CSCI 1300 CS1:Starting Computing
Recitation 1
Instructor: Fleming/Gupta
Due: at the end of your recitation section

Objectives:

1. Setup your Moodle account and test that it works.
2. Setup the Cloud 9 (c9.io) IDE and test that it works.
3. Introduction to Linux
4. Zipping files

Moodle Accounts:
All students in CSCI 1300 this semester will be accessing course materials through the Computer Science Moodle: http://moodle.cs.colorado.edu. To use Moodle, you log in with your identikey and password, and then enroll in your class.

Once you've logged in, click on the link to your class:

CSCI 1300 - Fleming/Gupta - CS1 Starting Computing

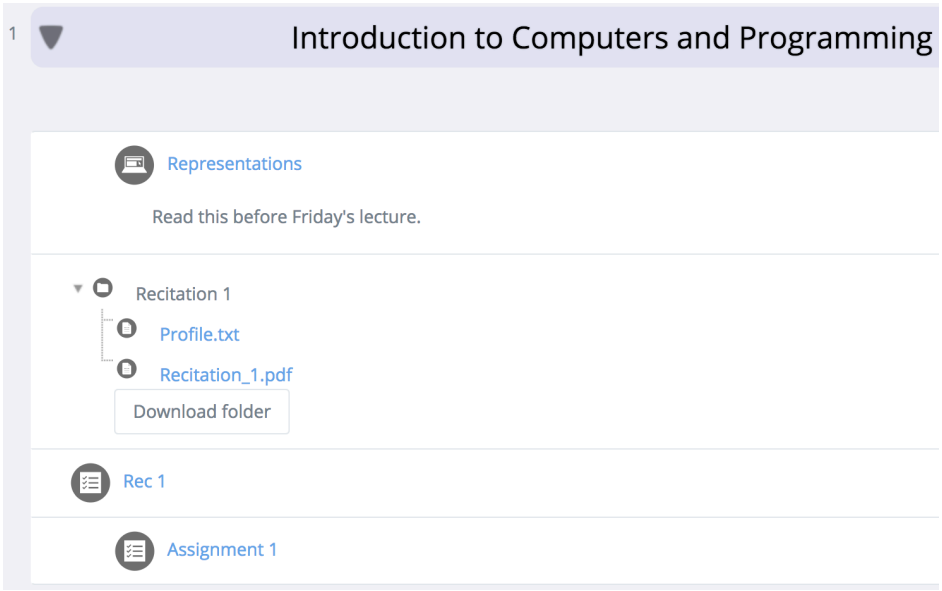
The enrollment key for CSCI 1300 is:

1300fg

Once you're enrolled in the class, you should see the course materials that have been uploaded so far, organized by topic/textbook_chapter/week.

Submit to Moodle:

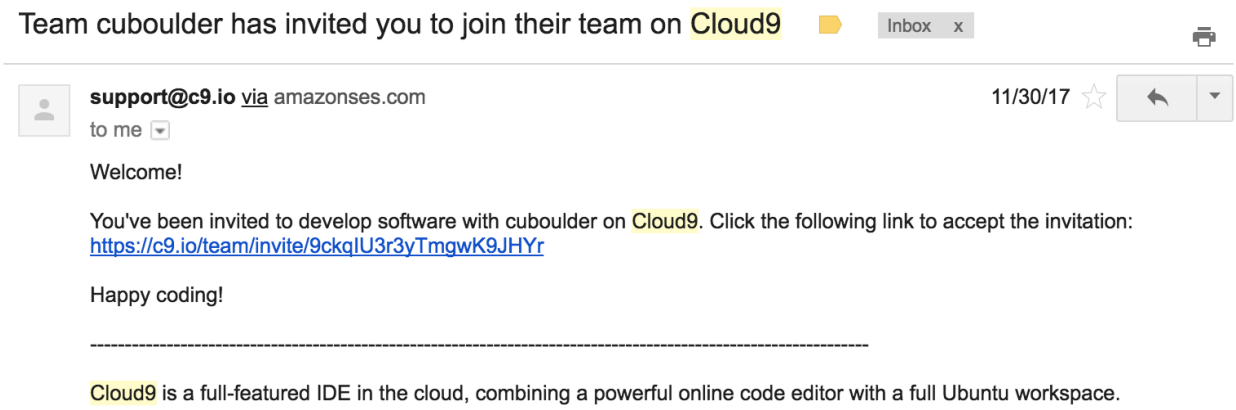
To test that your Moodle account is working, download the *Profile.txt* file found on the first Moodle topic ("Introduction to Computers and Programming" – Topic 1), in the folder **Recitation 1**. Open the file using a text editor like Microsoft Office - Word, OpenWord, Textedit, etc. Insert your answers and save the modified file as *Profile_<CU identikey>.txt* Submit the file to Moodle using the **Rec 1** Submit link.



Getting Started with Cloud 9 IDE:

In recitation today, you will be getting started with Cloud9, a cloud based development environment, which is designed to provide a consistent development environment for all students. Follow the steps below. If you need help ask your TA or one of the CAs.

1. You should have received an email invitation from your TA to join the *cuboulder* team on Cloud 9. Follow the link in the invitation and make an account with Cloud 9. Make sure to use your colorado.edu email address.



Team cuboulder has invited you to join their team on Cloud9 Inbox x

support@c9.io via amazonses.com 11/30/17

to me

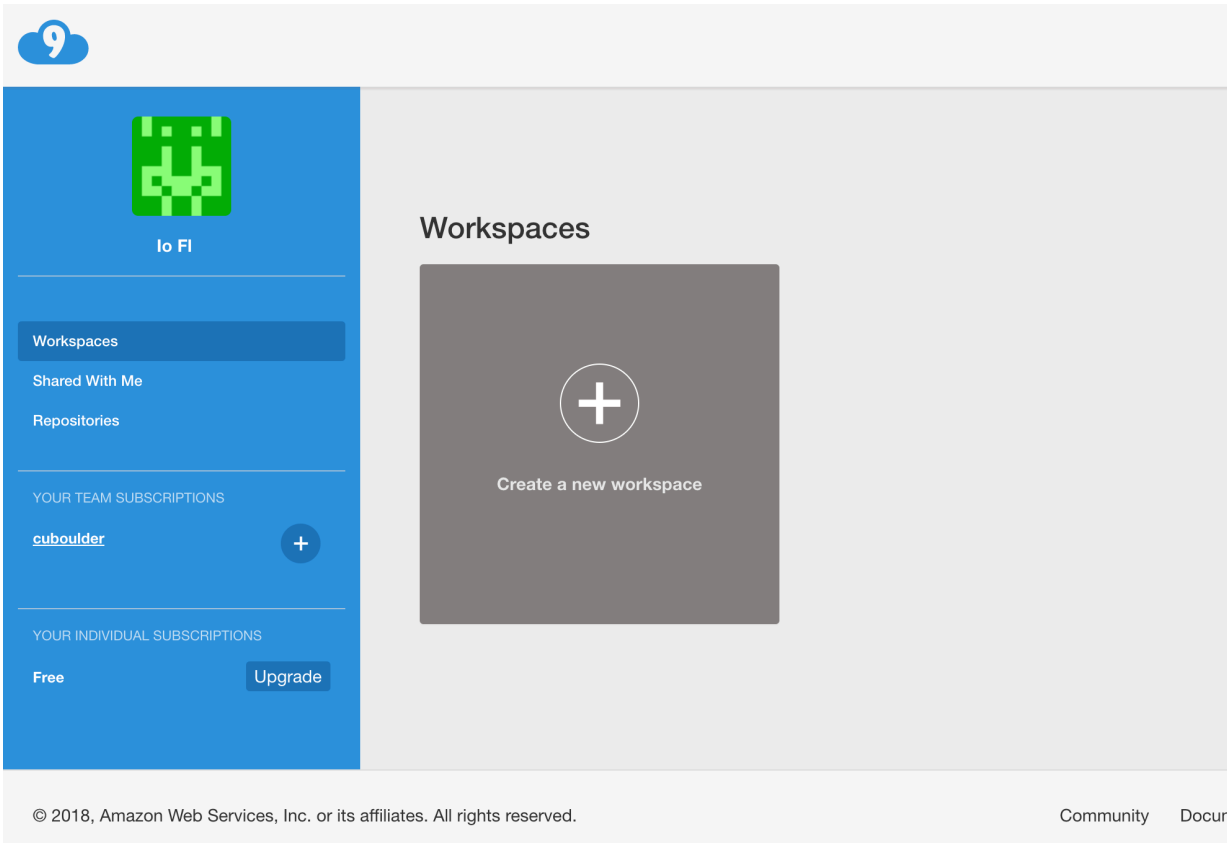
Welcome!

You've been invited to develop software with cuboulder on Cloud9. Click the following link to accept the invitation:
https://c9.io/team/invite/9ckqlU3r3yTmgwK9JHYr

Happy coding!

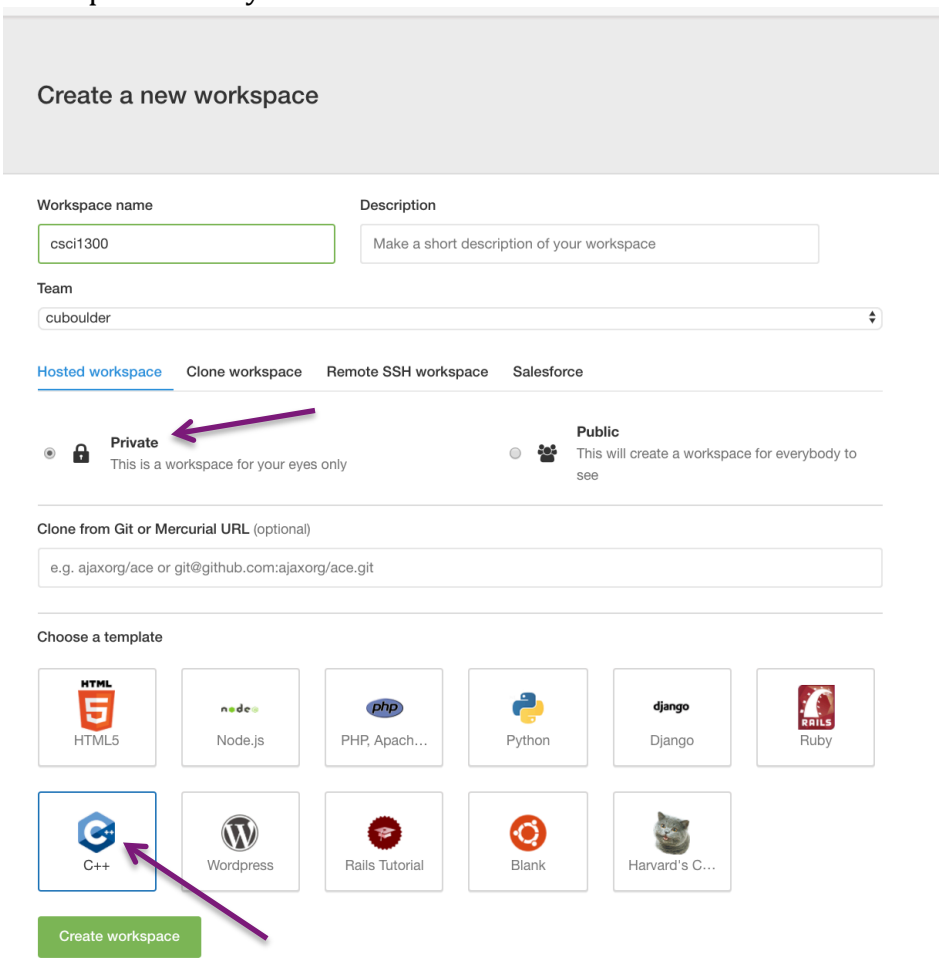
Cloud9 is a full-featured IDE in the cloud, combining a powerful online code editor with a full Ubuntu workspace.

2. Once you have an account and a password, click on "Create a new workspace":



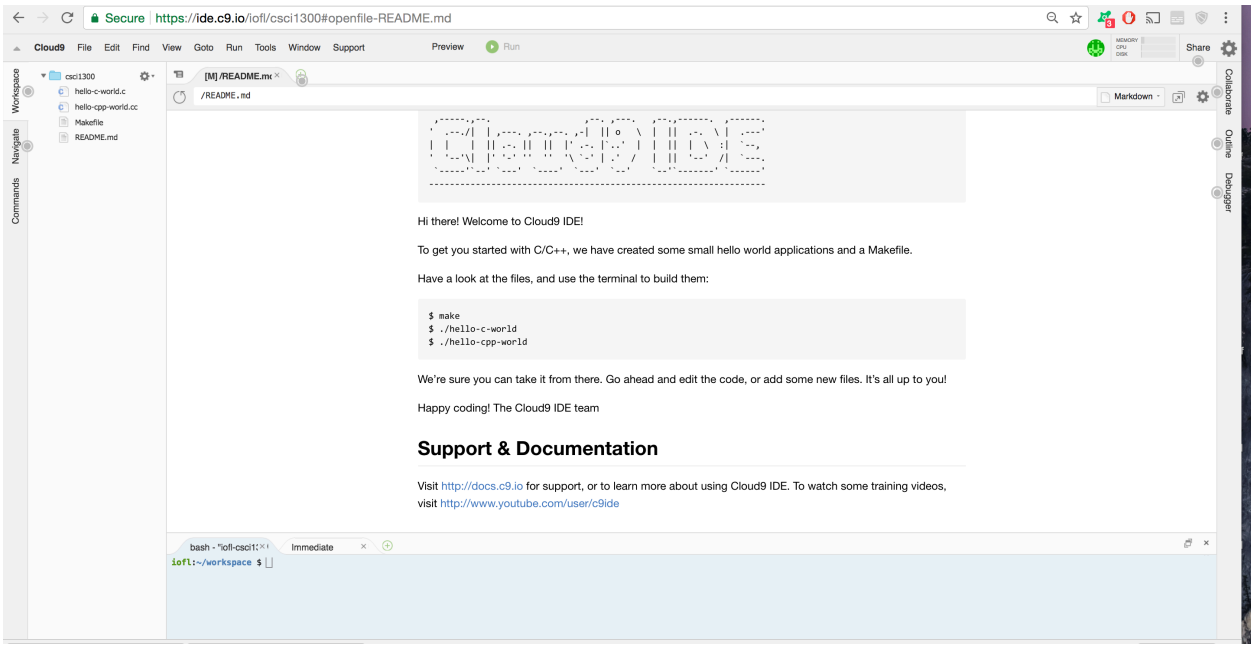
3. The name of your workspace must include the characters “csci1300”. You can add your name, or initials, or S18, or keep it at that. Choose the C++ template!

Very important: make sure your workspace is **Private**. In Cloud9, your subscription allows you one Private workspace, and unlimited Public workspaces. As you can see below, all the members of the cuboulder team can see Public workspaces. To prevent violations of the collaboration policy, we will delete any Public workspaces. You must have a Private workspace for all your work in CSCi 1300 this semester.



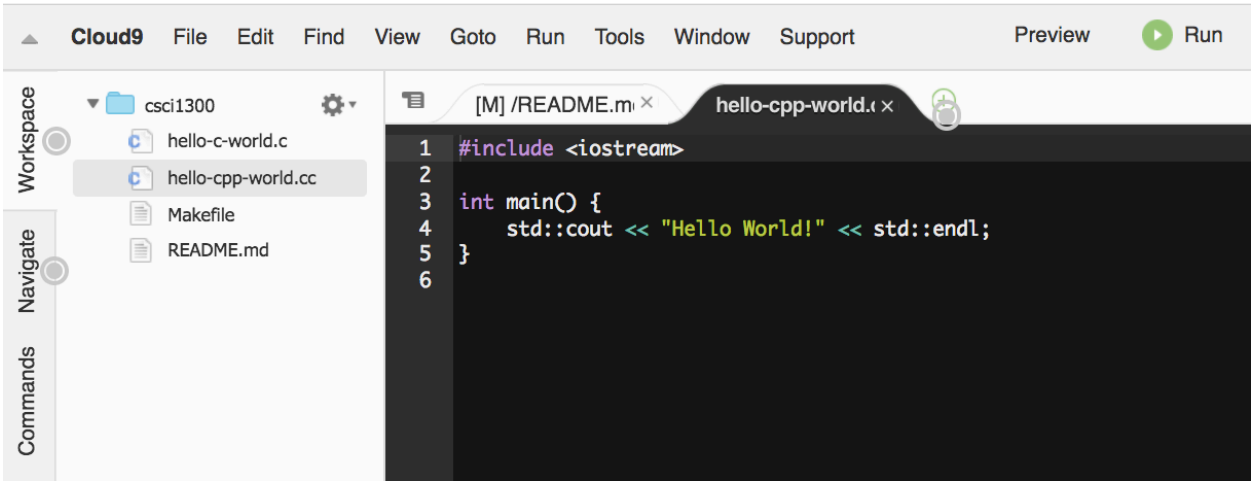
4. After you finish the process of creating a workspace you will need to wait a little bit while your container is being created.

5. Once the container is created, this is how your develop environment will look like:



Familiarize your self with the environment. Click on every gray "target" to learn about its different features. The name of your workspace will be the same as the name of your main folder in the container. You can see it in the Workspace tab, on the left side.

The container opens with two C/C++ files. Double click on the "hello-cpp-world.cc" file. You will see it open in the editor, just like below:



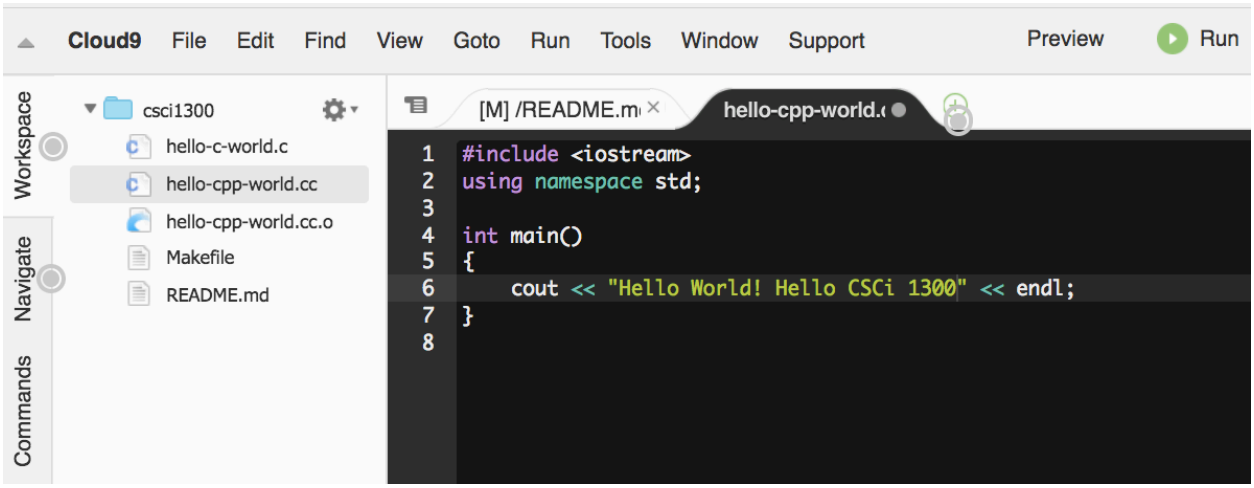
6. Let's try to modify the file. Change the text in the file using the editor window. Make modifications to look like below.

We are adding a statement about using the *standard namespace*:

```
using namespace std;
```

And we are changing the printout statement to print “Hello World!”, followed by “Hello CSCi 1300”. Note that because we added the *standard namespace*, we do not need the “std::” anymore:

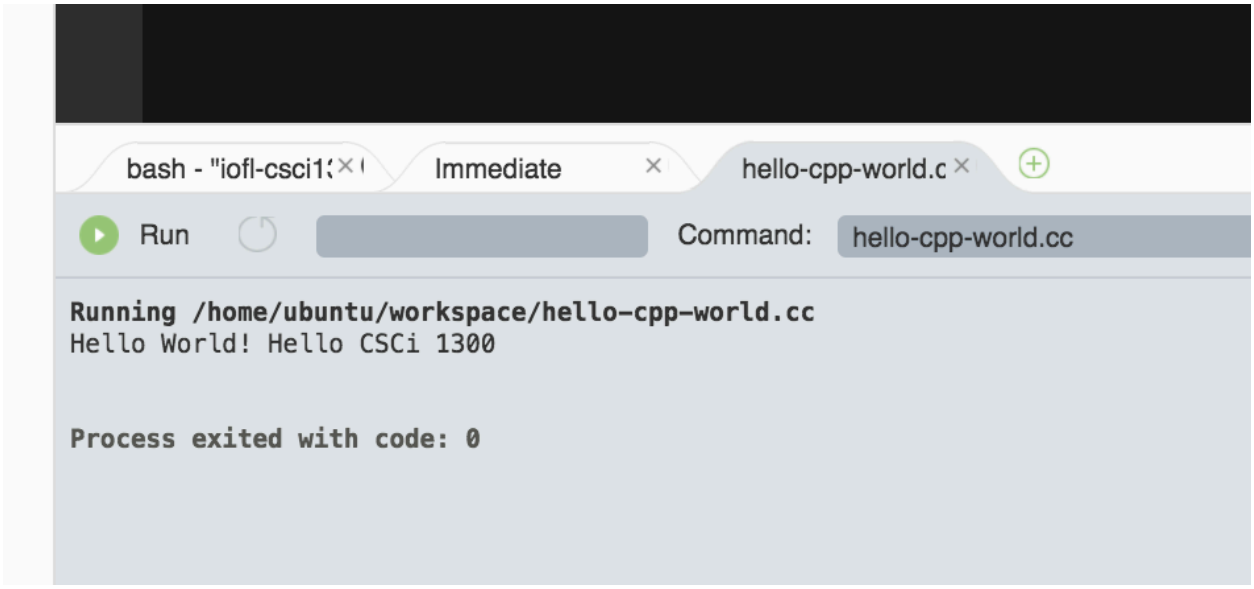
```
cout << "Hello World! Hello CSCi 1300" << endl;
```



7. Now let’s run it and see what happens. Click on the Run button above the Editor window. A new tab should open at the bottom of the screen. It is the *console* in which our program will run. You should see the name of our file as the tab title, and then the result of running the program in the console:

```
Running /home/ubuntu/workspace/hello-cpp-world.cc
Hello World! Hello CSCi 1300
```

The last line: “`Process exited with code: 0`” indicates our program ended “successfully”. We will see in class that one of the attributes of an algorithm or a program is that it has to terminate. *hello-cpp-world* terminated successfully. **Show it to your TA**!

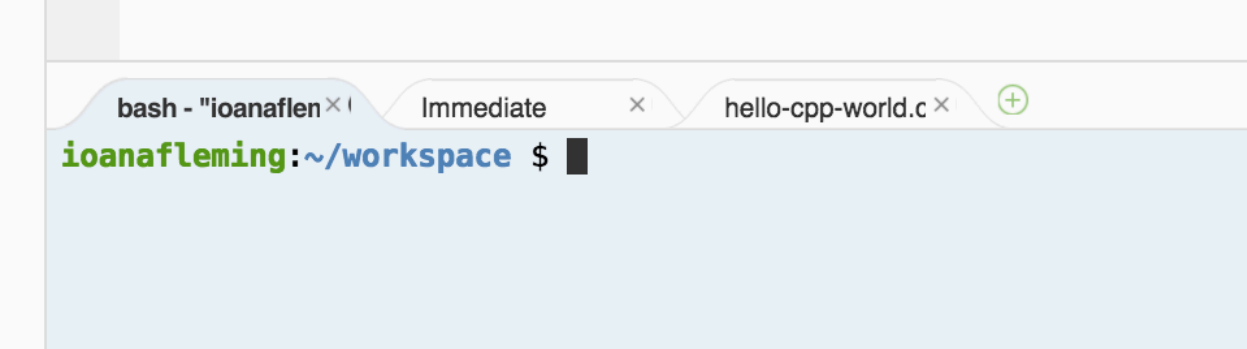


Basic Linux Commands:

Switch to the *bash* tab and let's learn some useful Linux commands.

You see `<somethingA>:~/workspace $`

- **SomethingA** = what your name is (the account name/your login name). You can also think of it here as your computer's name, or your drive's name.
- **~/** (i.e. everything else before the $) = your current directory (**think of a directory as the folder you're in**). When you start you are in a folder named *workspace*
- **~** is shorthand for the current user's home directory.
- **$** = end of prompt for entering a command.

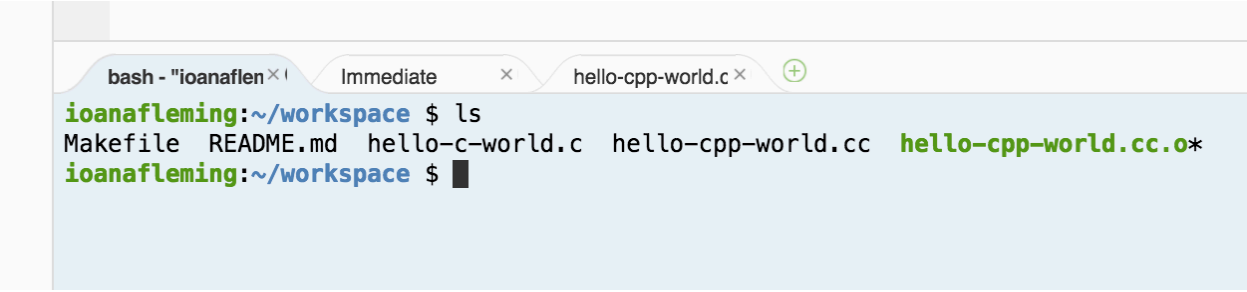


File browsing using the terminal is like using Windows explorer or like clicking on folders and navigating to different folders on your laptop. In the terminal, instead of clicking on folders we use commands to tell the computer what we want. If we want to go to a folder where we saved our last homework, we can type the commands to navigate to that folder and display its contents.

Try these commands:

1. `ls` = (that's a lowercase "L" not an uppercase "i") stands for *list* and is used to 'list' or show you everything in the current directory.

   Note: after you type the command and press ENTER, something is displayed as a result of the command, but then the prompt returns.

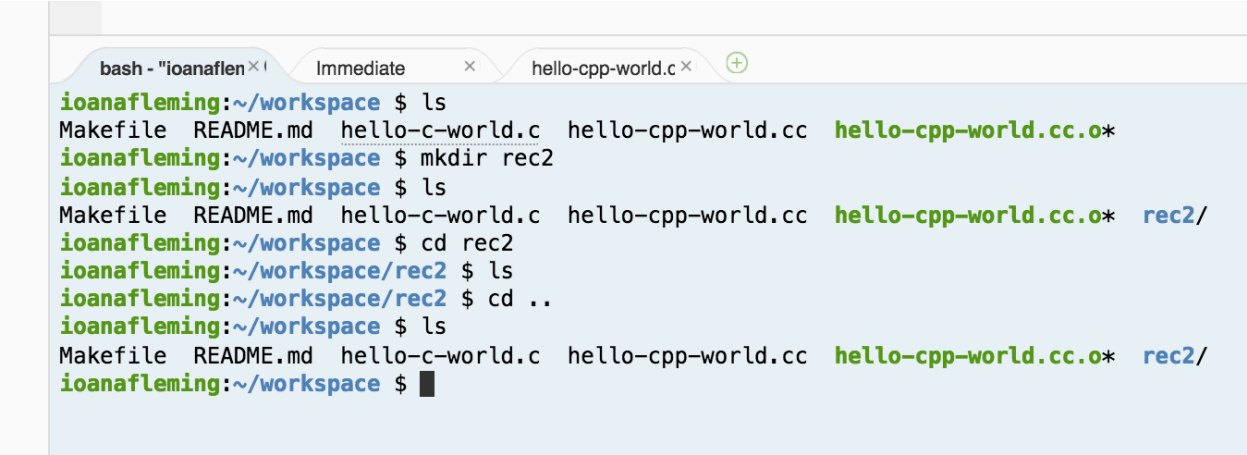


2. `mkdir my_folder` To create a new directory, use the command *mkdir*
   Note: Spaces are troublesome on the command line, so we'll use underscore.

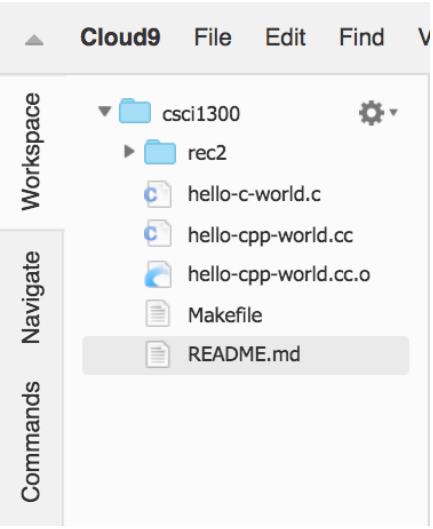
   Let's create a directory for today's recitation, named *rec2*

   ```
   $ mkdir rec2
   ```

rec2 is now a directory/folder, nested under the folder *workspace*. If you want to list the contents of the folder *workspace*, you will now notice a new item: *rec2/*, where the forward-slash (/) indicates rec2 is a subdirectory.



You will also notice a new folder in the Workspace tab, to the left of your window:



3. cd = stands for *change directory* is just like changing folders. It means, take me to place X. Commonly used as cd <name_of_directory>. **Note that there will always be a space between cd and the name of the directory that you want to navigate to and the name of the directory will **not** include the carrots (side arrows) displayed above.**
    - **Note:** cd takes you places in reference to your current location. It's like going into a folder, and then clicking on a folder within that folder, and then clicking on another folder within that folder. You will always navigate deeper within that folder. To back up, we use "cd .." (explained in a little bit).
    - **Note:** cd (and other commands) is case sensitive. 'rec2' does not equal 'Rec2'. Make sure you type in directory names exactly as they are spelled.
4. "cd .." = go to the parent of my current location. It's essentially backing up.

Want to learn more about linux commands?
http://community.linuxmint.com/tutorial/view/244 has a list of categorized linux commands. Codecademy also has a course covering the command line (https://www.codecademy.com/learn/learn-the-command-line).

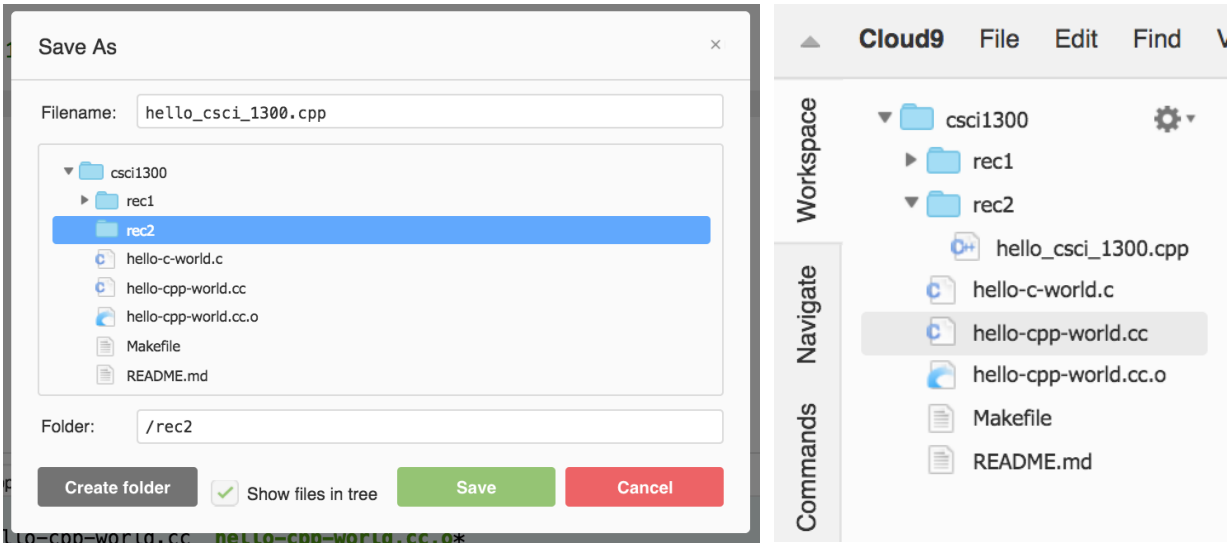
Fun tips and tricks

- **Tab Complete** = if you're typing something in the command line that's very long, but unique, you can hit tab when you're partway through and it will try to fill in the rest (kind of like auto complete). If it doesn't, and you tap tab twice, it tells you everything it has as options.
- **Command history browsing** = if you have typed a command (e.g. gedit myFile.txt) and want to repeat it, just press the up arrow. It will bring up your last executed command. Pressing up again will go to the one before. Pressing down will go forward in time through the list.
- ASCII is a character encoding standard. It includes numerical characters: 0-9, letters A-Z and a-z, punctuation, and blank spaces. These character codes represent text in computers. For example, the ASCII representation of 'D' is 01000100, and the ASCII representation of 'd' is 01100100. Notice the difference? 'D' is different from 'd', therefore, it is important for you to always make sure that you have proper capitalization and spelling when you are typing from the command line.

More Linux Commands

Lesson: Copying Files
(**Note:** Be very careful not to overwrite a file when copying.)

1. First, click inside the editor window, where the file *hello-cpp-world.cc* is open. Go up to the menu and choose File -> Save As.. and, when the dialog box opens, change the name of the file to *hello_csci_1300.cpp*. Choose *rec2* as the destination folder.



Once you click Save, the new file will appear in the file tree, in the Workspace tab (see the figure on the right, above).

2. Let's make another directory named *rec1*.

```
$ mkdir rec1
```

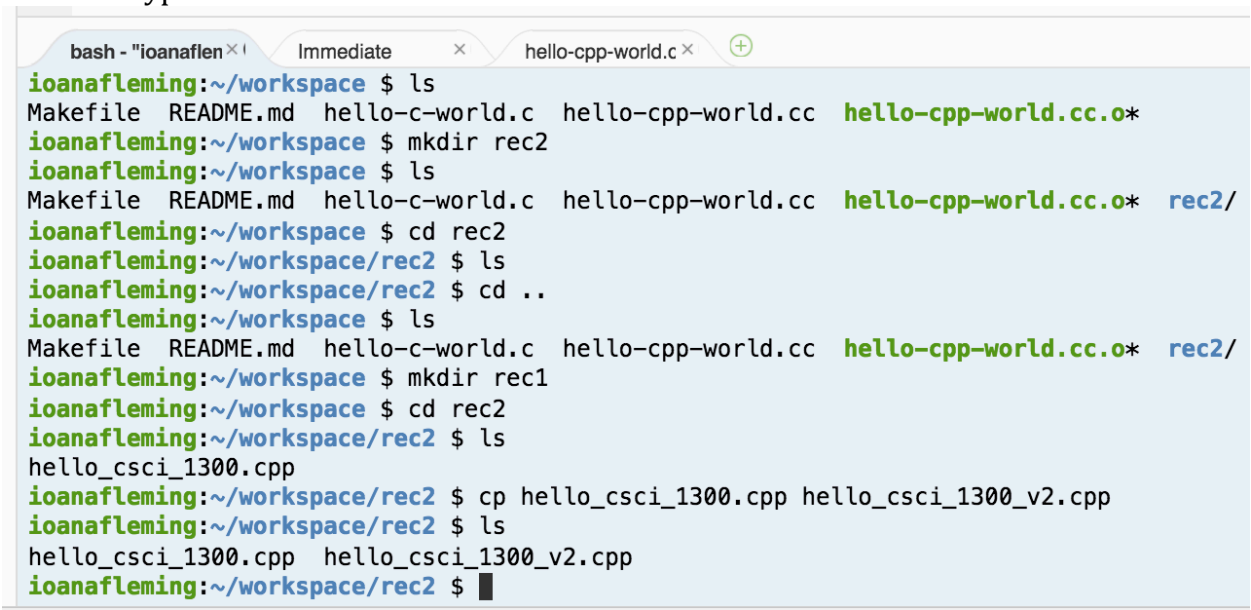
3. Now we want to copy the file from *rec2* into *rec1*. First, we go into the *rec2* directory:

```
$ cd rec2
```

Ok, now we use the command for copying files:

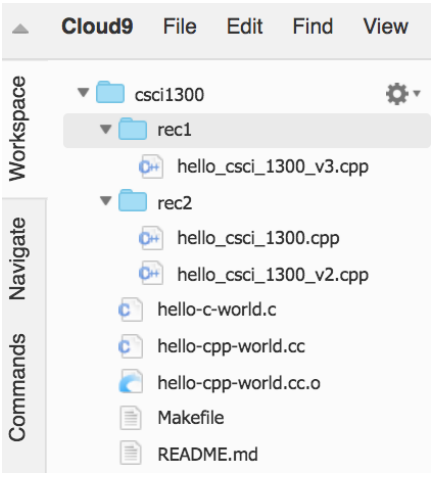
```
$ cp hello_csci_1300.cpp hello_csci_1300_v2.cpp
```

Now let's lype *ls* and see what we have:

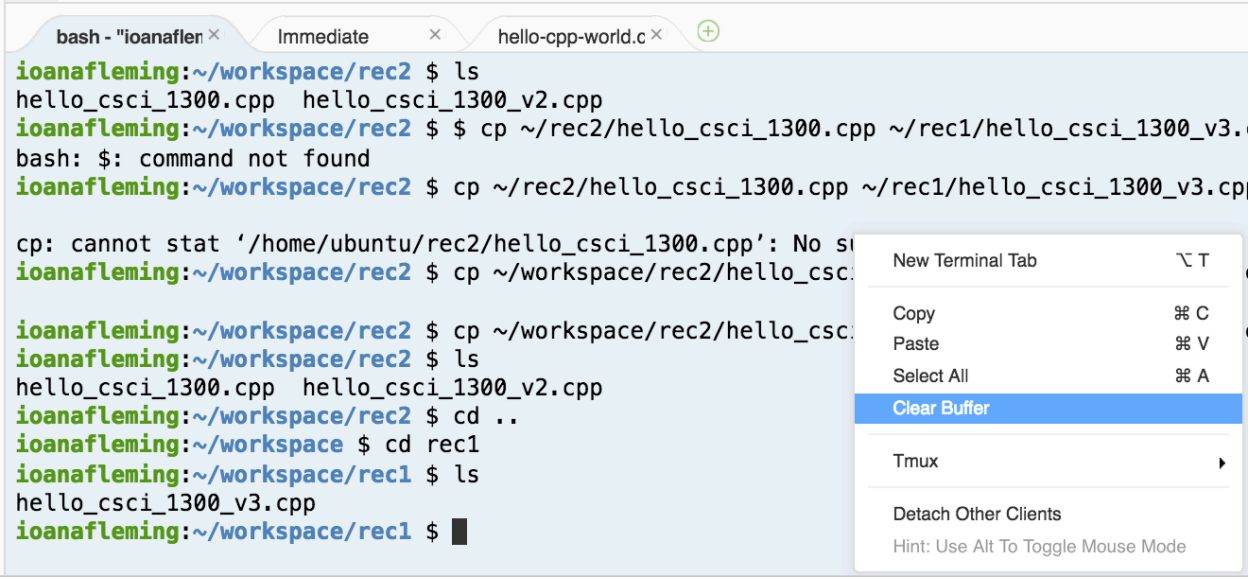


Since we did not specify the path for the file we wanted to copy, nor for where we wanted it copied, we ended up with an identical copy of the file, both of them inside the folder *rec2*.

Let's do this again, including the paths:



If your terminal window has gotten full, you can always clear it. Right-click anywhere in it, and choose "Clear Buffer"

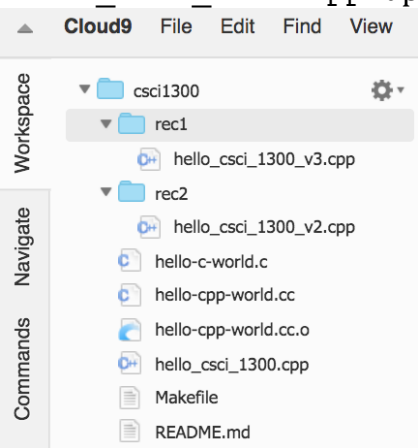


Lesson: Moving Files

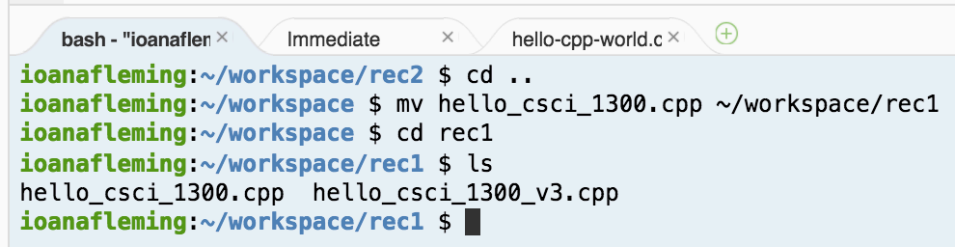
It's possible to move files around using the command *mv*. Starting in directory *rec2*, type:

```
$ mv hello_csci_1300.cpp ..
```

Remember, two dots indicate the *parent* directory (one level up). Now if we list the files in *rec2*, there is only `hello_csci_1300_v2.cpp` left. And we will the file `hello_csci_1300.cpp` up in the root directory:



Let's move one more file into the *rec1* directory:

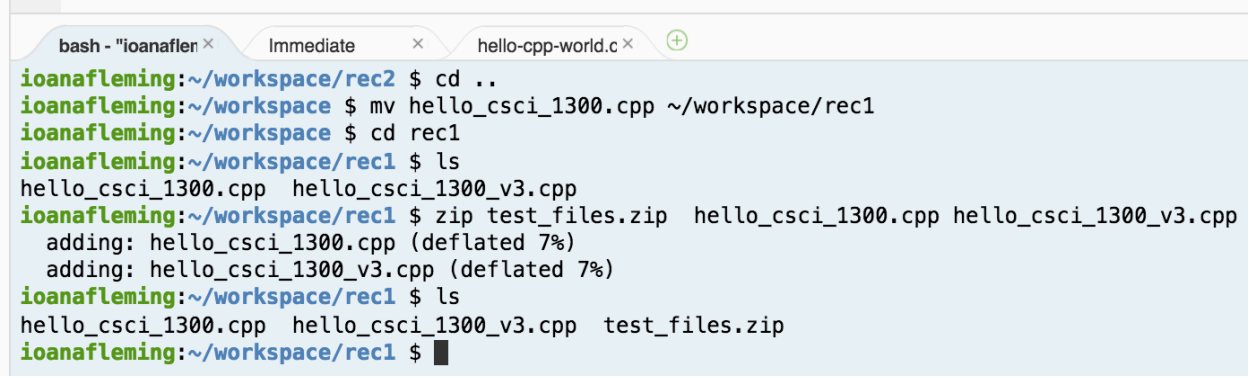


Lesson: Zipping and Unzipping Files

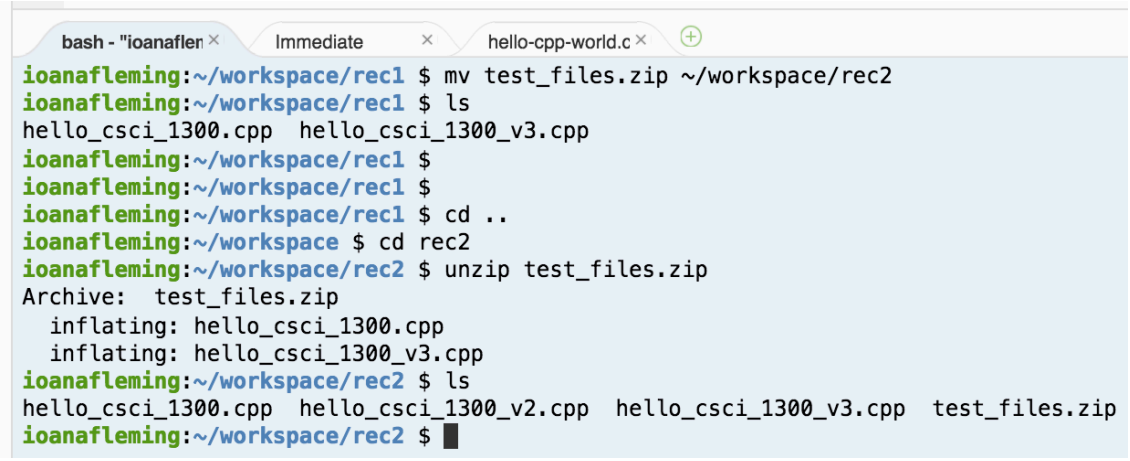
An important thing to know for this class is how to zip your solution files into one assignment submission file. To do this, there's a convenient command called zip. Let's zip up the two files in the directory *rec1* into a single zip file.

```
$ zip test_files.zip  hello_csci_1300.cpp hello_csci_1300_v3.cpp
```

The first argument after zip is the name of the zip file we want to produce. The files listed next are the files that go into this zip file. If you list your directory now, you should see three files. Notice that the original files are still present--zip and unzip don't destroy any files.



Let's now move the .zip file into the *rec2* directory, and try to unzip the files:



You should see the three files now in the directory. Note that the original zip file didn't get destroyed.