VI(Visual) Editor Reference manual

The vi is a text editor. It is small, powerful, and standard on most UNIX systems. The vi often frustrates new users with a unique distinction between its two modes: Command Mode and Insert/Overtype Mode. This distinction, although difficult to become accustomed to for many users, provides great power and ability to the editor. Insert/Overtype Mode is designed for inserting text only. All text manipulations and cursor moving should be done from with in Command Mode.

1. Entering the vi

prompt$ vi

prompt$ vi file1 file2 ...

prompt$ vi -r file #Recover file from crash

prompt$ vi +string file #Execute ex command "string"

prompt$ vi @rcfile #Read commands from rcfile

Insert/Overtype Mode

Insert/Overtype Mode is solely for entering text. To leave one of these two modes press the [ESC] key. if you wish to enter the ESC character or any other control character while in insert mode: type [CONTROL]-V and then the control sequence. The only difference between Insert Mode and Overtype Mode is that characters are placed in front of the text after the cursor in Insert Mode, where as existing characters are overwritten in Overtype Mode.

2. Command Mode

Entering Insert Mode

a -- append text, after the cursor

i -- insert text, before the cursor

R -- enter Overtype Mode

A -- append text, after end of line

I -- insert text, before first non-whitespace character

o -- open new line below cursor in Insert Mode

O -- open new line above cursor in Insert Mode

vi Syntax

vi commands follow the general form:

n operator m object

which means:

execute operator n times on m objects. If n and/or m are omitted, they default to 1.

Operators which take objects are(if the operator is pressed twice then the object is the current line)

c -- Change

d -- Deletion

"cy -- Yank, if "c is omitted, uses general buffer.

< -- shift lines left by shiftwidth variable

> -- shift lines right by shiftwidth variable

!cmd -- filter trough cmd

#The operators <, >, and ! are line based so the set of objects is diminished greatly.

Operators which do not take objects:

s -- Substitute

x -- Delete character

r -- Replace character

~ -- change case of character

Objects (if given with out an operator are interpreted as a cursor motion command):

w -- forward until beginning of word

e -- forward until end of word

b -- backward until beginning of word

$ -- forward until end of line

^ -- backward until first non-whitespace character

0 -- backward until first column of line

nG -- line number n (default: $, i.e. last line of file)

n| -- column n of current line

/pat -- forward until beginning of pat, search

?pat -- backward until beginning of pat, backward search

n -- repeat last search

N -- repeat last search/backward search, but in opposite direction

% -- until match of parenthesis, brace, or bracket tc -- until next appearance of c on current line

Tc -- backward until next appearance of c on current line

fc -- until and including next appearance of c on current line

Fc -- backward until and including next appearance of c on current line

; -- repeat last f, F, t, or T

, -- repeat last f, F, t, or T in reverse

} -- forward until end of paragraph

{ -- backward until end of paragraph

) -- forward until end of sentence

(-- backward until end of sentence

]] -- forward until end of section

[[-- backward until end of section

nH -- n lines before first line on screen; n defaults to 0

nL -- n lines before last line on screen; n defaults to 0

M -- the middle line of the screen

j -- down one line

k -- up one line

h -- left one character

l -- right one character

[BS] -- left one character, backspace usually equals ^H

[SPACE] -- right one character

_ -- the entire current line

- -- until first non-whitespace character on previous line

+ -- until first non-whitespace character on next line

[RETURN] -- until first non-whitespace character on next line

Miscellaneous 1

u -- undo last change

U -- undo entire line

"cp -- put "c or general buffer after the cursor

"cP -- put "c or general buffer before the cursor

mc -- set mark with character c

`c -- goto mark c

'c -- goto beginning of line with mark c

`` -- return to position before mark jump or search

'' -- return to beginning of line before mark jump or search

J -- join two lines

D -- delete rest of line

C -- change rest of line

Y -- yank current line into general buffer

& -- execute last ex-style substitution

. -- execute last modification

! object command -- send object as stdin to command and replace with stdout

[Ctrl]-G -- print information about file

: map x y -- when character x is pressed, execute y

: map! x y -- map input mode character x to string y

: ab x y -- x is an abbreviation for y, changes are made on the fly

: su -- Suspend the current editor session

: sh -- run a shell

ex Commands

ex syntax

ex commands in the vi follow this general form:

: address command

which means:

Execute command on specific lines obtained from the address part of the general form. If address is omitted, current line is used. Keep in mind that the ex is a line based editor, so all actions are line based.

addresses:

% -- all lines in file

x,y -- lines x to y

. -- current line

n -- line number: n

$ -- last line of file

x-n -- n lines before line x

x+n -- n lines after line x

/pat/ -- forward to line containing pat

?pat? -- backward to line containing pat

Some commands are:

s/pat/text/ -- substitute 1st match of pat with text

s/pat/text/g -- substitute every match of pat with text

s/pat/text/n -- substitute the nth occurrence of pat with text

ya c -- yank into buffer c or the general buffer if c is omitted

g address cmd -- execute cmd on all lines which satisfy address

> -- shift right

< -- shift left

d -- delete line

! UNIX-cmd -- execute UNIX-cmd on line

m address -- move lines to address

3. **refer to ex manual page for more commands**

The vi environment variables

set

You can customize your environment with this command by typing set var=value, this will set the specified var to value for a scalar variable. For boolean variables, use set var to set and set novar to unset. You can see which variables are set by just typing the set by its self. You can see a list of all variables by typing set all. Some environment variables are specific to the ex editor and some are specific to the vi editor. I have included both.

boolean variables:

autoindent(ai) -- begin editing next line at same level of indent-ion as this one.

autowrite(aw) -- write current buffer before leaving

exrc(ex) -- tells vi/ex if it should read the .exrc file in the current directory.

errorbells -- editor sends a beep to the terminal when an incorrect

ignorecase(ic) -- ignore case of characters in searches.

list -- place a $ at the end of each line and a ^I on each tab.

magic -- allow ., [, and * to be interpreted as special characters in RE's.

number(nu) -- number lines in left margin

showmatch(sm) -- when closing a paren., brace or bracket; move the visual cursor to opening item to check scope

showmode(smd) -- show type of insert mode

wrapscan(ws) -- when searching and at bottom of file, continue searching from the top

scalar variables:

shiftwidth -- number of spaces to to insert on a shift operation

4. File Saving and Loading

: wq -- write file and quit

: w -- write file

: w file -- write to specified file

: w! -- overwrite existing file

: e file -- edit new file

: r file -- put contents of file

: q -- quit the editor

: q! -- force quit the editor, do not save changes

: x -- quit the editor, save file if it was modified

ZZ -- quit the editor, save file if it was modified

: n -- start editing next file in list

: rew -- rewind file list, start editing 1st file on argument list again

Q -- quit vi and enter ex

: pre -- Preserve file.

: rec file -- recover file

5. Examples 2

j -- move cursor down

k -- move cursor up

h = [BS] -- move cursor left

l = [SPACE] -- move cursor right

+ = [RETURN] -- first non-whitespace character on next line

cw -- change word

dd = d_ -- delete line

yy = y_ -- yank current line into the general buffer

"ayj -- yank current line and one below into buffer a

yfc -- yank until next occurrence of c on current line into the general buffer

3dl = d3l -- delete next 3 characters

4c(= 2c2(= c4(-- change next 4 sentences

>% -- while on a brace, paren., or bracket; shift right until closing brace, etc.

:%!sort = :1,$!sort -- sort current file

:5,10s/foo/bar/2 -- change the second occurrence of foo with bar on lines 5-10

:map g 1G -- map g to really run 1G

3J -- Join next 2 lines to current one

3,9m$ -- move lines 3 through 9 to the end of the file

ab w/o with out -- when w/o is typed change to with out

:?foo?,/bar/d -- delete from the reverse match of foo until the next match of bar

g/{/,/}/< -- shift all lines between, and including, a "{" and a "}" left

:$-4,$d -- delete last five lines of buffer

:%s/^\(.*\) \(.*\)$/\2 \1/ -- swap everything before and after the first space

----

1 It is noteworthy to add that most control sequences are bound in the vi. I do not mention them here because they remind me of emacs and I hope to spare you such pain.

2 For the record, no animals were physically harmed during the testing of these examples; although some elephants are now in psychological therapy as a direct result of my actions. I kind of feel guilty about that one :(
