98-172 : Great Practical Ideas for Computer Scientists

Time To Make A Choice (I Choose You, `vim`!)

Getting Started

Welcome to Lab-itation 3! You seem to have chosen `vim`! When you logged in, you should have gotten a message to update your `.bashrc` file. Type `gpi_update` on the command line to update it.

Part of the update was to add a new `gpi_install` function which will let you install important things that we give you.

If you run `gpi_install`, it will give you some potential things to install. You should install `vim_config` before continuing. Go ahead and open up `vim`.

Important Overall `vim` Things

(a) Type the *i* key to switch into `insertion mode`, where you can actually type text.

(b) Type the *escape* key to switch into `command mode`, where you will type all your commands.

Getting Your Source File

We have provided example source files for you to use for this lab-itation. The catch is that they are provided on our website, so you will have to copy them into `vim`.

If you want to use `python`, you can find the file at
`http://www.andrew.cmu.edu/course/98-172/examples/example.py`

If you want to use `c0`, you can find the file at
`http://www.andrew.cmu.edu/course/98-172/examples/example.c0`

In either case, go into insert mode in `vim` by pressing the *i* key, and copy the contents of the file into `vim` by using Ctrl-C/Ctrl-V like normal. (If you are in `putty`, then right click to paste.)

Uh-oh, the indentation got all messed up! That's because we weren't in *paste mode*.

Before we switch into *paste mode*, undo the pasting of the code by switching to command mode (type the *escape* key), and pressing *u*).

If you want to redo something, you should use Ctrl-R. Undo and redo keep track of a large number of mistakes; so, you can use both of them several times in a row.

Next, to get into *paste mode*, type *escape* to get into command mode, `:set paste`, to tell `vim` the command, and enter to make it happen.

Then, switch back into insertion mode (type *i*) and copy the contents of the file into `vim` again. Yay! The indentation is correct this time! At this point, you should switch out of *paste mode* by (once again!) going back into command mode and typing `:set nopaste`

Now that you have the code, the sixth line should be a space for your andrewid. Move to that line by typing `:<line number>` in command mode. So, here, you would type `:6`

Finally, type your andrewid in the right place!

Get Ready To Code

In this section, you'll finish up the header of the file. It teaches **searching**, **selecting**, and **deleting a line**.

(a) Search for `delete-this-text-and-replace-it-with-your-name` in the file by typing `/<the string to search for>` when in command mode. So, here, you would type `/delete-this-text-and-replace-it-with-your-name`

(b) Switch into *visual mode* by going into command mode and typing *v* to get to *visual mode*. Then, select the text you want to delete ("delete-this-text-and-replace-it-with-your-name") by moving to the right. Once it is selected, you should type `d` to delete it. Now, write your actual name in place of the thing you deleted.

(c) The next line is too personal for you to enter; so, you should go to the line you want to delete and type `dd` to delete it.

Fix Some Stuff

In this section, we will cover **search** and **replace**.

(a) To search for the text `this_function_name_is_terrible` by going into command mode and typing `/this_function_name_is_terrible`

(b) To replace the text `X` with the text `Y` in the entire file, you should type `:%s/X/Y/` So, go ahead and replace the function name with `add` by doing `:%s/this_function_name_is_terrible/add/` If you want `vim` to replace all the instances of `X` on each line with `Y` (as opposed to just the first one), you should use the command `:%s/X/Y/g`

(c) Unfortunately, the person who was writing the file "accidentally" mispelled terrible as turrible. To fix this, let's first highlight the section at the end of the file with the print statements by going into *visual mode* and moving around until the right lines are highlighted. Then, type `:` (notice that `vim` automatically adds `'<,'>` to the command). Since we want to replace, finish the command until it reads `:'<,'>s/X/Y/` (here `X` is `this_function_name_is_turrible` and `Y` is `add`). You should note that the only difference between this command and the last one is `'<,'>` instead of `%`

Indentation

The default behaviour when you press the tab key is for `vim` to insert an ASCII tab character. By default, a tab is displayed as 8 spaces. (Tabs in the middle of lines are a bit more subtle.) You can change the way a tabs is displayed using the command `:set tabstop=<number>`. For example, `:set tabstop=4` would show tabs as 4 spaces.

The `tabstop` setting will changes the way the tab key is handled, but it does not change `vim`'s smart indentation. We suggest that whenever you change the `tabstop` setting, you also change the `shiftwidth` setting to the same value: if you run `:set tabstop=4`, also run `:set shiftwidth=4`.

Some styles use spaces instead of tabs for indentation. The example file you are working with uses 4-space indentation. To have `vim` use spaces instead of tab characters, use the command `:set expandtab`.

Now whenever you press tab, or when `vim` automatically indents something, it will use spaces. This won't change any tab characters that are already in the file; if you want to do that, use `:retab`.

All of these settings will only last for your current session. To make them permanent, add the commands to your `~/.vimrc`.

Fixing Indentation

To fix indentation of a group of lines manually, you should highlight them using visual mode and type the command > or < depending on which direction you want to go in. You might need this in a moment if you are using `python`.

For many languages, you can have `vim` automatically figure out the correct indentation. To do this, hilight lines and type the command =.

Line Numbers

Before, we mentioned that to go to line number `X`, you should use the command `:X`

At some point, you may want to toggle the line numbers on the left of the screen. To turn them on, use `:set number`, and to turn them off, use `:set nonumber`

Save Your Code

Save your code **and quit** by typing the command `:wq` (If, instead, you just wanted to save, you would do `:w` or to just quit, you would type `:q`)

`vim` doesn't particularly like when you want to quit without saving; so, to do that, you'll have to type `:q!`

If you are using `c0`, compile your code on the command line using `cc0 example.c0` for `c0`. There will be an error that looks like `examples/example.c0:23.49-23.49:error:Parameter missing type`.

If you are using `python`, run your code on the command line using `python example.py` . There will be an error about indentation. (You should move the lines all the way to the left.)

In either case, go ahead and fix the error.

Run the code again, but this time, run it inside `vim` by using "!" So, for `python`, type `:!python %` (the % is for the current file, you could replace it with `example.py`), and for `c0`, type `:!cc0 %`

Make it Prettier

If you are using `c0`, your program still won't compile, because the `do_nothing` function is being used before it is created. In general, this is bad style; so, even if you are using python, you should fix this.

Select all of the lines of the `do_nothing` function using *visual mode* again and type *d* to delete **and copy** them. If you wanted to just copy them, you would use *y* for "yank". Then, to paste them, move to the line above `loop_forever`, and type *p* for "paste."

Finish It Up

`do_nothing` and `loop_forever` are fairly useless functions; so, go ahead and delete all references to both of them. Then re-compile/re-run. You should get sane output.

Tabs and Splits

The command `:tab split` will create a new tab editing the same file you currently have open. `:tab new` will create a new tab with an empty file. To switch tabs, use `gt`. This will move forwards; to switch backwards, use `gT`.

To open a new window editing the same file you current have open, use `:split` or `ctrl-W s`. `:new` or `ctrl-W n` will create a new window editing with a new file. This will create two wide windows, one above the other. To create two tall windows side-by-side, use `:vsplit` or `ctrl-W v`. To switch between multiple windows, use `ctrl-W ctrl-W`.

Color Schemes

You may find yourself wishing that the colors used in syntax hilighting were different. To change color schemes, use the command `:colorscheme <colorscheme>`. You can find valid values for `<colorscheme>` by hitting the tab key–`vim` does tab completion(!), though it is slightly different than bash's. To make the new colorscheme persist, add the command to your `~/.vimrc`.