Michael Flanagan

CIS 140U Lab 3

Chemeketa CC Online

Summer 2017

**1.)**

**A) Movement**

Typing the escape key takes you out of text edit in normal mode, while in normal mode you can move around by using either the arrow keys or k for up, j for down, h for left, and l for right.

This is a necessity for navigating around the document. Quickly if you have not enabled mouse support.

**B) Insertion**

I - puts you into edit mode.

This is needed to type in the editor.

**C) Quitting the editor**

:q! <enter> exits the editor without saving the changes.

**D) Opening the editor**

To get into the editor tutor type vimtutor <enter>.

More likely, in a real world scenario we would access vim by typing vim <filename>. This command will open an empty file in the VIM editor and the save will then be saved under that filename.

This command is needed to access the editor.

**F) Deletion**

Press x to delete the character at the cursor.

This needed to correct mistakes.

**G) Appending**

Typing a lets you append text onto the end of the lines

This is an easier way to add text onto the end of a document. In addition, later on we will append with copy and paste. It is convenient to append sometimes.

2.) Deletion commands

In this document, I may refer to a word as a string. If you are familiar with computer programming this term will not be new to you. If you are not familiar than you can just think of a string as a word. A string also can refer to any string of characters so it may be letters, numbers, or symbols, put together to tell the computer system that these particular characters in this order have a meaning.

Furth more if you see me use of the hash sign # that just means a number.

These are ways to be more efficient when removing items from the text file

**A) At the cursor deletion**

Typing dw will delete the word at the cursor location.

**B) Cursor to end of line deletion**

d$ deletes everything from the cursor to the end of the line.

**C) Word deletion**

dw deletes everything up to the next word on that line and places the cursor on the first letter of the following word.

De deletes the word placing the cursor before the first letter of the next word.

**E) Word jump**

2w - jumps two words up and places the cursor at the first char of the word the cursor jumped to.

5w - jumps 5 words and places the cursor at the first char of the word the cursor jumped to.

Any number placed in front of the w causes the cursor to jump that many words.

2e acts the same but it places the cursor at the end of the word the cursor jumped to.

The numbers are operators, and tell the jump to command how to behave.

0 will jump back to the start of the line.

With the knowledge we have obtained already we can now combine these 2 commands to do actions on multiple words here are some examples.

de - means delete from the cursor to end of word, and we have 2e which says jump 2 words and land at the end of that 2 word.

d2e - delete to end of word but jump 2 while doing it so it will delete 2 words and land at the end of the second word.

D2w - delete 2 words and land at the beginning of the word following.

dd - delete the whole line

The makers of the system found that you delete whole lines so often that it was easier to just give that its own command.

**F) Undo**

u - will undo the previous command, capital U will return the line to its original state.

U – will undo all previous changes (be careful with this one).

Remember this it could save you sometime it can be quicker to just undo something than to delete. It also sets you up for redo which you might decide you didn’t want to undo something.

Ctrl r will redo commands that were undone.

As explained previously might be handy if you undo something than change your mind you can redo.

**3) Put**

**A) Put Back**

p will put back text that has just been deleted and places it after the cursor.

This is kind of like redo for accidentally deleted text.

**B) Replace**

To replace a character type r than type in the new character you want to be placed in that spot.

This can be nice you used the wrong character. Some functions seem like duplicates of others but as you you vim and get comfortable you might find one seemingly duplicate action is slightly different.

**C) Change**

ce will clear the text til the end of the line and you may than type whatever you for as many characters that word contained.

Good for quickly changing a word that you know you will need to rewrite multiple characters. While maintaining the part that you know is good.

**D) Change to endline**

If you would like to clear everything from the cursor to the end of the line.

You can use c$.

You can also use backspace while in this mode to correct any mistakes, so correct them than hit escape to jump out of the “c” mode.

This can be more efficient than moving around the file and doing modifications than scrolling around.

**4.) Location**

**A) Show Location**

ctrl-G will show your location in the file and the status. It will be displayed at the bottom of the screen.

This is good to tell you line number specifics so when jumping around you know what line numbers to jump to.

**B) jump to location**

G jumps to the end of the file. Number G jumps to that line in the file. gg jumps to the first line.

Actually quicker moving around the document than using the movement keys

**C) Search**

You can search for specific phrases going down in the file by /. You can search for specific phrases going up in the file by ?.

You can search for the next instance of the phrase by n. It the search will be done upwards or downwards dependent on if your are in ? mode or / mode accordingly.

You can also search in opposite mode of the mode your in by using lowercase N.

This is kind of self explanatory its quicker to search for a for quite often than try to look through the whole document with human eyes for that word.

**D) Matches**

% while the cursor is on a (,),[,],{, or } goes to its match.

Nice for removing unnecessary brackets and punctuation

**E) Substitutions**

Substitutions can be done with :s command the format is “:s/old string/new string”.

You can also add a g flag to replace all instances of the old string with the new string. Example: to replace or substitute a word you can use. “:s/old string/new string/g” .

(be sure to leave out the quotations when executing the command)

If you want to substitute strings between 2 line numbers use #’s. Example “#,#s/oldstring/newstring/g.”

If you want to substitute all occurances in the file use “%s/oldstring/newstring/g”

If you would like to get a confirmation prompt each time add c. Example “:%s/old/new/gc”kk

**5) External Commands**

**A) The Familiar command**

:

**B) The exclamation point**

**!** These 2 commands need to be combined to execute any external command

**C) Executing an external file**

Some examples are :!w FILENAME which will save the current text to a file under the name FILENAME. (replace FILENAME with whatever you want the file to actually be named).

This could be useful when programming if your viewing a text file filled with data and you want access your IDE while your in text file to make a quick code modification while staying in VIM

**D) Deleting a an external file**

You can also delete the file you just created by !del FILENAME

To save a subsection of the file type v than use the the movement keys to highlight the portion you want to capture than type :w FILENAME.

**E) Retrieving an external file**

To retrieve a file and put it below the cursor in the current file type :r FILENAME.

**D) Retrieving an external Directory**

You can even use this to retrieve the current directory that the file is by :r !dir

**6.) Open**

**A) Open a new line below the current**

Type o insert a new line below the cursor in insert mode than you hit escape to add the text on that newline.

For efficiency rather than moving around inserting and hitting return.

**B) Open a new line above the current**

Type O to start newline above the cursor in insert mode.

For efficiency also just when a newline is needed above the current line

**C) Append text onto the end of the word that the cursor is on.**

Type a to append text onto the end of the word that cursor is on. Type A to append text to end of the line.

This is very handy when merging multiple text files, or copying text around in the file

**D) Copy**

The y operator copies text. The VIMTutor says that y stands for yank. That’s a helpful way to remember the command. When you press the y key it will Vim will than be in copy mode and as you move over text it will highlight. You can quickly highlight to the end of the line by j$.

Kind of self explanatory. You have one piece of text you want to copy than use the y operator.

**E) Paste**

p pastes text. Capital R puts you into replace mode as usual escape takes you out of replace mode.

Kind of self explanatory. You have one piece of text you want to paste than use the p operator.

**F) Replace**

Capital R puts you into replace mode as usual escape takes you out of replace mode.

Useful if you want overwrite text with different text.

**F) Copying whole words**

You can also copy whole words by yw instead of just y.

Quicker than selecting characters and when you don’t need the whole line this is a good one to use.

**E) Set options**

You can set additional options by using :set <option>. These all have long hand and short hand versions. The options available are

1. ‘ic’ or ‘ignoreCase’ which will ignore upper/lower case when searching.
2. ‘is’ or ‘incsearch’ which is show partial matches for a search phrase.

**3)** ‘hls’ or ‘hlsearch’ which will highlight all matching phrases.

Hint: for most of the commands its best to try to think of an anagram that makes sense in your mind. You could think ignore capitals for ic and ignoreCase. You could think of is and incsearch as incremental search. You can think of hls and hlsearch as highlightsearch.

**F) Toggling set options off**

You can also switch the previously set option off by prepending a no. Example :set nois.

If you want an option turned off.

**7.) The Help Menu**

**A) Accessing the help menu**

There are multiple ways to access the help menu.

1) press the help key (This is likely not on a standard keyboard)

2) press the f1 key

3) type :help <enter>

A menu will pull up with guidance on how to navigate the menu. It pretty much works the same as VIM, and is also similar to the standard unix filesystem to access different options and menus type its relevant command given by the top level help menu.

This is likely my favorite part of the whole tutorial. The help menu is extremely useful. If I can’t remember anything else I make sure to remember how to access help, than I can look up whatever you need.

**B) Switching between Vim and the help menu**

Typing CTRL-W lets you toggle between the help window and the standard vim window.

Once you open help it likes to keep you stuck in one window or the other, you can get jump between the windows with CTRL-W.

:q <enter> will quit the help menu.

If you no longer need to look at the help menu and its annoying you on the screen use this command.

**C) Getting help on specifics**

If you type :help command you can get help on basically any subject. All commands need to be followed by the <enter> key of course to execute them.

You can type part of a word and it will do its best to match it to a word in the help database.

For example typing ‘:help w’ will bring up the help menu for “word” and so fourth.

This makes the help menu much easier and quicker to use. You don’t even have to know where in the help menu the thing you need help with is located. You can just type what you need help with and it will than find that in the help menu for you.

**D) Creating custom Vim configurations**

If you would like to create custom settings that are the same every time you start you can create a vimrc file. to start editing the file type :e ~/.vimrc

After a while you will likely figure out that you have certain settings that you use all the time and it’s a pain to keep toggling them on or off each time you start vim. Vimrc is is a tool you can use to create permanent configurations.

To see an example vimrc file for guidance type :r $VIMRUNTIME/vimrc\_example.vim

This will likely come in handy at least to set vimrc the first time. You likely won’t modify it that often once you find a configuration you like. However if you decide to change something after a while of not messing with the vimrc settings you can pull up the example again for reference.

To write the file use :w

For help documentation on custom vimrc files type :help vimrc-intro

**E) Completion**

Completion is the ability of Linux to autocomplete your word based on the keys already typed.

This is standard action in systems, such as Windows cmd line, Linux cmd line, most IDE’s when in doubt on word completion hit tab to see if the computer can find what you are trying to type. This can also speed up time at times if you know how to use it correctly because you do not have to type out full words.

To edit a file you can type :edit filename

Completion would work by :edit fi <tab> than you will see it will complete and say edit filename.