Advanced Vim Tutorial

Sunil Mohan Ranta
smr [at] students.iiit.ac.in

1 Introduction

Vim is not just an editor. It can be an IDE if used at its best. I have found it more convenient to use than any other word processing tool i have used. With proper knowledge of the countless features provided by vim, one can be much more efficient in whatever editing job, and programming in my case.

Every time i wished for a feature, i found it there in vim. I didnt allow myself to do anything the stupid way, and always figured out a better way to do the same thing. Thats how i have got a decent expertise on vim. With this tutorial i wish to give you a feel of what all is possible with vim, and show you a direction to learn more.

Before i proceed, i would like to ask you a question. Suppose you have got a file with about 50 lines only, and you have to make few changes to the file. Lets assume that you are asked to add 10 to every number present in the file. Lets assume that there are only 25 numbers in the file and rest is text. Would you start doing it the stupid way or search for a efficient way to do it ? Most of the times one will think that its just 25 numbers, and without thinking about a better way to do it, starts editing it right away. Or even if one does want to do it a better way he doesnt do it thinking that figuring out the better way will take much more time than doing it the obvious way. But what we always ignore is that its not just this time. We do it many times, and thats why it makes sense to go for the better way to do it, even if it takes more time at first time.

So next time when the thought "its just this much" comes to your mind, just remember that "its not just this time" , and the choice is yours.

If we go by Brook's words "The tool that save the most labor in a programming project is probably a text-editing system" [The Mythical Man-Month, proposition 12.10], then we should probably work on our editing skills more than our programming skills.

In this tutorial I assume the reader to have a basic knowledge of vim. Basic features like editing, movement, searching, replacing, opening, saving

etc are not covered in this tutorial. I'd recommend going through `vimtutor` for basic understanding of vim.

Remember : **"if its just this much, its not just this time"**

2 Shortcuts

easily and efficiently execute more complex commands and statements.

2.1 Map

map is used to change the meaning of sequence of keystrokes, usually to perform a job using easier keystrokes, which would otherwise need more complex keystrokes. Sometimes simple but frequently used keystrokes are also mapped.

- o map can be used for
  - changing meaning of typed keys
  - execute a function on certain keystrokes
- o mapping can be set for one or more of following modes using the commands written against them.

normal	`nmap` , `map`
visual	`vmap` , `map`
insert	`imap` , `map!`
command-line	`cmap` , `map!`
operator-pending	`omap`, `map`

- o two forms of map
  - `map`
  - `noremap`

    to avoid recursion
- o `unmap`

  remove the mapping
- o examples

`map` <C-S>	:w<CR>
`imap` <F3>	<Esc><Right>n<Insert>

see `:help mapping`

2.2 Abbreviations

change the meaning of a word. Can be used in insert , replace and command line mode.

Abbreviations can be used for

- typos correction

  `iab` tihs this

- making abbreviations

  `iab` smr@ smr [at] smr.co.in
  `iab` #i #include

see `:help iab`

3 Windows, Buffers and Tabs

Buffer is the vim's copy of the file you are editing. Window is a viewport onto a buffer. Tab holds windows.

3.1 Window Management

`:split` or C-W s
: split the active buffer horizontally

`:vsplit` or C-W v
: split the active buffer vertically

`:new` or C-W n / :vnew
: open a new buffer (empty)

other commands
: all other commands work as they use to for single window therefore `:w` for saving and `:q` for quitting

-o argument to vim
: start vim with multiple windows each opening the files supplied to -o

Navigation through windows
: C-W w / C-W W / C-W C-W
  - iterate through the windows (go to next/previous window)

see `:help windows`

3.2 Buffer Management

`:e` <file>

open new buffer (of the <file> supplied)

`:ls`

show current buffers

`:b` <num>

switch to buffer <num>

`:bdelete`

unload buffer

`:bwipeout`

unload buffer and deletes it

see `:help buffers`

3.3 Tabs Management

Tabs are available only in version 7 or later. Tabs provide a traditional(like in other GUIs) tabbed view over the vim windows and buffers. If mouse is enabled in vim, tabs can be opened, closed or selected using mouse clicks. Tabs shares registers, and therfor allow deleted/yanked text from one tab to be pasted on others. Also undo history of every tab is maintained separately, which allows switching between the tabs without loosing the changes history. This is not possible in buffers. Also buffers does not allow switching between buffers without saving the current buffer, which is now possible between tabs.

`:tabe` <file>

Open <file> in new tab. Opens empty tab if no file is supplied

`tabs`

List opened tabs along with windows/buffers they contain

`tabc`

Close the active tab

`tabn` and `tabp`

Go to next(on tabn) or pervious(on tabp) tab
<Ctrl-PageDown> and <Ctrl-PageUp> also works

`CTRL-W gf`

Open the filename under cursor in new tab

see `:help tabpage`

4 Visual Mode

Visual Mode provides us flexible and easy way to select text and execute command (operator) on it.

see `:help Visual`

4.1 Visual Block

Rectangular block of selected text
invocation key : CTRL-V

4.2 Visual Line

Text is selected line-wise
invocation key : V (ie Shift-v)

4.3 Visual (Char)

Text is selected char-wise
invocation key : v

4.4 Role of Mouse in visual mode

setting mouse to 'n' or 'a' allows selecting visual area using mouse click and mouse drags

4.5 Select Mode

select mode is similar to selection mode in Microsoft Windows Notepad. Typing any printable key replaces the selected text with the key.

Invocation keys : gh / gV / gH / G CTRL-H

see `:help Select`

4.6 Important keys

`gv`	select previous visual area again
`CTRL-G`	switch between Visual and Select
<	shift selected text toward left
>	shift selected text toward right

`d or y`	Delete or Yank selected text
`I or A`	Insert or Append text in all lines (visual block only)

5 Advanced Navigation Techniques

5.1 Change-list jumps

helps in iterating through recent cursor positions (where changes were made)

`g;`	go to the previous position of cursor in change list
`g,`	go to the next position of cursor in change list

see `:help changelist`

5.2 Marks

Marks allow us to label cursor position. It can be used to label import locations in file for easy navigation.

`m<char>`	set label <char> to current cursor position
`‘<char>`	go to position labeled <char>

see `:help mark-motions`

5.3 Tags

Tags are also labels used for jumping through them. Tags are read from the tags file.

Important keys

`:tag` <tagname>	position the cursor on tag <tagname> position
`CTRL-]`	position the cursor on tag under the cursor
`CTRL-T`	go back after the previous jumps
`:ptag` <tagname>	show tag in the preview window (see :help preview-window)
`CTRL-W }`	show tag under cursor in preview window
`CTRL-W z`	close preview window (opened by above 2 commands)

`ctags` *.cpp *.h

generates tags using GNU/Linux command `ctags` (see `man ctags`)
this command will create a file named tags with all tags found in files supplied

see `:help tags`

5.4 Mouse

enable use of mouse in vim, for scrolling or visual selection

`:set mouse`=n/v/i/c/h/a/r/A

enable mouse for the corresponding mode (normal/visual/insert/etc..)
eg :
`:set mouse`=nvi enables mouse in normal,visual and insert mode
`:set mouse`=a enables mouse in all modes

see `:help mouse`

* although mouse and arrow keys can be helpful sometimes, but vim users are advised not to use mouse or arrow keys. It helps gain more efficiency over the keyboard.

6 Repeating commands

Repeating capability of vim allows us to execute a command or set of keystrokes multiple times without any pain.

6.1 Repeat Last Action

.(dot) Repeat last change
@: Repeat last command

see `:help single-repeat`

6.2 Recording

Record keystrokes and playing them again.

q<char>	start recording characters into register <char>
q	stop recording
@<char>	play keystrokes recorded into register <char>
@@	repeat last recording

see `:help record`

6.3 Select and execute

Run command on selected lines only

`:<range>g/<pattern>/<cmd>`

runs command <cmd> on all lines in <range> where <pattern> matches
g! or v works on NOT of above. That is execute <cmd> where <pattern> doesnt match.

see `:help multi-repeat`

7 Registers

Registers are used to store the deleted, yanked text and recorded statements.
Registers are named single char from the list a-zA-Z0-9.

a-z	named registers. These are used used only when explicitly specified
A-Z	appends to the lowercase register
0-9	filled with deleted lines. Shifted on new entries.
_(underscore)	black hole register. Use when u don't want to change the value of any register.

`:reg`	shows contents of all registers
<reg><action>	action is stored in register <reg>. <action> can be yank or delete.
<reg><paste>	paste the contents of register <reg>.

see `:help registers`

8 Folds

folds are useful for providing readable and easyily navigable view of the file.

8.1 Fold Methods

decides the criteria for creating folds.

foldmethod can be set to

manual	manually created
indent	based on indentation. Useful for python or other indented codes
marker	based on foldmarker

8.2 Folding Commands

zf	create fold
zo/zO	open fold
zc/zC	close fold

`:set foldmethod`=manual/indent/marker
`:set foldmarker`=, or <,>

see `:help Folding`

9 Functions

Sometimes what we want to do cannot be implemented with commands or keystrokes. We need to define our functions and call them. Ex :

```
function! MyCompile()
    let choice = confirm(Compile method , \&make\\n\&g++ \%.cpp )
    if choice == 1
        exec :make
    elseif choice == 2
        exec !g++ \%.cpp
    else
    endif
endfunction
```

see `:help function`

10 Compiling and Executing

vim helps in edit-compile-edit cycle with its compiling and executing features called quickfix commands.

`:make` runs makeprg

makeprg is set to make by default but one can change it to anything
eg `:set makeprg`=g++\ main.cpp or `:set makeprg`=g++\ %

`:cnext`	show next compilation error
`:cprevious`	show previous compilation error
`:copen`	open errors window
`:cclose`	close errors window

Now you can use the following mappings -

```
:map <F5>  :mak<CR>
:map <F7>  :cn<CR>
:map <F8>  :cp<CR>
:map <F6>  :!./a.out<CR>
```

and compilation and execution become becomes as easy as single key press. And more importantly we are able to iterate through errors more easily, and fix them without any pain.

see `:help quickfix`

11 Filters

Filter allows us to run an external command as filter on the text specified by range or visual selection. The selected text is changed inplace.

```
:<range>!<filter-cmd>
:<selectoin>!<filter-cmd>
```

see `:help complex-change`

12 Plugins

Vim allows to use plugin scripts. Plugins are loaded everytime vim starts. plugins can be placed in 'runtimepath/plugins/' (generally .vim/plugins/).

vim.org hosts a huge repository of vim plugin scripts at http://vim.org/srcipts/.

A few useful plugins scripts are -

taglist (source code browser) [http://www.vim.org/scripts/script.php?script_id=273]
minibufexlporer [http://www.vim.org/scripts/script.php?script_id=159]
vimspell [http://www.vim.org/scripts/script.php?script_id=465]
template file loader [http://www.vim.org/scripts/script.php?script_id=198]

see `:help plugin`

13 Misc

This section lists other misc features

13.1 Sessions

Sessions allows you to save your buffers, windows, tabs and settings of current vim session to a file for restoring at later time.

`:mksession <file>`	save session to <file>
`$ vim -S <file>`	restore session from <file>

see `:help session`

13.2 Undo Branching

Vim has simple undo (key: `u`) and redo (key: `Ctrl-R`) functionality. Starting from version 7.0 vim has also added undo branching. Undo branching happens when you undo a few levels and make new changes. These branches are not reachable with simple undo/redo functionalities. Undo branching support of vim makes these branches reachable.

`:undol`	Show the Undo Branches (the leaves of these branches)
`:earlier`	Go to older state. Time or steps can be specified.
`:later`	Go to newer state. Time or steps can be specified.

see `:help undo-branches`

13.3 Spell Checking

Spell checking can be enabled using the command `:set spell spelllang=en_us`. We also set the language to use. US English in this case. Spell checking is turned off using `:set nospell`. Spell checking is also a feature added in version 7.0.

Important keys -

`]s` or `[s`	Move to next (`]s`) or previous (`[s`) misspelled word
`zg`	Add word under cursor as good word
`z=`	Suggest corrections for the word under cursor

see `:help spell`

13.4 Highlights

Set the foreground and background color and style of distinguished text

`:hi`
– show current highlight settings
`:hi Comment cterm=bold ctermfg=white ctermbg=black`
– changes the hilighting settings for Comments

see `:help hi`

13.5 Autocommand

specify commands to be executed automatically for when reading or writing a file , entering or leaving buffer etc. (Goes in .vimrc)

example :
autocmd BufRead *.c,*.h,*.cpp,*.cc setlocal cindent

see `:help autocommand`

13.6 Incrementing and Decrementing integers

CTRL-A increment number under cursor
CTRL-X decrement number under cursor

13.7 Numerical Arguments

Most of the vim commands take numerical arguments. These arguments are set by typing the number before typing the command.
Eg: 8dd , 2K , 7yy, etc..

13.8 Useful settings

better visualization of tabset	: `set list` , `set listchars`=tab:. ,trail:.s
in place search	: `set incsearch`
keep cursor away from top/bottom	: `set scrolloff`=<num> (4 is good)
always show a status bar	: `set laststatus`=2

13.9 Others

ga	Show ASCII value of character under cursor
gf	Open file under cursor
g<Ctrl-g>	Count words in the file
<<	Shift lines to left
>>	Shift lines to right
`:!cmd`	Run command on terminal
`vimdiff`	show difference between files (shell command)

13.10 Fun

```
:help!
:help 42
:help holy-grail
:help quotes
```

14 Appendix

14.1 Special Key Names

<Up>	Up arrow key
<Down>	Down arrow key
<Right>	Right arrow key
<Left>	Left arrow key

<Home>	Home key
<BS>	BackSpace
<Space>	Space key
<Tab>	Tab
<C-char>	Ctrl + char
<S-char>	Shift + char
<C/S-@>	Ctrl/Shift + Space (exception)

14.2 Important keywords

%	current file name
%<	current file name without extension
<cword>	word under the cursor
<cWORD>	WORD under the cursor
<cfile>	path name under the cursor
<cfile><	idem, without extension
<Leader>	set using mapleader, / by default

14.3 Emacs

Another equally good editor. GNU & RMS product. Lisp based.

Emacs vs Vim

its your call.
But let the reason be more than geekyness and blind support of GNU.

Bottom line : If unlike me you find Ctrl-X Alt-Y Shift-Z natural way of editing, there is no reason you shouldnt use it.

Emacs Tutorial

Ctrl-h t	emacs turorial. Good enough to get you started.
Ctrl-x u or Ctrl-/	undo
Ctrl-s	search
Ctrl-h c <key>	show help on <key>
Ctrl-h k <key>	show more help on <key>
Ctrl-h f	describe a function
Ctrl-h a	show command apropos

Ctrl-x 1	close all windows but active
Alt-x	execute a command by its name
Ctrl-g	quit current command. Return to normal mode.
Ctrl-x Ctrl-c	quit emacs. Dont enter emacs without remembering it.

15 References

http://www.vim.org/

http://www.rayninfo.co.uk/vimtips.html