The vi Editor

To write and edit some programs and scripts, we require editors. UNIX provides vi editor for BSD system – created by Bill Joy. Bram Moolenaar improved vi editor and called it as vim (vi improved) on Linux OS.

vi Basics

To add some text to a file, we invoke,

vi <*filename*>

In all probability, the file doesn't exist, and vi presents you a full screen with the filename shown at the bottom with the qualifier. The cursor is positioned at the top and all remaining lines of the screen show a ~. They are non-existent lines. The last line is reserved for commands that you can enter to act on text. This line is also used by the system to display messages. This is the command mode. This is the mode where you can pass commands to act on text, using most of the keys of the keyboard. This is the default mode of the editor where every key pressed is interpreted as a command to run on text. You will have to be in this mode to copy and delete text

For, text editing, vi uses 24 out of 25 lines that are normally available in the terminal. To enter text, you must switch to the input mode. First press the key i, and you are in this mode ready to input text. Subsequent key depressions will then show up on the screen as text input.

After text entry is complete, the cursor is positioned on the last character of the last line. This is known as current line and the character where the cursor is stationed is the current cursor position. This mode is used to handle files and perform substitution. After the command is run, you are back to the default command mode. If a word has been misspelled, use ctrl-w to erase the entire word.

Now press esc key to revert to command mode. Press it again and you will hear a beep. A beep in vi indicates that a key has been pressed unnecessarily. Actually, the text entered has not been saved on disk but exists in some temporary storage called a buffer. To save the entered text, you must switch to the execute mode (the last line mode). Invoke the execute mode from the command mode by entering a: which shows up in the last line.

The Repeat Factor

vi provides repeat factor in command and input mode commands. Command mode command k moves the cursor one line up. 10k moves cursor 10 lines up.
To undo whenever you make a mistake, press

Esc u

To clear the screen in command mode, press

ctrl-l

Don’t use (caps lock) - vi commands are case-sensitive
Avoid using the PC navigation keys

Input Mode – Entering and Replacing Text

It is possible to display the mode in which is user is in by typing,

:set showmode

Messages like INSERT MODE, REPLACE MODE, CHANGE MODE, etc will appear in the last line.
Pressing ‘i’ changes the mode from command to input mode. To append text to the right of the cursor position, we use *a, text.* I and A behave same as i and a, but at line extremes
I inserts text at the beginning of line. A appends text at end of line. o opens a new line below the current line

- *r<letter>* replacing a single character
- *s<text/word>* replacing text with s
- *R<text/word>* replacing text with R
- Press esc key to switch to command mode after you have keyed in text

Some of the input mode commands are:

COMMAND	FUNCTION
i	inserts text
a	appends text
I	inserts at beginning of line
A	appends text at end of line
o	opens line below
O	opens line above
r	replaces a single character
s	replaces with a text
S	replaces entire line

Saving Text and Quitting – The ex Mode

When you edit a file using vi, the original file is not distributed as such, but only a copy of it that is placed in a buffer. From time to time, you should save your work by writing the buffer contents to disk to keep the disk file current. When we talk of saving a file, we actually mean saving this buffer. You may also need to quit vi after or without saving the buffer. Some of the save and exit commands of the ex mode is:

Command Action

:W	saves file and remains in editing mode
:x	saves and quits editing mode
:wq	saves and quits editing mode
:w <*filename*>	save as
:w! <*filename*>	save as, but overwrites existing file
:q	quits editing mode
:q!	quits editing mode by rejecting changes made
:sh	escapes to UNIX shell
:recover	recovers file from a crash

Navigation

A command mode command doesn't show up on screen but simply performs a function. To move the cursor in four directions,

k	moves cursor up
j	moves cursor down
h	moves cursor left
l	moves cursor right

Word Navigation

Moving by one character is not always enough. You will often need to move faster along a line. vi understands a word as a navigation unit which can be defined in two ways, depending on the key pressed. If your cursor is a number of words away from your desired position, you can use the word-navigation commands to go there directly. There are three basic commands:

b	moves back to beginning of word
e	moves forward to end of word
w	moves forward to beginning word

Example,

5b takes the cursor 5 words back
3w takes the cursor 3 words forward

Moving to Line Extremes

Moving to the beginning or end of a line is a common requirement.
To move to the first character of a line

0 or |

30| moves cursor to column 30
$ moves to the end of the current line

The use of these commands along with b, e, and w is allowed

Scrolling

Faster movement can be achieved by scrolling text in the window using the control keys. The two commands for scrolling a page at a time are

ctrl-f	scrolls forward
ctrl-b	scrolls backward

10ctrl-fscroll 10 pages and navigate faster

ctrl-d	scrolls half page forward
ctrl-u	scrolls half page backward

The repeat factor can also be used here.

Absolute Movement

The editor displays the total number of lines in the last line

Ctrl-g	to know the current line number
40G	goes to line number 40
1G	goes to line number 1
G	goes to end of file

Editing Text

The editing facilitates in vi are very elaborate and invoke the use of operators. They use operators, such as,

d	delete
y	yank (copy)

Deleting Text

x	deletes a single character
dd	delete entire line
yy	copy entire line
6dd	deletes the current line and five lines below

Moving Text

Moving text (p) puts the text at the new location.
p and P place text on right and left only when you delete parts of lines. But the same keys get associated with “below” and “above” when you delete complete lines

Copying Text

Copying text (y and p) is achieved as,

yy	copies current line
10yy	copies current line & 9 lines below

Joining Lines

J	to join the current line and the line following it
4J	joins following 3 lines with current line

Undoing Last Editing Instructions

In command mode, to undo the last change made, we use u
To discard all changes made to the current line, we use U

vim (LINUX) lets you undo and redo multiple editing instructions. u behaves differently here; repeated use of this key progressively undoes your previous actions. You could even have the original file in front of you. Further 10u reverses your last 10 editing actions. The function of U remains the same.

You may overshoot the desired mark when you keep u pressed, in which case use ctrl-r to redo your undone actions. Further, undoing with 10u can be completely reversed with 10ctrl-r. The undoing limit is set by the execute mode command: set undolevels=n, where n is set to 1000 by default.

Repeating the Last Command

The . (dot) command is used for repeating the last instruction in both editing and command mode commands
For example:

2dd deletes 2 lines from current line and to repeat this operation, type. **(dot)**

Searching for a Pattern

/ search forward
? search backward

/printf
The search begins forward to position the cursor on the first instance of the word

?pattern
Searches backward for the most previous instance of the pattern

Repeating the Last Pattern Search

n repeats search in same direction of original search

n doesn't necessarily repeat a search in the forward direction. The direction depends on the search command used. If you used? printf to search in the reverse direction in the first place, then n also follows the same direction. In that case, N will repeat the search in the forward direction, and not n.

Search and repeat commands

Command	**Function**
/pat	searches forward for pattern pat
?pat	searches backward for pattern pat
n	repeats search in same direction along which previous search was made
N	repeats search in direction opposite to that along which previous search was made

Substitution – search and replace

We can perform search and replace in execute mode using :s. Its syntax is,

:address/source_pattern/target_pattern/flags

:1,$s/director/member/g	can also use % instead of 1,$
:1,50s/unsigned//g	deletes unsigned everywhere in lines 1 to 50
:3,10s/director/member/g	substitute lines 3 through 10
:.s/director/member/g	only the current line
:$s/director/member/g	only the last line

Interactive substitution: sometimes you may like to selectively replace a string. In that case, add the c parameter as the flag at the end:

:1,$s/director/member/gc

Each line is selected in turn, followed by a sequence of carets in the next line, just below the pattern that requires substitution. The cursor is positioned at the end of this caret sequence, waiting for your response.

The ex mode is also used for substitution. Both search and replace operations also use regular expressions for matching multiple patterns.

The features of vi editor that have been highlighted so far are good enough for a beginner who should not proceed any further before mastering most of them. There are many more functions that make vi a very powerful editor. Can you copy three words or even the entire file using simple keystrokes? Can you copy or move multiple sections of

text from one file to another in a single file switch? How do you compile your C and Java programs without leaving the editor? vi can do all this.

- Source: Sumitabha Das, "UNIX – Concepts and Applications", 4th edition, Tata McGraw Hill, 2006