A Quick Guide to Vim

Why read this?

The objective of this guide is to teach you how to use `vim` effectively. In this module, you will be writing your programs with `vim` on Sunfire (UNIX) server. It can be very frustrating to use `vim` at first. However, after some time, most of you would be impressed by its convenience and finally, fall in love with it. 

Learn `vim` well, and you'll be much more productive when writing code, allowing you to focus more on the logic of your program. Declined to put in reasonable effort of learning at least some of the shortcuts of `vim`, you'll end up spending your precious PE time scrolling around your code...

Starting Vim

To edit an existing file or start a new file, type **`vim filename`** (e.g., **`vim box_volume.c`**).

Using Vim

There are two modes, **Command Mode** and **Insert Mode**.

(1) Command mode is used for moving around the file quickly and to make use of the huge number of commands that make `vim` such a powerful text editor.
(2) Insert Mode is used just for typing in text.

All commands listed below are to be used in Command Mode and are *case sensitive*; for example, G means shift+g.

Inserting Text

When you open a file using `vim`, by default you are in the Command Mode. To switch to Insert Mode, press **`i`** or **`a`**. "-- INSERT --" will then appear at the left bottom of the screen.

Pressing **`i`** (for insert) will put the cursor before the currently highlighted position, while pressing **`a`** (for append) will put the cursor after the currently highlighted position.

To return from Insert Mode back to Command Mode, just press the **ESC** key on the top left corner of your keyboard.

Moving around in your program

There are two options here:

(1) Use the arrow keys on your keyboard

(2) Use the **`hjkl`** keys. (**`h`** -- left, **`j`** -- down, **`k`** -- up, **`l`** – right)

Other movement keys

`W` -- Move to the next word

`b` -- Move to the previous word

`^ or 0` -- Move to the start of current line

`$` -- Move to the end of current line

`%` -- Jump to a matching bracket (e.g., jump from an opening bracket { to its matching })

`gg` -- Jump to the start of the file

`G` -- Jump to the end of the file

☆ **`<line number>G`** -- jump straight to a certain line (e.g., 5G will jump to the fifth line)

`H` -- Jump to the top of the screen

`M` -- Jump to the middle of the screen

`L` -- Jump to the bottom of the screen

Move and change to insert mode

☆☆ **`i`** -- Insert before cursor

`a` -- Insert after cursor (append)

`I` -- Insert at the start of the line

`A` -- Append at the end of the line

☆ **`o`** -- Insert below the current line

`O` -- Insert above the current line

The key to learning `vim` commands is to use them - when you use them, you will remember them. Read through this article just give you a feel of what `vim` can do, but DON'T try to memorize all commands just verbally!

Keep this article with you as a reference in case of need!

Deleting (cutting) text

(Pressing the d key followed by a movement key deletes text in that direction.)

`dj` -- Delete current line and the line below

`dh` -- Delete character to the left

`dw` -- Delete word from cursor to the end of word

`d$` -- Delete from cursor to the end of line

`d0` -- Delete from cursor to the start of line

☆☆ **`dd`** -- Delete current line. This puts the line in buffer, so that you can paste it somewhere later.

☆ **`x`** -- Delete the character under the cursor.

Undo and Redo

☆☆ **u** -- Undo your last action

ctrl + r -- Redo your last undo

Save your work

☆ **:w** -- Save the file you are working on.

☆☆ **:wq** -- Save your work and exit `vim`.

:q! -- Quit `vim` without saving your modification.

Copy and paste

☆ **yy** -- Yank (copy) 1 line

<digit>yy -- Copy <digit> lines (e.g., 5yy will copy the next 5 lines since the current line)

☆ **p** -- Paste after cursor

P -- Paste before cursor

Search and replace

/ -- Search forwards (it's the forward slash). For example "**/myVariable**" will search for myVariable from current cursor position. To search for the next occurrence, press '**n**'. To search for the previous occurrence, press '**N**'.

? -- Search backwards. Similar usage as '**/**', but just search backwards.

-- While your cursor is over a particular word/variable, you can press **#** to search for the previous appearance of this word.

gd -- While your cursor is over a particular word/variable, you can press **gd** to go to the line where the variable is declared.

:%s/word1/word2/g -- Replace every occurrence of *word1* with *word2* in the whole file.

Other shortcuts

☆☆ **gg=G** -- The all-time magic command: automatically indents all your code! Note that auto indentation only works if your brackets match.

== -- Indents the line the cursor is on.

ctrl+n -- Word completion. Typing part of a word, then press **ctrl+n** to see a list of possible options (`vim` will search through your file to list down all similar words for your choice). `Vim` will also complete the word if there is only one choice. This is very useful if you define long and error-prone variable names (you only need to type a bit and `vim` will auto complete the rest). Press repeatedly to cycle through possible words.

`:set number` -- Show the line numbers in `vim`. Makes it much easier to see where to jump to with the **`<line number>G`** command.

`:set nonumber` – Hide the line numbers in `vim`.

`:vsp` -- Split panes vertically. This command will open your current file in a parallel pane; thus you can view and modify different parts of your program concurrently in two panes. **`ctrl+w ctrl+w`** shifts focus from one pane to pane (yes, press it twice). You can close a pane with the quit command (e.g., **`:q)`** in that pane.

Running Unix commands from inside vim

```
:!gcc -Wall %
:!a.out
```

The above sequence of commands will compile and run your program from within `vim`. Remember to save the file first or the previous saved copy will get compiled.

Useful UNIX Commands

`ls` -- List files and sub-directories in the current directory.
`cd <directory>` -- Enter a directory.
`cp <file1> <file2>` -- Copy file1 to file2.
`rm <file>` -- Remove this file.
`mkdir <directory>` -- Create a directory.
`mv <file1> <file2>` -- Rename file1 as file2.
`<tab>` -- for auto completion of UNIX commands.

Disaster Mistake

Every semester, some students will (by accident) type **gcc -Wall code.c -o code.c** and thus overwrite your source code with unintelligible machine code. You MUST be careful as it is almost impossible to recover your program.

The right command to compile your program to a specified output file is:
gcc -Wall code.c -o code.exe **or**
gcc –Wall code.c

Original author:
Fong Li Heng, Chang Yin Hong, Bachelors of Engineering (Computer), Class of 2010

Modified by:
Zhou Lifeng
zhoulife@comp.nus.edu.sg
19 Aug. 2012