- ❑ Move the cursor to the bottom of the file using the **j** or **[Down Arrow]**. Observe that the cursor moves over each line and stops on the last line of the file.

- ❑ Press **$**. Observe that the cursor moves to the end of the current line.
- ❑ Press **0**. Observe that the cursor moves to the beginning of the current line.
- ❑ Press **w**. Observe that the cursor moves forward one word.
- ❑ Press **b**. Observe that the cursor moves backward one word.
- ❑ Press **e**. Observe that the cursor moves to the end of the current word.
- ❑ Press **[Return]**. Observe that the cursor moves to beginning of the next line.
- ❑ Press **[Spacebar]**. Observe that the cursor moves one position to the right.
- ❑ Press **[Backspace]**. Observe that the cursor moves one position to the left.
- ❑ Practice using the cursor movement keys until you feel comfortable using them or you get tired of using them, whichever comes first.
- ❑ Press **[Esc]** to change vi to the command mode.
- ❑ Type **:q! [Return]**. This quits vi without saving your changes.

Text Correction

While in the command mode, you can replace (overwrite) characters and delete characters, a line, or a number of lines. You can also correct some of your mistakes by using the **undo** command. As the name implies, this disregards your most recent command. These text-correcting commands apply only when vi is in the command mode, and most of them do not change the vi mode. Table 4.5 summarizes the text-correction keys and their applications.

Table 4.5
The vi Editor Keys for Correcting Text in Command Mode.

Key	Option
x	Deletes the character specified by the cursor position.
dd	Deletes the line specified by the cursor position.
u	Undoes the most recent change.
U	Undoes all the changes on the current line.
r	Replaces a character that the cursor is on.
R	Replaces characters starting from the cursor position. Also changes vi to the text input mode.
. (dot)	Repeats the last text changes.

Character Deletion: Using x

Suppose you have the `myfirst` file on the screen and want to correct some text in the file. The cursor is positioned on the letter *m* of the word *most.* Using **x**, you can delete characters starting from the cursor position:

- ❑ Type **vi myfile [Return]**. This opens the `myfile` file.
- ❑ Press **[Esc]** to make sure vi is in command mode.

❑ Use the cursor movement keys to place the cursor on the letter *m* of *most*.

❑ Press **x**. The vi editor deletes the *m,* and the cursor moves to *o,* the next letter to the right. The vi editor remains in command mode.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by ost of the UNIX operating systems.
```

❑ Press **x** three more times. The vi editor deletes *o*, *s*, and *t*, respectively, as you press **x** repeatedly.

```
The vi history
The vi editor is an interactive text editor that is
supported by _of the UNIX operating systems.
```

The vi editor remains in the command mode and the cursor moves to the space before the *o*. If you want to delete more than one character in a command, you can use the **nx** command, where *n* is an integer immediately followed by the letter *x*. For example, the command **5x** deletes five characters starting from the cursor position.

❑ Press **5x**. The vi editor deletes five characters and the cursor moves to the letter *h.*

```
The vi history
The vi editor is an interactive text editor that is
supported by he UNIX operating systems.
```

❑ Press **[Esc]** to make sure vi is in the command mode.

❑ Type **:q! [Return]**. This quits vi without saving your changes.

Repetition can be used with other vi commands. For example, ***dd*** *deletes one line, and* ***3dd*** *deletes three lines.*

Deletion and Recovery: Using dd and u

By pressing **d** twice, you can delete a line, starting from the current line. To learn about using **dd**, do the following:

❑ Type **vi myfile [Return]**. This opens the `myfile` file.

- Press **[Esc]** to make sure vi is in command mode.
- Use the cursor movement keys to place the cursor on the letter *m* of *most.*

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

- Press **d** twice. The vi editor deletes the current line, regardless of the cursor position on the line.

```
The vi history
The vi editor is an interactive text editor that is
supported
```

- The vi editor is in the command mode, and the cursor moves to the beginning of the next line. Press **u**, and the vi editor undoes your last delete.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

The vi editor remains in the command mode, and the cursor moves to the beginning of the line. If you want to delete more than one line in a command, you can use the **ndd** command, where *n* is an integer followed immediately by two presses of **d**. For example, the command **3dd** deletes three lines starting from the current line.

- Use the cursor movement keys to place the cursor on the letter *T* of *The* on the first line.
- Press **3dd**. This deletes three lines starting from the current line. The vi editor remains in the command mode.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
=
~
~
```

- Press **u**. vi undoes your last delete command, and the deleted lines appear again in your file.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

- Practice using the **dd** and **u** commands.
- Press **[Esc]** to make sure vi is in the command mode.
- Type **:q! [Return]**. This quits vi without saving your changes.

Text Replacement: Using r, R, and U

Using **r** or **R**, you can replace a character or a group of characters, starting from the cursor position. However, pressing **R** puts the vi editor in the text input mode, and you must press **[Esc]** to return to the command mode.

To learn about **r**, do the following:

- Type **vi myfile [Return]**. This opens the `myfile` file.
- Press **[Esc]** to make sure vi is in command mode.
- Use the cursor movement keys to place the cursor on the letter *m* of *most*.
- Press **r** to replace (overwrite) the character that the cursor is on.
- Press **9**. The vi editor responds by changing the *m* to *9*. The vi editor remains in the command mode, and the cursor stays on the same position.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by 9ost of the UNIX operating systems.
```

To learn about **R** and **U**, do the following:

- Press **R** to replace the characters starting from the cursor position. The vi editor enters the text input mode.
- Press **9** three times. The vi editor responds by adding 999 after the cursor position, overwriting *ost*. The vi editor remains in the text input mode.

```
The vi history
The vi editor is an interactive text editor that is
supported by 9999 of the UNIX operating systems.
```

- Press **[Esc]** to change to the command mode.
- Press **U** to undo all your changes on the current line.

The vi editor responds by restoring the current line to its previous state.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

- Practice, using the **r**, **R**, and **u**, **U** commands.
- Press **[Esc]** to make sure vi is in the command mode.
- Type **:q! [Return]**. This quits vi without saving your changes.

Pattern Search: Using / and ?

The vi editor provides operators to search a file for a specified pattern. The forward slash (**/**) and question mark (**?**) are the keys used to search forward and backward, respectively, through your file. If you are editing a large file, you also can use these operators to position the cursor at a specific place in the file. For example, if you want to look for the word *UNIX* in your file, press **[Esc]** (to make sure vi is in command mode) and then type **/UNIX** and press **[Return]**.

When you press **/**, vi shows the **/** at the bottom of the screen and waits for the rest of the command. After you press **[Return]**, the vi editor starts searching forward for the character string *UNIX* from the current position of the cursor. If the character string *UNIX* is in your file, vi positions the cursor on its first occurrence.

You can move the cursor to the next occurrence of *UNIX* (or whatever word you are looking for) by pressing **n** (for next). Every time you press **n**, vi shows the next occurrence of the pattern you are searching for, until the vi editor reaches the end of the file. Then it goes back to the beginning of the file and continues the process.

If you prefer to search backward through the file, type **?UNIX** and press **[Return]**. The vi editor continues to search backward as long as you press **n** after each finding of the word *UNIX*.

Repeating the Previous Change: Using . (dot)

The . **(dot)** key is used in the command mode to repeat the most recent previous text change. This feature is quite useful when you want to do a lot of repetitive changes in a file.

To experiment with **. (dot)**, try the following:

- Type **vi myfile [Return]**. This opens the `myfile` file.
- Press **[Esc]** to make sure vi is in command mode.
- Use the cursor movement keys to place the cursor on the letter *m* of *most*.
- Press **dd** to delete the current line. The cursor moves to the beginning of the line above.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported
```

- Press **. (dot)**. The vi editor repeats the previous text change and deletes the current line. The cursor moves to the beginning of the line above, and vi remains in the command mode.

```
The vi history
~
~
~
```

- Practice, using the **.** and **u**, **U** commands.
- Press **[Esc]** to make sure vi is in the command mode.
- Type **:q! [Return]**. This quits vi without saving your changes.

Exiting the vi Editor

There is only one way to enter vi, but there are several ways to exit it. The vi editor gives you a choice, depending on what you intend to do with your file after editing. Table 4.6 summarizes the vi editor quit commands.

Table 4.6
The vi Editor Quit Commands

Key	Operation
wq	Writes (saves) the contents of the file and quits the vi editor.
w	Writes (saves) the contents of the file but stays in the editor.
q	Quits the editor.
q!	Quits the editor and abandons the contents of the file.
ZZ	Writes (saves) the contents of the file and quits the vi editor.

The :wq Command Most of the time, you use the **:wq** command at the end of an editing session. This saves the file and exits the vi editor. UNIX displays the **$** prompt, indicating that the shell has returned. The **ZZ** command (uppercase *z*) works just the same: It saves your file and quits the vi editor.

The :q Command You use the **:q** command to look at the contents of a file without doing any editing and exit the vi editor. However, if you have changed something in your file and use the command to exit, vi responds by showing the following message at the bottom line on the screen (a typically terse UNIX message) and the vi editor remains on the screen:

```
No write since last change (:q! overrides).
```

The :q! Command If you change something in a file and then decide not to save the changes, use the **:q!** command to leave the vi editor. In this case, the original file remains intact and changes are abandoned.

The :w Command Use the **:w** command to save your file periodically during the course of a long editing session and to prevent losing your work accidentally. The **:w** command will accept a new filename to save your changes in a new file, if you do not want to write over the original file.

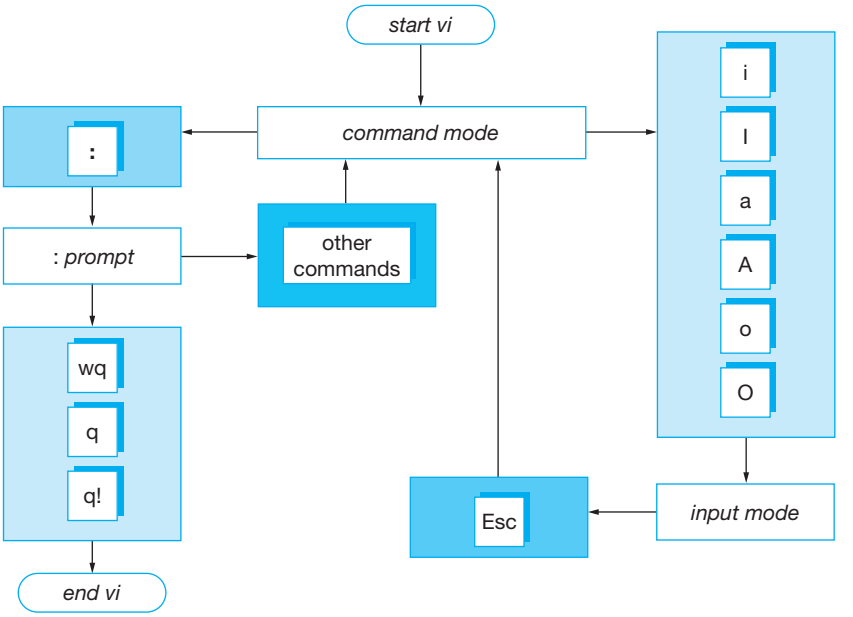
The ZZ Command Use the **ZZ** (uppercase *z*) command to quickly save the file and exit the vi editor.

The **ZZ** *command is not preceded by* **:** *and you do not press* **[Return]** *to complete the command. Just type* **ZZ**, *and the job is done.*

Figure 4.5 shows the vi editor modes of operation and the key sequences or commands that change the vi editor from one mode to the other.

Figure 4.5

The vi Editor Modes of Operation



1. **Most of the commands just discussed start with :.**
2. **Pressing : positions the cursor on the last line on the screen. Then vi displays any key you press to complete the command on the same line.**
3. **Remember to press [Return] to signal the completion of a command.**

4.3.5 Linux: vi Online Help

Linux supports an enhanced version of vi called vim (vi improved). This version is upwardly compatible with vi and provides some extra command and features. One of these features is an online help feature that is described here.

In command mode, you type **:help [Return]**. This will show a general help command description similar to that shown in Figure 4.6.

Figure 4.6
vim Help Screen

```
*help.txt*
              VIM - main help file
                                                                    k
     Move around: Use the cursor keys, or "h" to go left,         h  I
                  "j" to go down, "k" to go up, "l" to go right.   j
 Close this windo: Use ":q<Enter>".
   Get out of Vim: Use ":qa!<Enter>" (careful, all changes are lost!).
Jump to a subject: Position the cursor on a tag between |bars| and hit CTRL-].
   With the mouse: ":set mouse=a" to enable the mouse (in xterm or GUI).
                   Double-click the left mouse button on a tag between |bars|.
       jump back: Type CTRL-T or CTRL-O.
Get specific help: It is possible to go directly to whatever you want help
                   on, by giving an argument to the ":help" command |:help|.
                   It is possible to further specify the context:
                         WHAT                 PREPEND       EXAMPLE   ~
                  Normal mode commands        (nothing)     :help x
                  Visual mode commands            v_        :help v_u
                  Insert mode commands            i_        :help i_<Esc>
                  command-line commands           :         :help :quit
~
~
:help
```

You type **:q [Return]** to exit the help screen and return to your file.

In order to get help for a specific command, you type **:help** followed by the name of the command. For example, while in command mode, type **: help wq [Return]** to get help for the **wq** command. This will show a description of the **wq** command similar to that shown in Figure 4.7.

Figure 4.7
vim Help Screen for wq Command

```
                                                                  *:wq *
:wq      Write the current file and exit (unless editing the
         last file in the argument list or the file is
         read-only)
~
~
~
:help wq
```

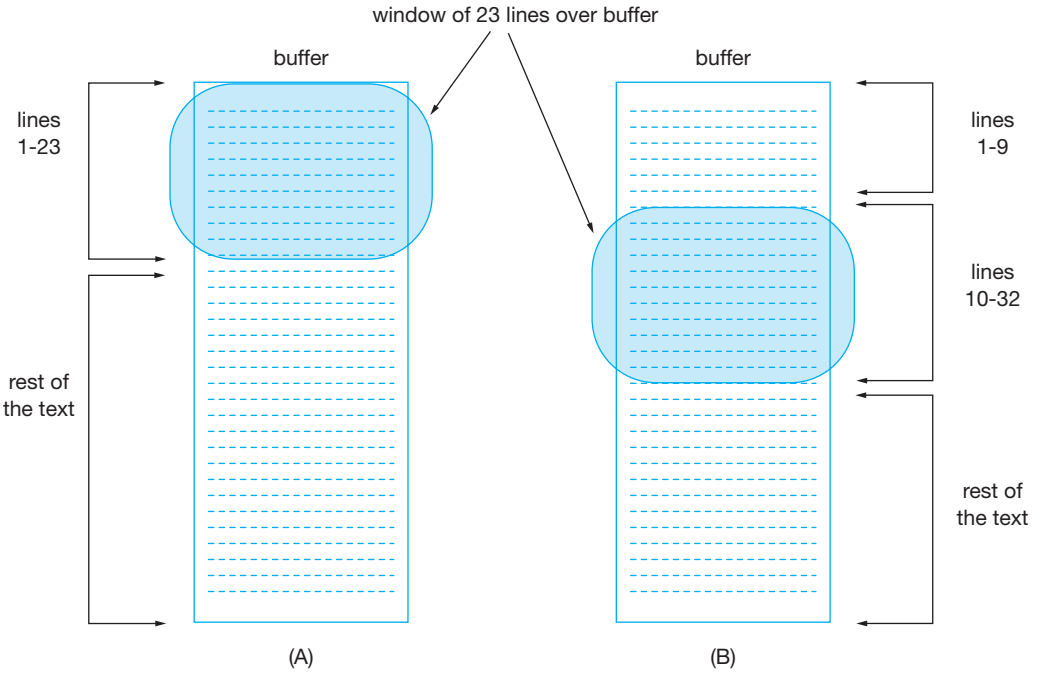
You type **:q [Return]** to exit the help screen and return to your file.

4.4 THE MEMORY BUFFER

The vi editor creates a temporary workspace for a file that you want to create or modify. If you are creating a new file, vi opens a temporary workspace for your file. If the specified file is an existing file, vi copies the original file into temporary workspace and the changes you make are applied to that copy and not the original file. This temporary workspace is called a *buffer* or *work buffer*. The vi editor uses several other buffers to manage your file during an editing session. If you want to keep the changes you have made, you must save the altered file (the copy in the buffers) to replace the original file. Changes are not saved automatically: You save your file by issuing a **write** command.

When you open a file for editing, vi copies the file into a temporary buffer and displays the first 23 lines of the file on the screen. A window of 23 lines of the text in the buffer is what you see on the screen (see Figure 4.8A). By moving this window up and down over the buffer, vi displays other parts of the text. When you use the arrow keys or other commands to move the window down, say, to line 10, the first nine lines on the screen are scrolled out (disappear from the screen) and you see lines 10 to 32 of the text (Figure 4.8B). Using arrow keys and other commands, you can move the window up or down over any portion of the file.

Figure 4.8
The vi Editor Temporary Buffer

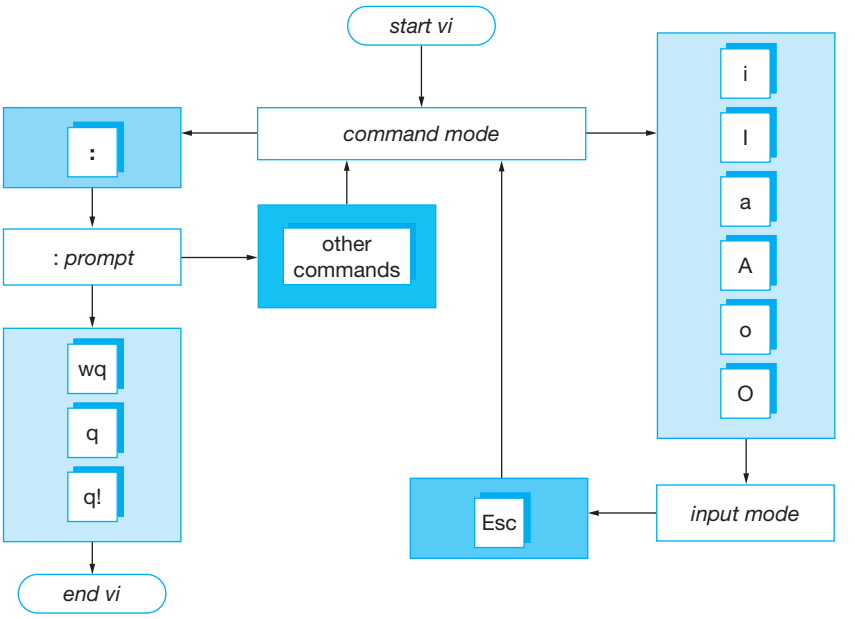


You must remember to write (save) your changes before quitting the vi editor; otherwise, your changes are discarded.

COMMAND SUMMARY

The following vi editor commands and operators have been discussed in this chapter. To refresh your memory, Figure 4.9 shows the vi editor modes of operation.

Figure 4.9
The vi Editor Modes of Operation



The vi editor
vi is a screen editor you can use to create files. vi has two modes: the command mode and the text input mode. To start vi, type **vi**, press **[Spacebar]**, and type the name of the file. Several keys place vi in the text input mode, and **[Esc]** always returns vi to the command mode.

The change mode keys
These keys change vi from the command mode to the text input mode. Each key places vi in the text input mode in a different manner. **[Esc]** places vi back in the command mode.

Key	Operation
i	Places the text you enter before the character that the cursor is on.
I	Places the text you enter at the beginning of the current line.
a	Places the text you enter after the character that the cursor is on.
A	Places the text you enter after the last character of the current line.
o	Opens a blank line below the current line and places the cursor at the beginning of the new line.
O	Opens a blank line above the current line and places the cursor at the beginning of the new line.

Correcting text keys

These keys are applicable in the command mode only.

Key	Operation
x	Deletes the character specified by the cursor position.
dd	Deletes the line specified by the cursor position.
u	Undoes the most recent change.
U	Undoes all the changes on the current line.
r	Replaces a character that the cursor is on.
R	Replaces characters starting from the cursor position, and changes vi to the text mode.
.	Repeats the last text changes.

Cursor movement keys

These keys allow you to move around in your document in command mode.

Key	Operation
h or **[Left Arrow]**	Moves the cursor position one space to the left.
j or **[Down Arrow]**	Moves the cursor position one line down.
k or **[Up Arrow]**	Moves the cursor position one line up.
l or **[Right Arrow]**	Moves the cursor position one space to the right.
$	Moves the cursor position to the end of the current line.
w	Moves the cursor position forward one word.
b	Moves the cursor position back one word.
e	Moves the cursor position to the end of the word.
0 (zero)	Moves the cursor position to the beginning of the current line.
[Return]	Moves the cursor position to the beginning of the next line.
[Spacebar]	Moves the cursor position one space to the right.
[Backspace]	Moves the cursor position one space to the left.

The quit commands

With the exception of the **ZZ** command, these commands start with **:**, and you must end a command line with **[Return]**.

Key	Operation
wq	Writes (saves) the contents of the buffer and quits the vi editor.
w	Writes (saves) the contents of the buffer but stays in the editor.
q	Quits the editor.
q!	Quits the editor and abandons the contents of the buffer.
ZZ	Writes (saves) the contents of the buffer and quits the vi editor.

The search commands

These keys allow you to search forward or backward in your file for a pattern.

Key	Operation
/	Searches forward for a specified pattern.
?	Searches backward for a specified pattern.

REVIEW EXERCISES

1. What is an editor?
2. What is a text formatter?
3. Name the editors that the UNIX operating system supports.
4. Name the vi modes.
5. Name the keys that place the vi editor in the text input mode.
6. Explain how the vi editor uses buffers.
7. Name the command that saves your files and quits the vi editor.
8. Name the command that just saves your file and remains in the vi editor.
9. Name the key that places the vi editor in the command mode.
10. Name the operator that deletes one line of text and the operator that deletes five lines of text.
11. Name the operator that deletes a character and the operator that deletes 10 characters.
12. Name the key that repeats your most recent text change.
13. Name the key that moves the cursor position to the end of the current line.
14. Name the key that moves the cursor position forward one word.
15. Name the cursor movement keys that move the cursor up, down, left, and right.

16. Name the key that appends the text you enter to the end of the current line.
17. Name the key that opens a line above the current line.
18. Name the key that opens a line below the current line.
19. Name the key that undoes the most recent changes.
20. Name the key that undoes all the changes on the current line.

Match the commands shown in the left-hand column to the explanations shown in the right-hand column.

1. **x**	a. Moves the cursor position one line up.
2. **r**	b. Deletes the character under the cursor.
3. **/**	c. Moves the cursor position to the beginning of the current line.
4. **?**	d. Places the text you enter after the last character on the current line.
5. **h**	e. Moves the cursor position to the end of the current line.
6. **A**	f. Searches backward for a specified pattern.
7. **q!**	g. Quits the vi editor without saving your file.
8. **wq**	h. Writes (saves) your file and quits the vi editor.
9. **a**	i. Places the text you enter after the character that the cursor is on.
10. **$**	j. Moves the cursor position one space to the left.
11. **0 (zero)**	k. Searches forward for a specified pattern.
12. **k**	l. Replaces the character that the cursor is on.

Terminal Session

In this terminal session, you will create a small text file and practice using the editing keys that vi provides. Use your imagination. Do not limit yourself to the small file in this exercise.

Try to use all the keys that are explained in this chapter.

1. Use the vi editor to create a file called `Chapter4` and type in the text shown on screen 1.
2. Save this file.

Screen 1

```
The vi history
The vi editor was developed at the University of california, berkeley
as part of the berkeley unix system.
~
~
"Chapter4" [new file ]
```

3. Open the test file again and add text to make it look like the text shown on screen 2.
4. Save this file again.

Screen 2

```
The vi history
The vi editor was developed at the University of california
berkeley as part of the berkeley unix system.
At the beginning the vi editor was part of another editor
The vi part of the ex editor was often used and became very.
This popularity forced the developers to come up with a separate
vi editor.
now the vi editor is independent of the ex editor and is available on
most of the UNIX operating system.
The vi editor is a good editor for everyday editing jobs.
```

5. Open the test file once more and edit the text to make it look like the text on screen 3.
6. Search for the word **vi** using the **/** (forward search). Use **n** to find the next occurrence of the word **vi**.
7. Search for the word **vi** using the **?** (backward search). Use **n** to find the next occurrence of the word **vi**.
8. Place the cursor at the beginning of the file and delete five lines. Undo your delete.
9. Place the cursor at the beginning of the second line and delete 10 characters. Undo your delete.
10. Use **r** to replace the character at the cursor position. Undo your action.
11. Use **R** to change the word **developers** to **creators**. Undo your action.
12. If you have a Linux system, practice the following:
    a. Get help for the **help** command.
    b. Get help for the **ZZ** command.
    c. Get help for the **search** command.
13. Save your file for later exercise sessions using the **ZZ** command.

Screen 3

```
The vi history
The vi editor was developed at the University of California
Berkeley as part of the Berkeley UNIX system.
At the beginning the vi (visual) editor was part of the ex editor
and you had to be in the ex editor to use the vi editor.
The vi part of the ex editor was often used and became
very popular. This popularity forced the developers to come up
with a separate vi editor.
Now the vi editor is independent of the ex editor and is available
on most of the UNIX operating systems.
The vi editor is a good, efficient editor for everyday editing jobs
although it could be more user friendly.
~
~
~
~
```