The vi Editor

History: The original editor on UNIX was a line editor called ed (for editor) that was later extended to ex (EXtended editor). vi (for VIsual) is a full screen editor based on the line editor ex. emacs (Editing MACroS) is another popular editor for UNIX systems. Linux has an improved version of vi called vim (VI iMproved) that is much more user-friendly. Interestingly, vi and ex are not really two different programs – they are two links to the same executable. The vi command can also be invoked as view, in which case the readonly option is set (the file is open in read-only mode, allowing only reading and no modification). Most of vim was made by Bram Moolennaar, with a lot of help from others.

Help on the vi command line can be found in the manual of vi. An extensive help on vi can be accessed by typing :help in the command mode in vi.

General Concepts:

- vi uses short commands for performing various tasks.
- Like other UNIX commands, **capital and little letters in commands are treated differently**.
- Vi has certain *operators*. The operators themselves are not commands, but can be turned into commands in various ways. Some common operators are d (delete/cut), y (yank/copy), c(change), etc.
- When an operator is repeated, it applies to the whole line. For example, d is the operator for delete. Hence dd deletes the whole line. Similarly, y is the operator for yank(copy). Hence yy copies the whole line.
- When an operator is followed by a cursor movement command, the operator applies to the text covered by the cursor movement that would have been effected by the cursor movement command. For example, w moves the cursor to the end of the current/next word, $ moves the cursor up to the end of the line and % moves the cursor to a matching (, [or {. Hence, dw deletes up to the end of the current/next word, d$ deletes up to the end of the line while d% deletes up to a matching (, [or {.
- Most commands, including commands formed from operators, accept an optional number before the command, indicated in this text by *n* (an italicized n), as a repeat count. When a repeat count is provided, the command is repeated that many number of times. For example, dd deletes one line, 5dd deletes 5 lines and 100dd deletes 100 lines. Similarly, 3dw deletes 3 words and 5dw deletes 5 words. i*ESC in command mode inserts the character * at the cursor. 20i*ESC in command mode inserts the character * 20 times.
- Commands beginning with : are actually commands of the ex editor, vi's predecessor and companion editor.

Modes: vi can operate in four modes – command mode, insertion mode, last line mode and visual mode.

- **Command mode:** in this mode whatever you type is treated as a command to be executed. When you open vi, it starts in this mode.
- **Insertion mode:** In this mode whatever you type is entered at the current cursor position. You enter this mode when you press any of the following keys in the command mode.

  a, A, i, I, o, O, cx (where x is the scope of the command), C, s, S, R

  In the vim editor, this mode is extremely user-friendly. It works almost the way users accustomed to PC-based editors expect.

  You may switch to command mode from insertion mode by pressing ESC.
- **Last line mode:** This mode is used for certain long commands (those beginning with : (colon), / (forward slash), ? (question mark) or ! (exclamation mark)). When you enter the initial character, vi places the cursor on the last line of the screen, where you enter the remaining characters of the command. Press ENTER to perform the command and enter the Interrupt key sequence (Control-C) to cancel it. If you press ESC then vi tries to complete your command on its own and executes it. When ! is used, the cursor moves only after the cursor movement command (or a second exclamation point) has been entered (see *operating on vi's internal buffers* below).
- **Visual mode:** This mode is used for "visually" selecting a block of text for performing an operation on it. The command v switches to the visual mode. Then, the arrow keys may be used to select a block of text. Once the desired text has been selected, an operator is typed. The operator operates on the selected block of text. Common operators used this way are y (for copy), d (for delete or cut) and ! (see *operating on vi's internal buffers* below).

Some common commands

Arrow keys	: Cursor movement
i	: Switch to insertion mode
Esc	: Switch to command mode from insertion mode
Ctrl-L	: Redraw the screen (useful when the screen gets garbled up, particularly during scrolling)
:q	: Quit (when the file has not been changed)
:q!	: Quit forcefully (without saving, when the file has been changed)
:w	: Write (save) the current file
:wq	: Write (save) the current file and quit

:w *filename*	: Write to (save as) *filename*
	Note: After save as…, vi continues to operate on the original file, unlike DOS/Windows based editors that close the original file and open the newly saved file.
^g (CTRL-g)	: Displays the current file's name and size and cursor position
:e filename	: edit (open) the file
a	: Append (cursor moves to the next position and you enter insertion mode)
A	: Append at the end of the line
x	: Delete the current character
X	: Delete the previous character
dd	: Delete (cut) the current line and put it into the buffer
*n*dd	: (where *n* is a number) delete (cut) *n* lines starting from the current line and put them into the buffer
o	: Open (create) a new line after the current line and switch to insertion mode
O	: Open (create) a new line before the current line and switch to insertion mode
J	: Join (Join the next line with this line)
u	: Undo the last command
.	: Repeat the last command
gg	: Go to the first line
G	: Go to the last line
*n*G	: (where *n* is a line number) go to line *n*
0	: Go to the first character position
^	: Go to the first non-white-space character
$	: Go to the end of the line
w	: Go to the beginning of the next word
e	: Go to the end of the current word (next word, if already at the end of the current word)
b	: Go to the beginning of the current word (previous word, if already at the beginning of the current word)
yy	: Yank (copy) the current line to the buffer
*n*yy	: (where *n* is a number) yank (copy) *n* lines starting from the current line to the buffer

p : Put (paste) from the buffer after the current position / line

P : Put (paste) from the buffer before the current position / line

Command line options

+*n* : (where *n* is a line number) opens the file and jumps to line number *n*

-c subcommand : performs the specified ex subcommand(s) before editing begins, multiple commands must be separated by | (the bar character)

-r *filename* : Recovers the file *filename* after an editor or system crash. If you do not specify a file, vi displays a list of all saved files. **Note:** vi automatically saves your file periodically (like the auto save feature in MS Office) to a temporary file. Later on, if the editor or system crashes and you have not saved your file, you can still recover your file through this option.

-R : opens the file in read only mode

-x : prompts for an encryption key and then unencrypts the file. If the file specified is not encrypted or an incorrect key is entered, garbled text is displayed.

The initialization file

If you want some command(s) to run every time you run vi or want to set some vi option permanently, create a file called .vimrc (VI iMproved Run Commands) in your home directory and put your command(s) in that file, one per line. Changes will take effect from the next time you start vi.

For example, if you want vi to display line numbers every time you start vi, you put the following line in your .vimrc file.

:set nu

Similarly, if you want to map the g key to a sort command, apart from enabling line numbers, you may put the following lines in .vimrc.

:map g 1G!Gsort^M

:set nu

Note that the ^M represents the ENTER key. Any nonprintable key like that can be entered by typing Control-V followed by that key.

Note that with the older version of vi, the name of this file was .exrc.

Commands in detail

Note:

the a, A, i, I, o, O, cx (where x represents the scope of the subcommand), C, s, S, and R commands cause vi to switch to insertion mode.

Some commands (those with the prefix : (colon), / (slash), ? (question mark), or !) take you to the last line mode.

Cursor movement:

h	move the cursor one position left
←	move the cursor one position left
j	move the cursor one line down
	move the cursor one line down
k	move the cursor one line up
↑	move the cursor one line up
l	move the cursor one position right
→	move the cursor one position right
ENTER	move the cursor one line down
+	move the cursor one line down
-	move the cursor one line up
*n*ENTER	move the cursor *n* line down
n+	move the cursor *n* line down
n-	move the cursor *n* line up

gg	move the cursor to the first line in the file
G	move the cursor to the last line in the file
*n*G	move the cursor to the n^{th} line in the file
n%	move the cursor to the *n*% position in the file (0% is beginning, 25% is at the $1/4^{th}$ part of the file, 50% is the middle of the file and so on)
Ctrl-g	Displays the current file name, position and total number of lines in the file
0	move the cursor to the first character position in the line

^ move the cursor to the first non-white character on the line (non-white means other than space and tab)

$ move the cursor to the last character in the line

n| move the cursor to the n^{th} character position in the line

w move the cursor to the beginning of the next word

e move the cursor to the end of the current word (next word, if already at the end of the current word)

b move the cursor to the beginning of the current word (previous word, if already at the beginning of the current word)

f*c* here *c* is a character. The cursor is moved to the next occurrence of the character *c* from the cursor (on the current line only). f stands for “find”. The cursor is positioned on the character *c*

F*c* here *c* is a character. The cursor is moved to the previous occurrence of the character *c* from the cursor (on the current line only). F stands for “find”. The cursor is positioned on the character *c*

t*c* here *c* is a character. The cursor is moved to the next occurrence of the character *c* from the cursor (on the current line only). f stands for “to”. The cursor is positioned on the character before *c*.

T*c* here *c* is a character. The cursor is moved to the previous occurrence of the character *c* from the cursor (on the current line only). T stands for “to”. The cursor is positioned on the character after *c*

The four commands above can be repeated with ; in the same direction and , in the opposite direction.

% jump the cursor to the matching parenthesis, square bracket or brace. Thus, you can put your cursor on one of the (,),{,},[,] and press % to take the cursor to its matching counterpart

Creating new lines:

o create a new blank line after the current line, position the cursor at the beginning of it and switch to insertion mode. Stands for “open”

O create a new blank line before the current line, position the cursor at the beginning of it an switch to insertion mode. Stands for “open”

Changing text:

c	operator for Change Text. Can be used as cw (change word), cc (change the whole line) 3cw (change the 3 words starting with the current word), etc. The text to be changed is deleted and vi enters the insertion mode. After typing the replacement text, press ESC to come out of that mode.
r*c*	here *c* is a character. replace (change) the single character at the cursor to the character c. Remains in command mode.
R	replace; enters the insertion mode with overwrite – what you type overwrites existing text until you come out of the mode by pressing ESC.
s	substitute; Change the single character at the cursor, just like the cl command (l is the command to move one character right)
S	change the whole line, just like the cc command
~	change the case of the character at the cursor, if it is an alphabetic character

Deleting text:

x	deletes the single character at the cursor (like DELETE key)
X	deletes the single character preceding the cursor (like the BACKSPACE key)
d	delete (cut). Deletes text and puts it in the buffer for pasting. Can be used as dw (delete word), d$ (delete up to the end of the line), dd (delete the whole line), 5dd (delete 5 lines starting from the current line), etc.
:*range*d	delete the lines specified in the given range of line numbers *range*. The syntax of range is as specified for find and replace.

Cut, Copy and Paste:

d	operator for Delete (Cut). Deletes text and puts it in the buffer for pasting. Can be used as dw (delete word), d$ (delete up to the end of the line), dd (delete the whole line), 5dd (delete 5 lines starting from the current line), etc.
y	operator for Yank (Copy). Copies text and puts it in the buffer for pasting. Can be used as yw (copy word), y$ (copy up to the end of the line), yy (copy the whole line), 5yw (copy 5 words starting from the current word), etc.
p	put (paste). Paste the contents of the buffer after the cursor.
P	put (paste). Paste the contents of the buffer before the cursor.

Named Buffers:

vi has one unnamed buffer, which stored the text that was deleted (cut) or yanked (copied) last. When we put (paste) text, it is put from this buffer,

In addition, vi has 26 named buffers (named a to z) and 10 numbered buffers (numbered 0 to 9).

The numbered buffers 1 to 9 hold the last 9 deleted (cut) texts, in reverse chronological order of deletion (i.e. the text last deleted is in buffer 1, the text deleted before that in buffer 2, and so on). They only store large deletions, where a large deletion is defined as at least 1 line. The numbered buffer 0 always holds the last text yanked (copied), large or small.

The 26 named buffers may be used to hold 26 arbitrary blocks of text that may be put (pasted) anytime we need them.

The command to yank (copy) text to a named buffer is

"***b****yank-command*

*where **b** is the single-character name of the named buffer (a-z) and yank-command is any yank (copy) command, like 3yy, 5yw, etc.*

The command to put (paste) text from a named/numbered buffer is

*"***b***p*

or

*"***b***P*

where ***b*** *is the single-character name/number of the buffer (a-z/0-9). p and P put the text after and before the cursor respectively.*

Marks:

A mark in vi is used to mark a position in the file. vi supports up to 26 marks in a file. These marks a named with a single lower case alphabet (a-z). We may jump to any of the marked positions at any time.

The command to put a mark at the cursor location is

m***a*** where ***a*** is the single-character name of the mark (a-z).

The command to jump to a mark is

'***a*** where ***a*** is the single-character name of the mark (a-z).

Note that this is a cursor movement command and can be used as such in combination with operators. E.g. y'a copies the text from cursor position up to mark a. d'z deletes the text from cursor position up to mark z.

Find and replace:

Note: patterns use regular expressions as described below.

/*pattern* search for the pattern *pattern* in the forward direction

?*pattern* search for the pattern *pattern* in the reverse direction

n repeat the previous search in the same (forward/backward) direction as the original search

N repeat the previous search in the opposite (forward/backward) direction as the original search

Search Command	**n**	**N**
/ (forward search)	Forward	Backward
? (backward search)	Backward	Forward

:*rang***e**/*pattern*/replacement*/*

s stands for substitute. substitute only the first occurrence (on each line) of pattern *pattern* with pattern *replacement* in the range of lines specified by *range*.

:*rang***e**/*pattern*/*replacement/g*

substitute all the occurrences of pattern *pattern* with pattern *replacement* in the range of lines specified by *range*. g stands for global.

Range can be specified as follows.

- When no range is specified, the current line is implied.
- single_line_specification
- starting_line_specification,ending_line_specification
- % stands for all the lines in the file (equivalent to 1,$)

Line specification:

- lines may be specified using a line number.
- . (dot) in line specification represents the current line.
- $ in line specification represents the last line in the file.
- One can also use arithmetic like .+5, .-5, $-5 in line specifications.
- Lines can also be specified as /*pat*/, which is interpreted as the next line containing the pattern *pat*.
- When one or more patterns are used in line number(s), the operation is applied only to the first range meeting the criteria. If the range is preceded by the g (global) flag, the operation is applied to all ranges in the file meeting the criteria.

Examples of the use of ranges: (: has been added to avoid possible confusion)

:7	the 7th line
:/try/	the first line after the current line containing try
:g/try/	all the lines containing try
:5,10	the lines from line number 5 up to line number 10
:1,$	all the lines in the file
:%	all the lines in the file
:1,.	from the 1st line up to the current line
:.,$	from the current line up to the end of file
:.,.+9	10 lines starting from the current line
:.,$-5	from the current line to the sixth line from the end
:/try/,30	from the first line after the current line containing *try* up to line number 30
:30,/try/	from line number 30 up to the first line after the current line containing *try*
:/try/,/catch/	from the first line after the current line containing *try* up to the first line after the current line containing *catch*
:g/try/,/catch/	all the ranges of lines from the line containing *try* up to the line containing *catch*

Regular Expressions:

.	(dot) Any single character
*	Previous element 0 or more times
^	An imaginary anchor at the beginning of line (only if ^ is at the beginning of the pattern)
$	An imaginary anchor at the end of line (only if $ is at the end of the pattern)
\	Escape character - removes the special meaning of the next character; also adds special meaning to some characters in some circumstances as described below
[abcd]	Any one character from a, b, c or d
[a-z]	Any character between a and z, inclusive

[a-z0-9] Any one character from between a and z, inclusive or between 0 and 9, inclusive

[a-zA-Z0-9] Any one character from between a and z, inclusive or A and Z, inclusive or between 0 and 9, inclusive

[a-z0-9ABCD]
Any one character from between a and z, inclusive or between 0 and 9, inclusive or A, B, C or D

[^a-z0-9] Any one character other than between a and z, inclusive or between 0 and 9, inclusive

[a-z^0-9] Any one character from between a and z, inclusive or ^ or between 0 and 9, inclusive

\w – equivalent to [a-zA-Z0-9] (alphanumeric)
\W – equivalent to [^a-zA-Z0-9] (non-alphanumeric)

Word matching:

\< An imaginary anchor at the beginning of every word
\> An imaginary anchor at the end of every word
\<*pat1*\> The input matches with the pattern \<*pat1*\> only if the input matches with pattern *pat1* and is a whole word (not part of a larger word)

\+ - Previous element 1 or more times

\? - Previous element 0 or 1 time

Repetition:

\{*m*\} in a pattern means the previous element *m* times.
\{*m*,*n*\} in a pattern means the previous element between *m* and *n* times.
\{*m*,\} in a pattern means the previous element *m* or more times.

Subexpressions (Tags) and Back References :

Any part of the search pattern enclosed between \(and \) is treated as a subexpression/group (considered to have been tagged). The exact input that matches with a subexpression is remembered and can be referenced later as *n*, where *n* is the subexpression number (you can have multiple subexpressions in your search pattern and they are

numbered serially starting from 1, so \1 means the input that matched with the first subexpression, \2 means the input that matched with the second subexpression and so on).
& in replacement pattern stands for the whole input that matched the search pattern.

Undo / Redo:

u	undo the effect of the last command (multi-level redo is supported, just press the key as many times as you want)
U	restore the last edited line to its original status, discarding all changes (pressing U again undoes the action of the previous U). This only works if we have not moved away from the line.
Ctrl-r	redo the actions undone earlier (multi-level redo is supported, just press the key as many times as you want)

File handling:

:w	Write (save) the current file
:wq	Write (save) the current file and quit
:r *filename*	read the whole file *filename* at the cursor location
:w *filename*	Write to (save as) *filename*
	Note: After save as…, vi continues to operate on the original file, unlike DOS/Windows based editors that close the original file and open the newly saved file.
:q	quit vi, without saving the file (only works when the file has not been modified)
:q!	quit vi forcefully. If any changes were made to the current file, they will be lost.
:n	switch to the next file when vi is opened with multiple files. The current file must have been saved.
:n!	switch forcefully to the next file when vi is opened with multiple files. The current file , if it was modified, will not be saved.
:rew	switch to the first file when vi is opened with multiple files. The current file must have been saved if it was modified.
:rew!	switch forcefully to the first file when vi is opened with multiple files. The current file, if it was modified, will not be saved.
:e *filename*	open the file *filename* for editing. The current file must have been saved if it was modified.

:e! *filename* open the file *filename* forcefully for editing. The current file, if it was modified, will not be saved.

:e# open the file that was opened in the current vi session immediately before the current file. The current file must have been saved if it was modified. The command can be used repeatedly to switch between two files. Adding an ! after the command will switch to the other file forcefully, discarding any changes to the current file.

:e! discard all changes made to the current file, and reload the last saved version from the disk.

Running external commands:

:!*command* open a shell, run the command *command*, and return to vi when the user presses ENTER. You may pass arguments to the command as well. % can be used as an argument to pass the current filename. The output of the command is displayed on the screen until you press ENTER.

:!sh run a shell. Returns to vi when the user exits from the shell.

:r !*command* run the command *command* and read its standard output at the cursor location

:w !*command* run the command *command* with the current file as its standard input

Operating on vi's internal buffers:

vi provides the most unusual and powerful feature of letting any external command operate on its internal buffers.

!*cursor-movement-commandexternal-command*ENTER

-- or –

:*range*!external-commandENTER

The external command *external-command* is executed and the text covered by the cursor movement that would have been effected by the cursor movement command *cursor-movement-command* is passed on to the command as standard input. The standard output of the command replaces the text in vi.

If the cursor movement command is omitted and a second ! Is typed, the current line is used.

Example:

!5+sortENTER

OR

:.,.+5!sortENTER

The external command *sort* is executed and the text of the current line and the next 5 lines is passed on to sort as standard input. The standard output of sort replaces the text in vi. In effect, the current line and the next 5 lines in vi's buffers get sorted using the external command sort without ever leaving vi. We may also pass arguments to sort.

It is important to note that when the user presses !, there is no change in display. Then, when the user presses a cursor movement command, ! followed by the line range followed by another ! Appear in the last line and the cursor is positioned at the end of it. The user then types the external command and presses ENTER to execute the command.

Example-2:

:1,15!sort

The external command *sort* is executed and the text of the lines numbered 1 to 15 is passed on to sort as standard input. The standard output of sort replaces the text in vi.

Example-3:

:1,15!sort -n -k2,2

The external command *sort* is executed with the arguments -n and -k2,2 and the text of the lines numbered 1 to 15 is passed on to sort as standard input. The standard output of sort replaces the text in vi.

Inserting Special Characters:

If you want to use a character that is interpreted by vi itself (e.g. ENTER) in any mode, press CTRL-V followed by that character.

Abbreviations:

abbreviations are short forms that are automatically expanded by vi when entered.

Abbreviations can be defined for different modes of vi. An abbreviation is expanded when a non-keyword character (like ESC or SPACE or ENTER) is typed.

:ab *abbreviation* full-formENTER

defines an abbreviation *abbreviation* and its full form *full*-form for all modes. *The full form can consist of any number of word.*

:iab *abbreviation* full-formENTER

defines an insertion mode abbreviation.

Examples:

:iab sout System.out.printlnENTER

:iab spuvvn Sardar Patel University^VENTERVallabh vidyanagarENTER

Mapping Keys:

A used or unused key sequence in vi can be mapped to another key sequence using the :map command.

:map *key-sequence new-key-sequence*

Now, if you press *key-sequence*, vi behaves as if you pressed *new-key-sequence*.

This facility can also be used like macro or for abbreviations.

Window Management:

vi supports multiple text mode windows.

Splitting a window into multiple windows (with the same file open in both):

:sp	horizontally splits the current window into two windows
:vs	vertically splits the current window into two windows
^w*arrow-key*	switch to the window in the direction of the arrow key *arrow-key*
:new	open a new window with a blank document in it
^wq	close the current window

Obtaining help:

:help opens a built-in extensive help document in a new window in vi. You may close the window by pressing ^wq.

:help*vi-command*

opens the built-in help document to the point where the vi command *vi-command* is explained in a new window in vi.

set options:

vi has several options that can be set to customize the user's vi experience. These options are accessed using the :set command in the command mode. There are two types of options. The Boolean type options can be enabled or disabled (set to true or false). To set (enable) this type of option use *:set*

optionname. To clear (disable) such option, use :set **no***optionname*. The other type of options can be set to a value using :set *optionname*=*value*.

:set all	displays all options with their current values
:set *optionname*	if the option *optionname* is Boolean type then sets (enables) the option. If it is the value type then displays the current setting of the option.
:set **no***optionname*	clears (disables) a Boolean type option.

For example, :set number (or :set nu in short) enables the display of line numbers in the file. :set nonumber (or :set nonu in short) removes this option. As another example, vi, by default, displays a message at the bottom of the screen when you yank(copy), put(paste) or delete 5 or more lines. If you want such messages for 2 or more lines instead, than you can issue the command :set report=2.

:set ic
: set the ignore case option for search and find and replace

:set noic
: set the no-ignore-case option for search and find and replace

:set tabstop=4
: set the tabstops at every 4th character position

:set mouse=a
: enable mouse usage in vi

:set nobackup
: do not create a backup file when the file is saved

:set noswapfile
: do not autosave to swap file

:set hlsearch
: highlight all matches for the last search

:set incsearch
: perform incremental search

:set ignorecase
: perform case-insensitive search

:set tabstop=4
: set tab stops at every 4th character positions

:set history=100
: save last 100 commands in history

:set expandtab
: convert tabs into spaces

:set shiftwidth=4

set the size of each shift in indent/unindent operations

Comparing files:

Typing the command

vimdiff file1 file2

at the prompt opens the two files in two vertical windows inside vi, with the differences between the two files highlighted. This is very useful for comparing two similar files.