**What is Vim?**

Vim is a lightweight text editor that is accessed via a command line interface. Vim allows you to modify what is written in files, whether it be a text file or a program, you can modify it in Vim; you can also create said files using Vim.

**Why should I learn Vim?**

Here is a scenario where learning any command line interface text editor would be applicable. Let’s say you are updating a file which holds some functions for a program. One of the libraries that your program uses has updated and changed the names of its function and parameter types. This server is only accessible via a command line interface and is also not connected to the internet; your only choice is to modify said file via Vim or any other command line text interface.

Generally, knowing how to use Vim is essential for tasks requiring interfacing with command lines to access your workflow. Vim is also useful for fast navigation and editing with its commands, which this tutorial will go over. It is also a highly powerful text editor that when modified with plugins can act more like an IDE while still being a lightweight text editor.

**Creating and saving files in Vim**

In a terminal, to create a file using vim, type: **vim “filename”** (replace “filename” with the actual filename).

**:w** – save current file, but stay in the vim editor

**:x** – save current file and exit out of vim

**:q!** – do not save current file and exit out of vim

**:saveas file** – save the file you are working on under a different name than what was accessed.

**Navigating through the Vim editor**

You cannot move and select text via your mouse. Vim uses the keyboard by default to move around the editor. This is because when you type, you want to keep your hands on the keyboards and away from the mouse in order to improve productivity.

**You can navigate using the arrow keys**

* Left and right arrows to move one character in selected direction
* Up and down arrows to move one line in selected direction

Else, there are other ways to navigate through Vim (in a more efficient manner perhaps)

**w** – jump forwards to the start of a word (**W** does the same, but for words with punctuation).

**e** – jump forwards to the end of a word (**E** does the same, but for words with punctuation).

**b** – jump backwards to the start of a word (**B** does the same, but for words with punctuation).

**0** – jump to the start of the line (0 is zero)

**$** - jump to the end of the line

**gg** – go to the first line of the document

**G** – go to the last line of the document

**nG** – go to line n (replace n with the line number you wish to go to. This is useful for quickly finding the line with an error when compiling).

**/pattern** – search for a pattern (find a word) in your file

* **n** traverses to the next found pattern
* **N** traverses to the previous found pattern

**Editing a file in Vim**

**Insertion mode**

You have now learned how to navigate through a file and hopefully have seen how quick moving throughout Vim is compared to a graphical text editor. The same can be done for inserting and replacing words throughout Vim.

**i** – insert before the cursor

**I** – insert at the beginning of the line

**a** – insert after the cursor

**A** – insert at the end of the line

**o ­­**– append a new line below the current line

**O** – append a new line above the current line

**ea** – insert at the end of the word (the word the cursor is at)

To exit insertion mode, press the **Esc** key. This will bring you back into the normal mode (navigation).

**Normal mode**

In the normal (not insertion) mode you may also replace characters, words, undo a change, redo a change, and copy/paste lines.

**r** – replace a single character where the cursor is at (after pressing r where the cursor is, you must type in the text you’d like to replace it with).

**u** – undo

**Ctrl** + **r** – redo

**.** – repeat last command

**:%s/old/new/g(c)** – replace all words that match the pattern “old” with the word “new” (if c is added after g, this allows you to confirm each swap).

**nyy** – copy n lines (replace n with the number of lines you’d like to copy)

**yw** – copy the word from the cursor position to the end of said word

**p** – paste what is in your vim clipboard after your cursor

**P** – paste what is in your vim clipboard before your cursor

**ndd** – cut n lines

**dw** – cut the characters of the word from the cursor position to the end of the word

**D** – cut to the end of the line

**x** – cut the selected character

**Workspace**

A feature of vim is to have multiple tabs and multiple windows for each tab, so that you can create a workspace that works best for you.

**:tabe** **filename** – create a new tab with the filename to open

**gt** – move to the next tab

**gT** – move to the previous tab

**ngt** – move to tab number n

**:split** **filename** – create a second window horizontally to show filename (in this tab)

**:vsplit filename** – same as split but a vertical window

To navigate through split frames, to go to the next one, you type **Ctrl** + **w** twice.

**.Vimrc**

With Vim, you have a file that stores vim settings that is loaded into Vim each time you run Vim. This is known as your vimrc. Your vimrc is located at ~/.vimrc, where ~ references your home directory. The vimrc file can load commands immediately when Vim is executed that normally you’d have to set each time Vim is run. For example, if you want to show lines, without setting it in your vimrc you’d have to type **:set nu** each time Vim is run. By putting **set nu** in your vimrc (notice I did not include a colon), this will opt to show lines each time Vim is opened without having to type it in at runtime.

To edit your vimrc, run **vim ~/.vimrc**.

Here are some useful things to include in your vimrc.

**set nu** – shows the corresponding line numbers to the left of each line.

**syntax enable** – enables syntax highlighting. Very useful when editing code.

**colorscheme “selectColorschemeHere”** – allows you to load a specific colorscheme into your editor. You can install colorschemes at ~/.vim/colors. To see the available colorschemes, on a vim editor, type :colorscheme, then hit space, then hit tab. Do not press enter.

**set smartindent** – autoindents your code accordingly to the language the file is written in

**set shiftwidth=n** – (with n as a number) the number of spaces to shift per indent.

**set tabstop=n** – the number of spaces to shift per TAB indent.

NOTE: the shiftwidth applies to a shift either done by **>>** (shift this text to the right), **<<** (shift to left), or smartindent. Tabstop applies when you hit the TAB button.

NOTE: if you want to use these commands in the vim editor as a one-time option (in this editor instance only), you add a colon (:). This is how vim knows you are writing a command.