Introduction to the vi Editor

This tutorial document covers basic usage of the vi text editor, which is available on all UNIX systems. Topics include cursor movement, text insertion, markers, and text buffers. Additional topics cover the cut, copy, and paste commands as well as searching and replacing. An advanced section introduces vi variables, key maps, macros, and saving a configuration in the .exrc file.

For a reference card of vi commands, see document Unix 4.01.

Table of Contents

Getting Started

This section presents a quick introduction to vi. After reading this section, you will have an adequate knowledge of vi to perform simple text editing operations.

General Information

About vi

vi (pronounced vee-EYE, short for “visual”) provides basic text editing capabilities. Three aspects of vi make it appealing. First, vi is supplied with all UNIX systems. You can use vi at other universities or any businesses with UNIX systems. Second, vi uses a small amount of memory, which allows efficient operation when the network is busy. Third, because vi uses standard alphanumeric keys for commands, you can use it on virtually any terminal or workstation in existence without having to worry about unusual keyboard mappings. As a point of interest, vi is actually a special mode of another UNIX text editor called ex. Normally you do not need to use ex except in vi mode.

Symbols Used in This Document

Throughout this document, you will see examples of commands which you can enter. Enter the text exactly as it appears with the following exception: text in *italics* means you should substitute the appropriate text (like a filename) for the italicized word.

vi commands are case-sensitive, which means that upper-case and lower-case commands are not the same command. For example, j moves the cursor down, but J combines two lines into one line.

The word RETURN represents the action of pressing RETURN key. ESC indicates pressing the ESCAPE key. Also, sometimes you will see a keystroke like CTRL-F. CTRL-F means you hold down the key marked CONTROL or CTRL, and press F. The CTRL key acts in a manner similar to the SHIFT key.

A Brief vi Session

Starting vi

To start vi, enter:

vi *filename* RETURN

where *filename* is the name of the file you want to edit. If the file does not exist, vi will create it for you. You can also start vi without giving any *filename*. In this case, vi will ask for one when you quit or save your work.

Exercise 1: Start vi in preparation for some tutorial exercises; throughout this document, the exercises will use the file, sample. In an xterm, type:

vi sample RETURN

The window clears and displays the contents of the file, sample. Since it is a new file, it does not contain any text. vi uses the tilde (~) character to indicate lines on the screen beyond the end of the file.

---

vi uses a *cursor* to indicate where your next command or text insertion will take effect. The cursor is the small rectangle, which is the size of one character, and the character inside the rectangle is called the *current character*.

At the bottom of the window, vi maintains an announcement line, called the *modeline*. The modeline lists the current line of the file, the filename, and its status.

Exercise 2: Look for the cursor at the beginning of the file, and check the modeline, which is at the bottom of the window. It should read:

```
"sample" [New file]
```

Command Mode and Input Mode

vi has two modes, *command mode* and *input mode*. In command mode, characters you type perform actions like moving the cursor, cutting or copying text, or searching for some particular text. In input mode, you type to insert or overwrite text. When you start vi, it is in command mode.

To switch from command to input mode, press the "i" key (you do not need to press RETURN). vi lets you insert text beginning at the current cursor location. To switch back to command mode, press ESC. You can also use ESC to cancel an unfinished command in command mode.

Unfortunately, vi does not normally indicate which mode you are in. The next exercise turns on a mode indicator. If you are uncertain about the current mode, you can press ESC a few times. When vi beeps, you have returned to command mode.

Exercise 3: Before you start typing text, the command below activates the modeline indicator. You are not required to use it, but it tells you whether you are in command or input mode. Type:

:set showmode RETURN

Nothing appears to change. When you are in command mode, there is no indicator, but if you enter input mode, one will appear in the bottom right-hand corner of the window.

Inserting Text

While in input mode, you can enter text by typing normally. vi recognizes a few special keystrokes as you type.

TABLE 1. Commands to Insert Text

Backspace or Delete	Erase the previous character
CTRL-W	Erase the previous word
CTRL-U	Erase the current line
RETURN	Start a new line

Exercise 4: To enter input mode, press:

i

Note that in the bottom right-hand corner, vi indicates that you are in input mode. Then enter the following text, and remember to press RETURN at the end of each line. You can use the special keystrokes if you make a mistake.

We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hands to do it. “Oh, no,” I said.

Our heavens darkened grimly. Thunder echoed overhead and shook the
clouds, even the ground rumbled as each clap loudly exploded.
Glancing back, I saw an ocean rising behind us.
It just wouldn’t stop raining.

Our weathered tent was a poor shelter tonight. As a lightning bolt
flashes over the hills, we made out a small cave on the mountainside.
A safe haven, thought I.

Our heavens fell down, but just up ahead lay safety.

After entering this text, press ESC to return to command mode. Notice that the modeline clears, too.

Using ex Commands

When in **vi** command mode, typing “:” allows you to access the **ex** editor command set. There are a number of **ex** commands that you will want to use within the **vi** editor. **ex** commands are very powerful and can allow you to make some large changes to your files quite efficiently.

Cursor Movement

You will clearly need to move the cursor around your file. You can move the cursor in command mode. vi has many different cursor movement commands. The four basic keys appear below. You can also use the backspace key (but not the DELETE key) and spacebar to move left and right, respectively.

k	move up one line
h	line move one character to the left
l	line move one character to the right
j	move down one line

A table of additional movement commands appears below. Notice that vi has its own definitions of sentences and paragraphs, so understanding how vi recognizes them is useful.

sentence A sentence is all the characters between normal sentence punctuation marks: period (.), question mark (?), and exclamation point (!). A blank line also ends a sentence.

line The text between two RETURN characters forms a line. Hence, it is possible to have lines which are wider than the vi display.

paragraph A paragraph is a sequence of lines which are not interrupted by any blank lines. Thus, the sample document from the previous exercise has four paragraphs.

TABLE 2. Cursor Movement Commands

Command	Cursor Moves to
b	beginning of previous word
w	beginning of next word
e	end of current/next word
0 (zero) or ^	beginning of line
$	end of line
(	beginning of current/previous sentence
)	beginning of next sentence
{	beginning of current/previous paragraph
}	end of current paragraph
H	top line on screen
M	middle line on screen
L	bottom line on screen

Deleting Text

Sometimes you will want to delete some of the text you are editing. To do so, first move the cursor so that it covers the first character of the group you want to delete, then type the desired command from the table below.

TABLE 3. Deletion Commands

Command	Function
x	Delete only the current character
D	Delete to the end of the line
db	Delete from the current character to the beginning of the current word
de	Delete from the current character to the end of the current word
dd	Delete the current line
dw	Delete from the current character to the beginning of the next word

Notice that the second letter of the command specifies the same chunk of text that the cursor movement commands do. In fact, you can use delete with all of the cursor movement specifiers listed in Table 2 (e.g. dH would delete from the top line on the screen) to delete the desired chunk of text. Also, **D** is equivalent to **d$**.

Exercise 5: Try using the cursor movement keys to place the cursor on the word “hands.” Move the cursor onto the “s.” Press **x** to delete it. Now let’s delete the word “loudly.” Move to the beginning of the word. Although you could press **x** seven times to delete the word and its trailing space, it is quicker to delete it by typing **dw**. Finally, remove the line “It just wouldn’t stop raining.” Move the cursor anywhere on that line, and type **dd**. The line vanishes.

Making Corrections

In other cases you will only need to change a single character or word, rather then deleting it. vi has change and replace functions, too. First move to the position where the change should begin (the desired line or the beginning of the desired word). Next type the proper command from the table below. Finally, enter the correct text, usually concluded with ESC (except for **r**) **.**

TABLE 4. Correction/Replacement Commands

Command	Action
cw	Change word. vi displays a dollar sign ($) marking the end of the change portion.
C	Overwrites to the end of the line.

TABLE 4. Correction/Replacement Commands

Command	Action
r	Replace a single character with another one. No ESC necessary.
R	Overwrite characters starting from the current cursor position.
s	Substitute one or more characters for a single character.
S	Substitute the current line with a new one.

The change command **c** works like the delete command; you can use the text portion specifiers listed in Table 2 (e.g. dH would delete from the top line on the screen) that act as cursor movement commands to replace the selected text. **C** is equivalent to **c$**.

Exercise 6: Let's correct some words. To maintain proper verb tense, "flashes" should be "flashed." Move the cursor to the last "s" in "flashes." To enter replace mode, type:

r

and notice that the modeline says, "`REPLACE 1 CHAR.`" Next type the letter:

d

to make the change. Notice that you do not need to press ESC or RETURN when replacing just one character. Next change "safety" to "sanctuary." Move to the beginning of the word "safety," then type:

cw

vi shows a dollar sign ($) indicating the end of the text being corrected. Type:

sanctuary ESC

To finish the exercise, move the cursor to "do." We'll change all the text from "do" to the end of the line. Start the correction by pressing:

C

Again vi shows a dollar sign (this time at the end of the line). Make the change by typing:

reach that island. ESC

Undoing

Occasionally you will accidentally issue a command or delete some text and want to restore your text to the way it was before you issued that command. vi lets you undo the last text change with the undo command, which you execute by typing **u**. Note, however, that vi will only recover the last text change.

Exercise 7: Move to the line "flashed over the hills..." and delete it by typing:

dd

Bring back the line by pressing:

u

which undoes the last text change. Press **u** a few more times, and watch what happens. When you undo something, you change the text, so your undo becomes the last text change! That explains why undoing appears to flip between two displays. Leave the line “flashed over...” on the display.

Joining Lines

Occasionally you will want to link two or more lines of text together, usually because deleting text has created a lot of empty space. The **J** command combines the current line with the line below it.

Exercise 8: Move to the blank line just after “in foot and hand.” Press:

i

and insert the following text:

I saw everything spinning wildly. RETURN

Press ESC to return to command mode. Now move to the line “in foot and hand” and press:

J

to join it with the line you just typed.

At this point the vi screen should look like:

We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hand to reach that island. I saw everything spinning wildly.

Our heavens darkened grimly. Thunder echoed overhead and shook the
clouds, even the ground rumbled as each clap exploded.
Glancing back, I saw an ocean rising behind us.

Our weathered tent was a poor shelter tonight. As a lightning bolt
flashed over the hills, we made out a small cave on the mountainside.
A safe haven, thought I.

Our heavens fell down, but just ahead lay sanctuary.
~
~

Saving Your Work

vi provides several means of saving your changes. Besides saving your work before quitting, it’s also a good idea to save your work periodically. Power failures or system crashes can cause you to lose work. From command mode, you type:

:w RETURN

to save your work (“w” is for “write”). Similarly, to quit vi use the command:

:q RETURN

Exercise 9: In command mode, type:

:w RETURN

to enter the “write” command. After a moment, vi gives you a report at the bottom of its window:

```
“sample” 13 lines, 539 characters
```

Don’t worry if the line and character numbers differ a bit from yours. Most importantly, you just saved your work. Now you are ready to quit vi. From command mode, type:

:q

Again the colon signals an ex command at the bottom of the vi display. Press RETURN to finish entering the command and to exit vi.

You can combine these two commands at once to write-and-quit by entering command mode and typing

:wq RETURN

A shorthand for the command mentioned above is “ZZ” (SHIFT ZZ).

In some cases you will want to abandon your changes since the last time you saved your work (the last :w command). To do so, type:

:q! RETURN

which tells vi to quit without saving. Use caution when abandoning vi in this manner because any changes you made will be permanently lost.

A Second Session

Repeating a Command

Often you will want to act on more than one character, word, or line. For example, suppose you want to delete three words. Rather than type **dw** three times, you can type **3dw**, and vi will execute the command three times. Many vi commands can be repeated in this manner.

Just type a number (it can be more than one digit), then type the command. If you want to abandon the number, press ESC to cancel it.

Exercise 1: Start vi with the sample document by typing:

vi sample RETURN

Once vi starts, turn on the modeline with:

:set showmode RETURN

Move to the “e” in “echoed overhead and,” then type:

3dw

and notice that vi deletes three words at once. (Then press **u** to undo the deletion.)
You can also repeat text input. Move to the blank line after “Glancing back...” and enter:

5iBOOM! ESC

(put a space between the exclamation point (!) and ESC). Notice that vi adds “BOOM!” five times.
Finally, try cancelling a number. Move to the word “up” on the last line and type (ignore the beep):

15 ESC 3x

The ESC cancels the number 15, so vi only deletes “up ” instead of the 15 characters “up ahead lay sa.”

Line Numbers

Many vi commands use *line numbers*, which simply count the number of RETURN characters in a file. You can cut and copy text by line number or jump to a certain line. Line numbers can be useful when you receive error messages during program compilation. Frequently compilers will print the line number, so you can use vi to jump to the appropriate line and look for the error.

Exercise 2: To display the line numbers, enter the following command:

:set number RETURN

This command will immediately display the line numbers in the left margin of your vi window. It may cause long lines to wrap around the right edge of the window, but they will not be damaged.

The **G** movement command lets you jump to any line within a file. First type the line number, followed by **G**. If you do not type any line number, vi jumps to the end of the file. Thus, **1G** takes you to the beginning, and **G** takes you to the end.

Another command reports the current line number and the status of your file. When you type **CTRL-G**, vi displays the filename, whether it has been altered since it was last saved, and the current line number along with the percentage of the text of the file representing all the lines up to the current one.

Exercise 3: From command mode, try jumping to line 1, the end of the document, and line 6 by entering:

1G
G
6G

After typing each of these "go to" commands, notice that the cursor jumps to the desired line. You can type a multiple-digit line number, of course. Next, turn off the line numbers by entering:

:set nonumber RETURN

Finally, press CTRL-G and the modeline should look something like:

```
"sample" [Modified] line 6 of 13 --46%--
```

Markers

During an editing session, you may grow weary of typing lengthy cursor moves or having to remember lots of line numbers. vi lets you set *markers* anywhere in a file. These markers make movement, copying, and cutting text much easier. Markers are named from "a" to "z," where the letter distinguishes them from each other. When you quit vi, the markers vanish, so you must set them each time you start vi.

Setting markers is easy. Just move the cursor where you want to place the marker, then type **m***letter* where *letter* specifies the marker name. *Letter* can be any lower-case letter from a to z. Regrettably, vi does not indicate that you just set a marker. You can set markers in any order. You do not have to label them sequentially (a, b, c, etc.).

Exercise 4: Let's set a few markers and see how they work. Move the cursor to the word "weathered," then set a marker there. Type:

mw

Now move over to the word "bolt" and set a marker by typing:

ma

Next go to "island" and type:

ml

Once you have set a marker, you can return to that line or exact character quickly.

- To jump to the marker *letter* type ‘*letter* (Use an accent grave (‘)).
- To jump to beginning of the line containing the marker type ’*letter*. (Use the single quote (’)).

Exercise 5: Move to the first line of the document, then jump among your markers by typing the following:

‘w
‘l
‘a
’l
’a
‘w

Lastly, try to jump to an undefined marker by typing:

’c

vi just beeps at you because it doesn’t know where to jump.

Other Input Modes

Besides insert mode, vi employs a few other input modes. They all let you enter text; the only difference is *where the insertion point is*. Table 5 below describes the three most common modes: append, insert, and open. Two other text input modes are change mode and replace mode, both of which you’ve used. The mode indicator displays the current mode.

TABLE 5. Input Modes

Command	Mode Name	Insertion Point
a	append	just after the current character
A	Append	end of the current line
i	insert	just before the current character
I	Insert	beginning of the current line
o	open	new line below the current line
O	Open	new line above the current line

Exercise 6: Move to the line “Our heavens fell...” and press **A** to append text at the end of the line. Then type (the symbol -- means a space character):

--We crawled in, and wept. ESC

Next go to the line “in foot and hand...” and open a new line below it. Press:

o

to start open mode, and observe that vi creates a new line for text. Type:

Take a step and fall. ESC

After performing the exercises in this chapter, the screen should look like:

We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hand to reach that island. I saw everything spinning wildly.
Take a step and fall.

Our heavens darkened grimly. Thunder echoed overhead and shook the
clouds, even the ground rumbled as each clap exploded.
Glancing back, I saw an ocean rising behind us.
BOOM! BOOM! BOOM! BOOM! BOOM!
Our weathered tent was a poor shelter tonight. As a lightning bolt
flashed over the hills, we made out a small cave on the mountainside.
A safe haven, thought I.

Our heavens fell down, but just ahead lay sanctuary. We crawled in, and wept.
~
~
~
~

Exercise 7: Now you have finished the exercises in this section. Enter:
:wq RETURN
to write your changes and quit vi**.**

Cut, Copy, and Paste

Frequently you'll need to cut or copy some text, and paste it elsewhere in your document. First you cut or copy the text into temporary storage, then you paste it into a new location. *Cutting* means removing text from the document and storing it, while *copying* means placing a duplicate of the text in storage. Finally, *pasting* just puts the stored text in the desired location.

Buffers

vi uses a *buffer* to store the temporary text. There are nine numbered buffers in addition to an undo buffer. The undo buffer contains the most recent delete. Usually buffer 1 contains the most recent delete, buffer 2 the next most recent, and so forth. Deletions older than 9 disappear. However, vi also has twenty-six *named buffers* (a-z). These buffers are useful for storing blocks of text for later retrieval. Buffer letters are independent of marker letters, so buffer k and marker k are unrelated. To retrieve text from a buffer, see the section, *Cut and Copy (Delete and Yank)*.

The content of a buffer does not change until you put different text into it. Unless you change the contents of a named buffer, it holds its last text until you quit. As with markers, vi does not save your buffers when you quit.

By Line Number

Two simple commands from the ex command set let you cut and copy text by entering the range (in lines) and the destination line. The **m** command *moves* (cuts-and-pastes) a range of text, and the **t** command *transfers* (copies-and-pastes) it. **t** is a synonym for **c** (copy). The commands have the form shown below.

:*line1*m*destline*	Move (cut) line number, *line1*, to the line number, *destline*
:*line1*,*line2*m*destline*	Move (cut) lines between and including *line1* and *line2* below line number, *destline*
:*line1*t*destline*	Transfer (copy) line number, *line1*, to the line just below line number, *destline*
:*line1*,*line2*t*destline*	Transfer (copy) lines between and including *line1* and *line2* below line number, *destline*

Exercise 1: First start vi as in the previous chapter, then try these commands. Move to the line "Take a step" and press CTRL-G. Note the line number (probably 3). Then move to the line "A safe haven" and press CTRL-G, and note this line number (probably 11). Now you are ready to copy ("move") "Take a step" to another line. Type:

:3m11
and press RETURN to finish the command.

Cut and Copy (Delete and Yank)

General

vi has its own terminology for “cut” and “copy”: “delete” and “yank,” respectively. Note that the delete command is the same one you have already used; every time you delete text, vi changes the automatic buffer and pushes the previous delete into the next numbered buffer. When you delete or yank, the desired text enters a buffer. If you do not specify a named buffer, vi uses the automatic buffer (buffer 1).

The delete and yank commands take the following form:

1. Move the cursor to one end of the desired text.
2. If desired, specify a named buffer by typing “*letter* where *letter* is a letter from a through z. If you do not give a named buffer, vi uses the automatic buffers 1-9.
3. Type a repetition number, if needed. (To copy 5 words or 8 lines, for example.)
4. Type **d** to delete text, or type **y** to yank text.
5. Type a cursor movement key (b, e, w, etc.—see Table 2) to determine the text unit; if you type **d** or **y** instead, vi uses the “line” unit. The cursor key completes the delete or yank and stores the text in the desired buffer.

Using Markers

Markers also let you specify a range of text for cutting or copying without having to count words or lines. The next procedure shows you how to delete or yank text by using markers.

1. Move the cursor to one end of the selection.
2. Type **m***letter* to set a marker.
3. Move the cursor to the other end.
4. If desired, specify a named buffer by typing

   “*letter*

   If you do not give a named buffer, **vi** uses the automatic buffers.
5. Type **d** or **y** to delete or yank text, respectively.
6. Using *letter* from the marker, type

   ‘*letter*

   to delete or yank the text between the marker and the cursor. You may instead type

   ’*letter*

   to delete or yank the text by line.

Paste

Pasting text from a buffer involves three steps:

1. Move the cursor to the desired pasting location.
2. If retrieving text from a named buffer, specify the buffer by typing "*letter*. Otherwise vi uses the automatic buffers.
3. Type **p** to paste the buffered text just after the current character or type **P** to paste it just before the current character. If the buffered text is stored by line, it will be pasted below or above the current line.

Exercise 2: This exercise performs a standard copy-and-paste. It uses the automatic buffer and does not use any markers. Move to the first line of text ("We scrambled..."), then yank three lines of text by typing:

3yy

Nothing appears to happen, but vi has put the first three lines into its automatic buffer. The message "3 lines yanked" appears at the bottom of the screen. Now move to the line "A safe haven and paste the lines in place. To do so, just press:

p

Exercise 3: In this exercise you use markers to identify the text you want to cut, as well as perform a normal cut. First, move to "Glancing" and put a marker there by typing:

mx

Then move to the "B" in the fourth "BOOM!". Cut the text and place it in buffer e by typing:

"ed'x

The text vanishes, but vi has put it into buffer e.

Second, copy the remaining "BOOM! BOOM!" into the automatic buffer. Just type:

dd

Third, paste the contents of the automatic buffer: move to the second instance of "We scrambled..." and press:

P (capital "P")

Fourth and finally, move to the blank line below "Take a step..." Paste the text from buffer e by typing:

"ep

After those two exercises, the screen should look like that below. Once again, end the chapter exercises by saving your work with:

:wq RETURN

We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hand to reach that island. I saw everything spinning wildly.

Our heavens darkened grimly. Thunder echoed overhead and shook the
clouds, even the ground rumbled as each clap exploded.

Our weathered tent was a poor shelter tonight. As a lightning bolt
flashed over the hills, we made out a small cave on the mountainside.
A safe haven, thought I.
BOOM! BOOM!
We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hand to reach that island. I saw everything spinning wildly.

Take a step and fall. Glancing back, I saw an ocean rising behind us.
BOOM! BOOM! BOOM!
Our heavens fell down, but just ahead lay sanctuary. We crawled in, and wept.
~
~

Search and Replace

As files become longer, you may need assistance locating a particular instance of text. vi has several search and search-and-replace features.

Simple Search

vi can search the entire file for a given string of text. A *string* is a sequence of characters. vi searches forward with the slash (/) key or backward with the question mark key (?). You execute the search by typing the command key, then *string* followed by RETURN. To cancel the search, press ESC instead of RETURN.

You can search again by typing **n** (forward) or **N** (backward). Also, when vi reaches the end of the text, it continues searching from the beginning. This feature is called *wrapscan*.

Exercise 1: Start vi as usual and try a simple search. Type:

```
/and
```

and press **n** and **N** a few times to see where the cursor goes.

Special Characters

vi supports a few special characters, which act as wildcards or search-exclusions. These special characters and their use are summarized below and in the exercises. Note that *cccc* stands for any number of characters; it could be **g**, **gefha**, or **23CG-4**. The special characters are: $. * [ ] ^ \.

When specifying search strings, you will sometimes want to search for one of the special characters. To do so, type a backslash (\) immediately before the special character. For example, \$5 matches “$5.00.” To specify a single backslash, type \\.

The table below summarizes the special characters. Note that the slash (/) starts a forward search. You can replace it with a question mark (?) to search backwards.

TABLE 6. Searching with Special Characters

Usage	Action	Example	Matches
[*cccc*]	match any of the characters *cccc*	/sa[fn]	safe, sanctuary
[^*cccc*]	match all characters except *cccc*	/[^a]nd	behind, ground
[*c1-c2*]	match any characters between *c1* and *c2*	/[d-h]er	thunder, weathered
\<*cccc*	match words beginning with *cccc*	/\<eac	each

TABLE 6. Searching with Special Characters (Continued)

cccc\>	match words ending with *cccc*	/und\>	ground
^*cccc*	match lines beginning with *cccc*	/^in	in foot and...
cccc$	match lines ending with *cccc*	/slow$	We scrambled...
.	match any single character	/i.l	grimly, hills
c*	match the character c zero or more times	/mb*d	scrambled, rumbled
.*	match any characters	/b.*k	back, scram[bled to strik]e

Exercise 2: Try some of the search strings in the preceding table, then try the following combinations:
/ \<s.o words starting with s, any letter, then o (shook, slow)
/ \<h.*s\> words starting with h and ending with s with any number of characters in between (heavens, hills)
/o.[rtk] words containing o, any letter, and r, t, or k (bolt, foot, poor, shook)

Search and Replace

vi can also search-and-replace, which means finding instances of a given string and replacing them with a new string. This search-and-replace operation is actually an ex command, and it has the following form:

: *line1, line2*s/ *oldstring / newstring*

You specify the range of text for the search-and-replace command with the line numbers *line1* and *line2*. If you only give *line1* and omit the comma, then the command only affects that line. If you give no line numbers, the command affects the current line. The search-and-replace command only finds the first occurrence of *oldstring* on the line.

You can repeat a search-and-replace on the current line by typing an ampersand (&). To repeat it on a different line or group of lines, type:

: *linenumber&*

or

: *line1,line2&*

Exercise 3: Let's replace the word "heavens" with "sky." Move to the last line ("Our heavens fell...") and press CTRL-G to see the line number. Note this line number (probably 16). Next move

to the line (“Our heavens darkened...”), where we’ll do the first search-and-replace. Type:

:s/heavens/sky

and press RETURN. The first instance of “heavens” becomes “sky.” Now press the ampersand (&) to repeat the command. Nothing happens; in fact, vi says:

```
Substitute pattern match failed
```

because vi looks for “heavens” on the current line. Instead use the line number from the beginning of this exercise and type:

:16 &

Press RETURN and vi replaces the occurrence of “heavens” on line 16.

Special Flags

You can add a flag to the search-and-replace command; the flag tells vi to replace every occurrence or to ask for confirmation before each replacement. To add a flag, use the following form:

: *line1, line2*s*/ oldstring / newstring / flag*

Flag is an optional parameter. If *flag* is **c** then vi will show you each instance of *oldstring* and wait for confirmation; type **y** (for yes) or **n** (for no) followed by RETURN. A flag of **g** requests a global search-and-replace, without confirmation. Global in this case means every occurrence of *oldstring* on the current line.

Exercise 4: Execute a search-and-replace with confirmation. Press CTRL-G and note how long the text is (probably 17 lines). Then enter:

:1,17s/Our/The/c

and press RETURN. vi will prompt you with each instance of the word “Our” and await a response. The first time (“Our sky darkened...”), press:

y RETURN

The second time (“Our weathered tent...”), press:

n RETURN

The third time (“Our sky fell...”), press:

y RETURN

Press RETURN once more (vi prompts you at the bottom of the screen), and vi updates the screen.

Check your screen against the one below, then save your work and quit vi to finish these exercises.

We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hand to reach that island. I saw everything spinning wildly.

The sky darkened grimly. Thunder echoed overhead and shook the
clouds, even the ground rumbled as each clap exploded.
Our weathered tent was a poor shelter tonight. As a lightning bolt
flashed over the hills, we made out a small cave on the mountainside.

A safe haven, thought I.
BOOM! BOOM!
We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hand to reach that island. I saw everything spinning wildly.

Take a step and fall.
Glancing back, I saw an ocean rising behind us.
BOOM! BOOM! BOOM!
The sky fell down, but just ahead lay sanctuary. We crawled in, and wept.
~
~

A Powerful Search and Replace

The ex command **g** (for global) can be used with **s** (substitute) to find and replace every occurrence of a string pattern in an entire file. The syntax of the global command is:

:g/*string*/*commands*

The global command finds each line in the file that has *string* in it and then applies the commands to it. You can combine the global and substitute it in the following manner:

:g/*oldstring*/s//*newstring*/g

You don't need to put *oldstring* in the search string part of the substitute command because the editor already has the pattern from the global command.

When specifying *oldstring*, you can use the special characters as explained in the section *Special Characters*. vi does not recognize special characters in *newstring*, since it is performing a replacement, not a search.

Advanced Topics

Variables

vi maintains several *variables* that control different aspects of its appearance. You have already used a few of these (:set number, :set showmode).

Toggle and Numeric Variables

The two types of variables are *toggle variables* and *numeric variables*. Toggle variables turn an option on or off (like displaying line numbers), while numeric variables take a number as their argument (like tab width).

You turn on a toggle variable with:

:set *variable*

and you turn it off by typing:

:set no*variable*

Numeric variables are set with an equals sign (=) and the corresponding value. For example, to set tab stops every 6 spaces, you would enter

:set tabstop=6

TABLE 7. Variable Settings

Variable	Default	Description
ignorecase	noignorecase	Do not distinguish between capital and lowercase letters during searches.
number	nonumber	Display line numbers.
showmode	noshowmode	Displays the input mode, blank for command mode.
wrapscan	wrapscan	When a search reaches the end of the file, it wraps around back to the beginning and continues the search.
report	report=5	When more than this number of lines are modified, deleted, yanked, or pasted, vi will print a message. This option alerts you to large modifications.
tabstop	tabstop=8	Sets tab stops to multiples of this value. Normally 8 works fine.
wrapmargin	wrapmargin=0	Sets the right margin. When you pass the margin set by *wrapmargin*, vi automatically creates a new line.

Useful Variables

This section contains a list of useful variables for working in vi. To see a list of all the variables and their settings, enter

:set all

Exercise 1: You already know how to turn line numbers on and off with *number*, and you have used the *showmode* variable, too. Now try a numeric variable. From command mode, type:

:set wrapmargin=5

This command sets the right margin of the file to 5 characters from the right edge. Unfortunately, it does not affect existing lines, only new text that you enter. Move the cursor onto the 'I' in "I saw everything." Enter insert mode by pressing:

i

and then type:

Staggering like a drunken bum, each step a wicked dance, ESC

Notice that your line wraps around the edge of the window, but vi automatically makes a new line for text beyond the right margin.

Special Keys and CTRL-V

Before proceeding with the rest of this chapter, you need to learn about *control characters*. Control characters are invisible characters which computers use to manage communication protocol. The RETURN key is a control character (CTRL-M), as is the ESC key (CTRL-[). Function keys are usually a sequence of control and regular characters. vi lets you put a control character into text by preceding it with CTRL-V (which is also a control character).

Exercise 2: Move to the first line of sample and press:

O (capital "o," not zero)

to create a new input line. Then type (do not type any spaces):

CTRL-V RETURN CTRL-V CTRL-T

Notice that when you first press CTRL-V, vi displays a caret (^) character, which indicates that the next character will be a control character. Next type a few more control characters (again do not type any spaces):

CTRL-V CTRL-V CTRL-V F1 CTRL-V ESC

Observe that you just inserted a CTRL-V, an ESC, and the F1 sequence into your text. Don't worry if the F1 sequence looks like garbage. Now erase the control characters (and first three lines of text), by typing:

ESC 4dd

Mapping Keys

You can map a single keystroke into several characters with the :map command. It lets you assign functions to the function keys. This command takes the form:

:map *keystroke result*

where *keystroke* is a single character or function key, and *result* is the character sequence that should be triggered. The key map remains in effect until you quit vi.

Exercise 3: This exercise maps function key F1 to mean “Go to beginning of file.” From command mode, type (_ means a space)

:map_CTRL-V F1_1G

Before you press RETURN make sure you understand what is happening: **:map** gives vi the “map” command. F1 is the *keystroke*, but since it starts with a control character, CTRL-V must precede it. Finally, 1G is the *result*, which is just the vi command to move to the first line of the file.
Now try your key map. Move to a different line and press the F1 key; the cursor should jump to the first line of the file. Try it again.

Macros

You can execute a vi *macro*, which means that vi reads the contents of a named buffer (a-z) and executes them as if they were commands. Because a macro is just text treated like vi commands, macro g and buffer g are the same buffer.

To set up a macro, delete or yank the macro text into a named buffer. Execute the macro by typing ***@char.***

Exercise 4: Here you make a macro that finds lines beginning with “BOO” and deletes them. First, go to the end of the document, then enter open mode by pressing:

o

Now you are ready to type the commands for the macro. Type (without any spaces):

/^BOO CTRL-V RETURN dd ESC

You delete (cut) the macro text into buffer z just like ordinary text: type:

“zdd

Now you are ready to execute the macro; just type:

@z

and the first line of “BOO” vanishes. Execute the macro again with **@z**.

Using fmt

The **fmt** command lets you format a range of text to fit certain margins. If you have a very raggedy document, vi can improve its appearance. **fmt** is actually another UNIX command. You will be using a feature of vi that lets you access other UNIX commands in order to use fmt.

fmt has several options, and the two most important appear below. First, the **-s** option: it preserves short lines of text, which keeps program code from becoming crammed together. Second, the **-width** option: it makes all lines shorter than the specified number, *width*.

fmt has the following structure:

:line1,line2 !fmt *options*

where *options* can be **-s** or ***-width***. For example, suppose that you are formatting a program segment, lines 8 through 27, with a file width of 68 characters. Because you are formatting program code, you should use the **-s** option with the **-68** option. Then you would enter:

:8,27!fmt -s -68

Exercise 5: Before you can clean up the file, you need to mess it up a little. Move to the line "in foot and hand," then join some lines together by pressing:

J

three or four times. Your file is quite messy now. Prepare to correct the margins. Press CTRL-G to see the file length (about 16 lines), then enter:

:1,16!fmt -72

That should clean up the file. Once again, write and quit with:

:wq RETURN

You can also format from the current line to the end of the current paragraph with the following:

!}fmt

The **!** is a vi command to filter the specified text through a command. The "}" is the text portion (i.e. paragraph, you could use ")" to refer to sentence), and **fmt** is the command. There is a slight difference between:

:line1,line2!cmd

and

!textportion cmd

The first is an ex command to pass text to another command, the second is a vi only command to do a similar thing. The commands are not interchangeable. In other words, the following line will not work:

!line1,line2cmd

The last screen of this tutorial looks like:

```
The sky darkened grimly. Thunder echoed overhead and shook the clouds
even the ground rumbled as each clap exploded. Our weathered tent was
a poor shelter tonight. As a lightning bolt flashed over the hills, we
made out a small cave on the mountainside. A safe haven, thought I.
We scrambled to strike camp. Water crashed down upon us, far too slow
in foot and hand to reach that island. Staggering like a drunken bum,
each step a wicked dance, I saw everything spinning wildly. Take a
step and fall. Glancing back, I saw an ocean rising behind us. The
sky fell down, but just ahead lay sanctuary. We crawled in, and wept.
```

Saving a vi Configuration

You have learned to turn on the mode indicator and set the right margin by typing:

:set showmode

:set wrapmargin=5

These two features are very useful, but you probably do not want to type them each time you start vi. Fortunately, you can save commands in a file called .exrc. Each time you start vi it reads .exrc and executes all the commands inside .exrc.

.exrc can contain **:map** and **:set** commands. Each one should be on a separate line, but most importantly, you should omit the colons in your .exrc**.** With a little work, you can map the function keys into your own functions: cursor movement, cut-and-paste, and write-and-quit.

Exercise 6: In this exercise, you'll make a basic .exrcfile, which you can update as you become more adept with vi.
Enter insert mode and set up two default variables with the text (notice the absence of the colons)

i

set showmode RETURN

set wrapmargin=5 ESC

Save this file. Now, whenever you start with any filename, vi will invoke these two commands. This exercise concludes the introduction to vi. Good luck!

Problems or Questions

Faculty, Staff, and Graduate Students:

If you have a problem, contact your computing support representative by sending an e-mail message to problem@rice.edu detailing your question. Your query is examined by a staff dispatcher for severity and assigned to the appropriate staff. This is the most effective communication method since computing support staff are often working in the field and unreachable by phone. In addition, the dispatcher is aware of who is on vacation or out ill.

Undergraduates:

If you have a problem, contact your computing support representative by sending an e-mail message to problem@rice.edu detailing your question. Your query is automatically assigned to your College Computing Associate (CCA).

If you need immediate assistance during normal business hours, you can call the Consulting Center at 713.348.4983. During the semester, the Consulting Center has limited evening and weekend hours as well.

To report emergencies, which are urgent system-wide problems (i.e.: all Wiess' network connections are down or all the PCs in a lab are non-functional), contact the Operations Center at 713.348.4989. Staff work 24 hours a day, 365 day a year and can page appropriate administrators for major network or computing problems.

More information is available at http://www.rice.edu/Computer/student.html