Vim Guide

1 What is Vim

Vim, or Vi IMproved, is a powerful yet lightweight command-line text editor that supports many features to allow users to improve their editing speed in many ways. It is in fact so powerful and useful that all modern **I**ntegrated **D**evelopment **E**nvironments or **IDE**s (like VSCode and Sublime Text) have a Vim emulation feature.

2 Basics of Vim

Important: Throughout this document, we will be using angle brackets to denote a Vim command (e.g. <`yy`>). When using Vim, you do not need to type out the angle brackets — only the actual command (in the above case, you would only type out *yy*).

In order to open up a file using Vim, simply enter the following onto the command line:

```
% vim path/to/file.txt
```

Vim has two main modes through which you can interact with the text:

- **Normal Mode** — In normal mode, you can interact with the text and run powerful commands to edit your file in bulk, or move around more freely. As its name suggests, it is meant for commands and not for typing actual text!
- **Insert Mode** — In insert mode, you can type text freely and it will be inserted into your file at the location of your cursor

Note that **Normal Mode** is the default mode. To switch to insert mode from normal mode, simply press <`i`>. To switch back to normal mode from insert mode, press <`ESC`>.

Once you are done with the file you are trying to edit, you can exit in one of three ways. These must be executed while in normal mode.

- <`:wq`> — This will save your file and exit Vim. **Note**: <`:w`> only save your file without quitting
- <`:q`> — This will exit Vim **without** saving your file. If you have unsaved changes, this will display an error, which you can bypass with the last command.
- <`:q!`> — This will exit Vim, ***discarding any changes*** **WARNING:** Any unsaved changes will be lost permanently!

3 Anatomy of a Vim command

Before we go into a list of useful Vim commands, we explore the structure of an arbitrary Vim command, and see how to use Vim to make the command do different things. A Vim command is composed of three parts:

- How many times to repeat this command. This can be a number, or just not included (which implies run it once).
- The command name — this is generally one letter (such as <`d`> for "**d**elete").
- What piece of the file the command should act on — this could mean a specific line number or something relative to the current location of the cursor.

For example, if we take the command <`3dtc`>; the `3` means run <`dtc`> 3 times; the `d` means "Delete"; the `tc` means "until you see the first instance of the letter c". So, <`3dtc`> means "Delete until you see the 1st instance of c 3 times"; or put more simply "Delete until you see the 3rd instance of c". Thus, if we were to run this on the text "I am ∧crossing the crosswalk at Forbes and cyert" (the cursor is represented by the ∧), we would end up with "I am ∧cyert".

Note: This search is case-sensitive, so if you were to run <`3dtc`> on "I am ∧crossing the crosswalk at Forbes and Cyert", nothing would happen, as Vim cannot find 3 instances of "c" to delete to.

4 Useful Pieces of the file

Note: For this section, we will continue to use the delete command as an example to illustrate the various pieces of a file that a command can act on.

- **$** — "Until the end of the current line". Executing <`d$`> would delete all the text up to the end of the current line.
- **0** — "Until the beginning of the current line". Executing <`d0`> would delete backwards up to the beginning of the current line.
- **w** — "Until the end of the current word". A "word" is considered to end once punctuation or a space is encountered. Thus, executing <`dw`> on "fo∧o_bar: banana" would leave you with "fo∧: banana".
- **iw** — "The whole word I am in". As before, a "word" is considered to begin/end once punctuation or a space is encountered. Thus, executing <`diw`> on "foo b∧ar baz" would leave you with "foo ∧baz".
- **i)** *or* **i(** — "Within the nearest set of enclosing parentheses". Thus, executing <`di)`> on "(a fool is (a pe∧rson) who)" would leave you with "(a fool is (∧) who)".
- **i[**, **i{** *or* **i<** — Same as above but with square brackets, curly brackets and angle brackets respectively.
- **t*char*** — "Until the first occurence of *char*". Thus, executing <`dts`> on "sudge∧icles" would leave you with "sudge∧s".
- **T*char*** — "Backwards until the first occrence of *char*". Thus, executing <`dTs`> on "sudge∧icles" would leave you with "s∧icles".
- **The same character as the command** — "The entire line". Thus, executing <`dd`> would delete the entire line.

5 Useful Vim Commands

d — *delete* — This command deletes a section of text as specified by the piece of the file (see the previous section). **Note:** This command will also move the deleted text into your clipboard automatically. As such it more resembles a "cut" operation than a "delete" operation.

y — *yank* — This command copies (yanks) a section of text as specified by the piece of the file and puts it in your clipboard.

p — *paste* — This command pastes the last thing to be placed in your clipboard into the text. Note that this command does not require any file location modifiers.

J — *join lines* — This command appends the line following the current line to the current line. Note that this command does not require any file location modifiers.

= — *autoindent* — This command indents a section of text as best Vim knows. **Note:** as its not really possible to indent anything less than a full line; the only way to actually use this command is by executing <`==`> (e.g. autoindent the entire line).

g — *go to* — This command goes to the section of text specified. The only standard sections of text that work with this command are the beginning of the line (<`g0`>), the end of the line (<`g$`>) and the whole line (<`gg`>). When using <`gg`>, you are asking to go to a certain line (so <`gg`> takes you to the first line, <`300gg`> takes you to line 300). One can also go to the last line by using <`G`>. **Note:** It is also possible to go to the next or previous word with <`gtw`> and <`gTw`> respectively.

u — *undo* — Undo the previous command or insertion.

CTRL-r — *redo* — Redo the previously undone command or insertion.

. — *Do again* — Repeat the previous command or insertion.

/{**TEXT**} — *Find* — Searches forward for the first occurance of {TEXT}. To go to the next instance, one can use <`n`>. To search backward and go to the previous instance one can use <`N`>.

:%s/OLD/NEW/g — *Find and replace* — Finds all occurrences of {OLD} and replaces them with {NEW}. If you only want to replace some instances, then you can add a **c** at the end of the command (so **:%s/OLD/NEW/gc**) and it will ask for confirmation before each occurrence. The Vim Find/Replace tool is extremely powerful and the above is just one of many ways to use it. See https://vim.fandom.com/wiki/Search_and_replace for more on this tool.

6 Useful Aliases

An alias is a command that is effectively two or more commands rolled into one. What follows is a list of useful aliases:

- **c** — *change* — This command deletes a section of text as specified by the piece of the file; and then immediately puts you in insert mode. <`diw`><`i`> will accomplish the exact thing as <`ciw`>.
- **A** — *Append text* — This command moves you to the end of the current line and immediately puts you in insert mode. As such, it is equivalent to <`g$`><`i`>.
- **o** — *open new line* — This command starts a new line immediately after the current one and puts you in insert mode at the start of the new line. It is equivalent to <`g$`><`i`><`ENTER`>.

7 Customizing VIM

Vim draws all of its settings from a global settings file called the *vimrc*. This file can be found at **~/.vimrc** and can be edited just like any other file. During the setup lab, we added some useful settings to your vimrc, so it should contain some things. If it does not, then please ask a TA for help!

Note: The settings file contains a list of Vim commands that is executed as any new file is opened. As such, any of these commands can be executed at any time by prefixing the command with ":". For example, in the default vimrc you may notice the line **set nu**, which causes line numbers to be displayed. If you wanted to enable this temporarily without putting it in the vimrc, you would simply execute <`:set nu`> in Vim.

Note: If you want to temporarily disable a setting, you can take the name of the setting, prepend it with **no** and then set that "setting". As an example, for line numbers, the name of the setting to enable line numbers is **nu**, so to disable it you would run <`:set nonu`>.

Some useful Vim settings are

- **set paste** — Turns off autocompletion and autoindent to allow for a nicer pasting experience
- **set nu** — Turns on absolute line numbering
- **set rnu** — Turns on relative line numbering
- **set mouse=a** — Turns on mouse support
- **set background=dark** — Changes the color scheme to better match a dark background
- **set background=light** — Changes the color scheme to better match a light background
- **noremap <C-a> g^** — Causes CTRL-a to move you to the beginning of the line (Note: CTRL-a has this behavior in many other applications including Chrome and Terminal)
- **noremap <C-a> g_** — Causes CTRL-e to move you to the end of the line (Note: (Note: CTRL-e has this behavior in many other applications including Chrome and Terminal)
- **noremap <C-k> d$** — Causes CTRL-k to delete the rest of the line (Note: CTRL-k has this behavior in many other applications including Chrome and Terminal)

8 "Found a swap file by the name ..."

Throughout your Vim career you may encounter the following message when opening a file:

```
Found a swap file by the name "PATH/TO/FILE.swp"
          owned by: astanesc   dated: Mon Dec 15 14:10:40 2018
         file name: PATH/TO/FILE
          modified: YES
         user name: astanesc   host name: unix4.andrew.cmu.edu
        process ID: 3348
While opening file "PATH/TO/FILE"
             dated: DATE

(1) Another program may be editing the same file.  If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes.  Quit, or continue with caution.
(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r PATH/TO/FILE"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file "PATH/TO/FILE.swp"
    to avoid this message.

Swap file "PATH/TO/FILE.swp" already exists!
[O]pen Read-Only, (E)dit anyway, (R)ecover, (D)elete it, (Q)uit, (A)bort:
```

This can appear if you were editing a file and then your instance of Vim unexpectedly closed — usually because you closed your terminal window or disconnected from the internet. The reason this message appears is that vim saves your unsaved work in a "swap" file which gets deleted when you close Vim normally or save (hence the beginning of the error message). Thus, if your instance of Vim unexpectedly closed, Vim would not have had a chance to remove the swap file, leading to this message.

What to do about this: Since the swap file **may** contain unsaved work, it is generally recommended you first try to recover any unsaved work. To do this, hit **R** when you see this screen. Once you are sure that there is no unsaved work, you need to delete the swap file so that the above message does not appear every single time you open up the file. To delete the swap file you can either hit **D** at the above screen or just delete it as with any regular file (`% rm path/to/file.swp`).

9 Vim Quick Reference Guide

Note: This section lists the most commonly used commands and what they do. This is not meant as a comprehensive guide and it assumes that you have read the previous sections too.

- <dd> — Delete the current line
- <di)> — Delete within the nearest set of enclosing parentheses
- <di}> — Delete within the nearest set of enclosing curly braces

- <`yy`> — copy the current line
- <`==`> — Autoindent the current line
- <`g0`> — Go to the beginning of the current line
- <`g$`> — Go to the end of the current line
- <**`NUM`**`gg`> — Go to line number **NUM**
- <`u`> — Undo
- <`CTRL-r`> — Redo
- /{**TEXT**} — Find {TEXT}