Lab Intro

Due Jan 16 by 10pm **Points** 1

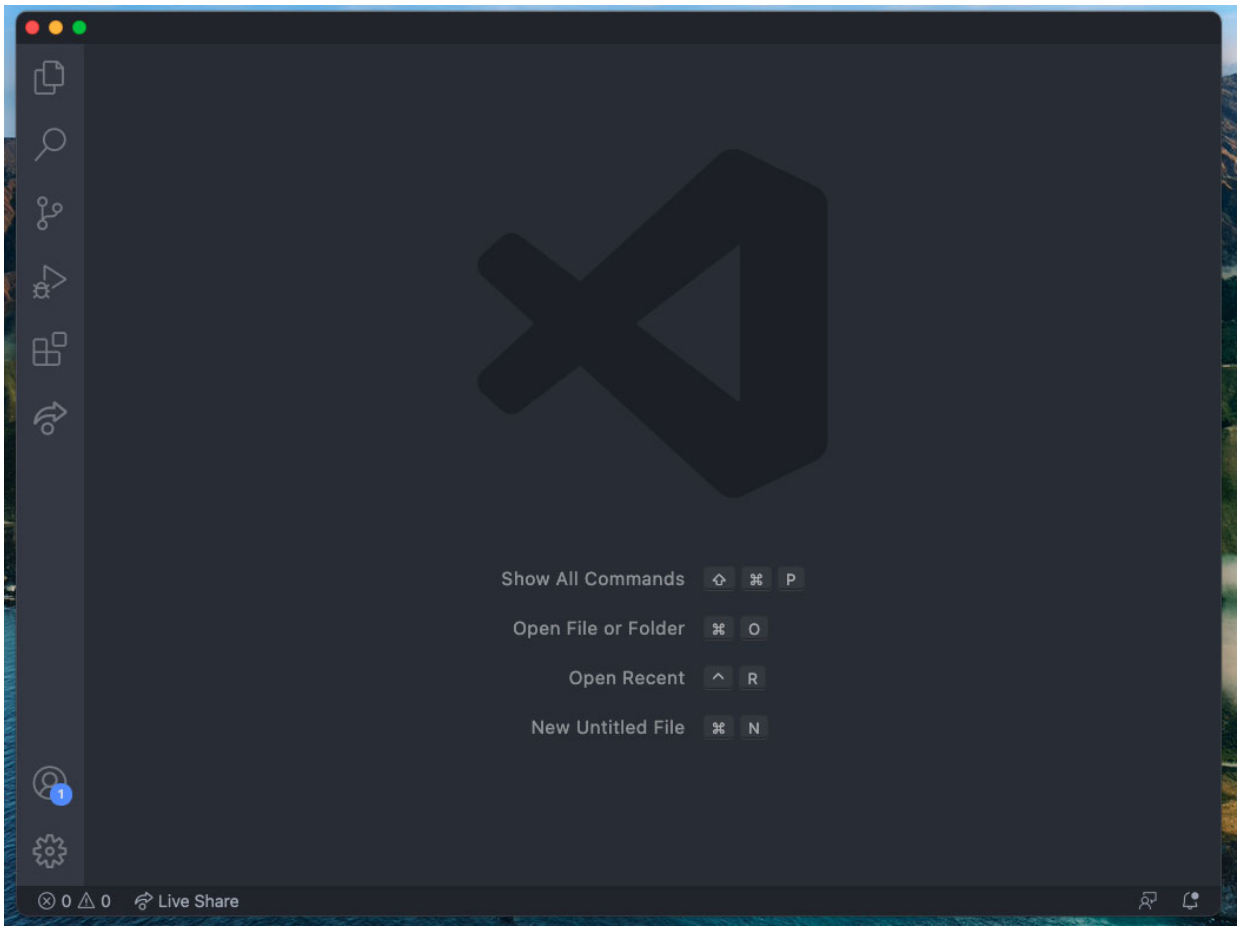
Welcome to CPSC 221!

In this class, you will be using C++ and the Linux operating system to aid your journey in becoming a data structures expert.

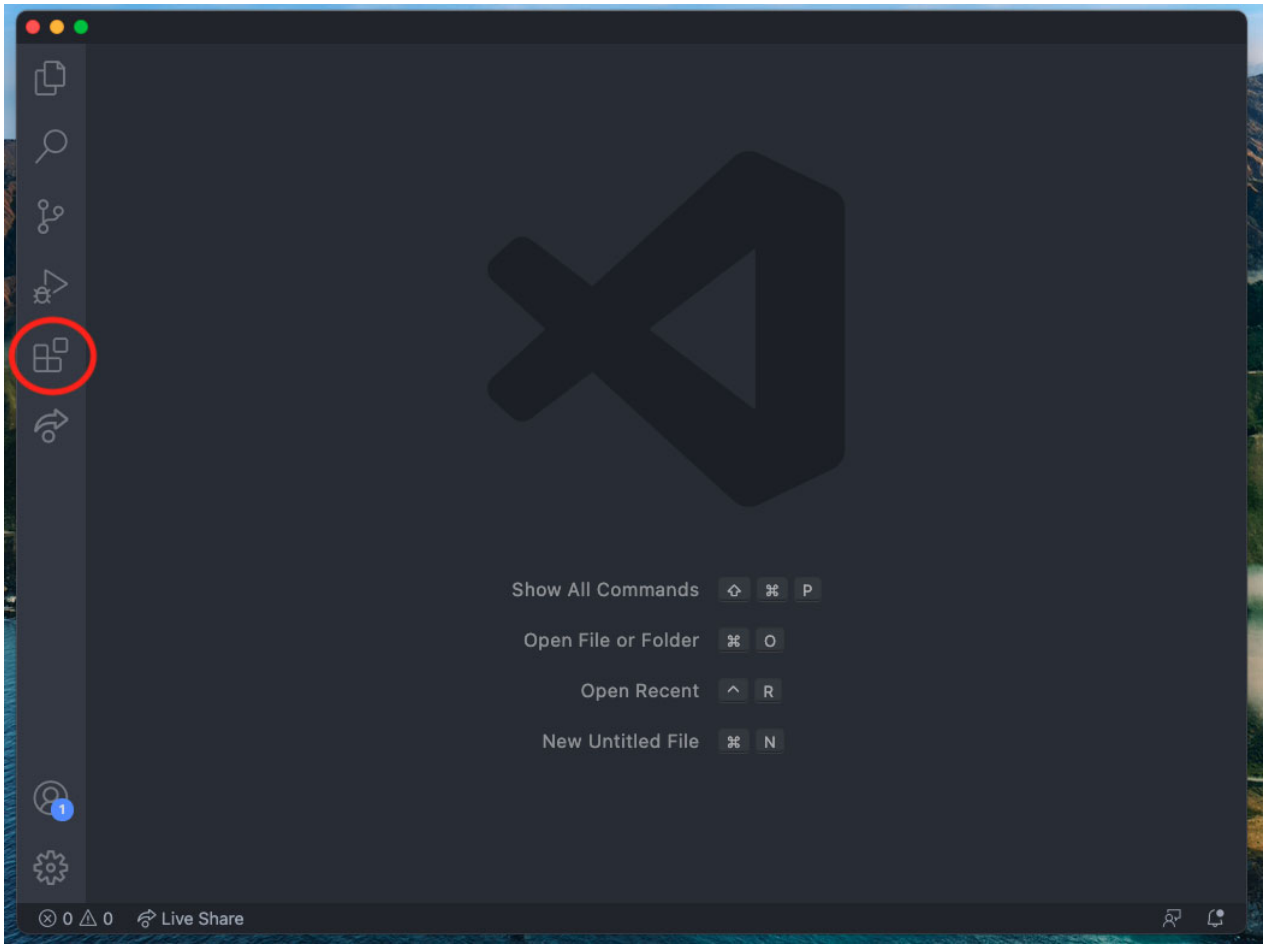
If you'd like to access your lab account from your own computer, you may log in remotely using your CWL account:

Visual Studio Code Set Up

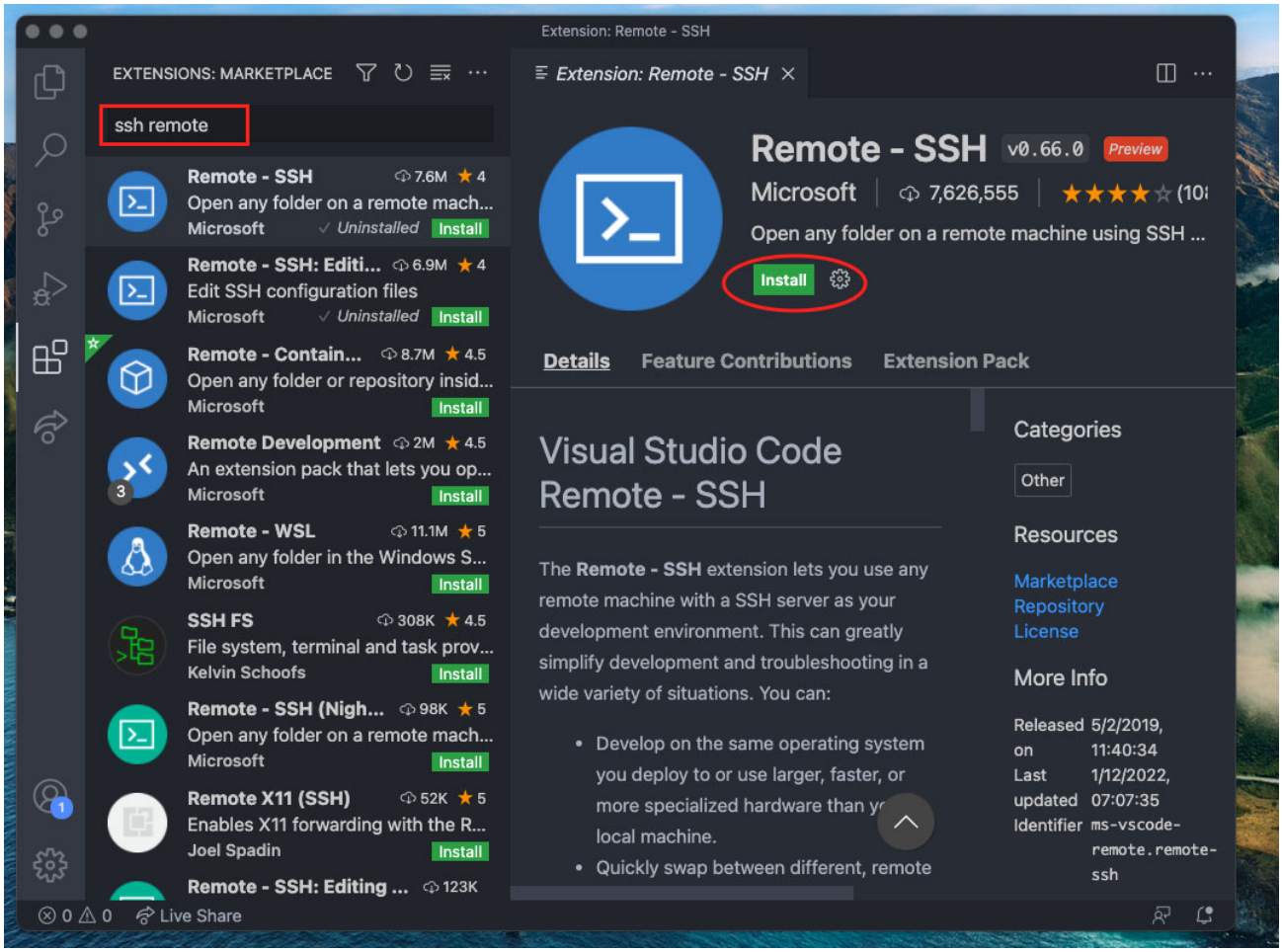
1. Download VS Code **here** **(https://code.visualstudio.com/download)**
2. Once you have opened VS Code it should look like this:



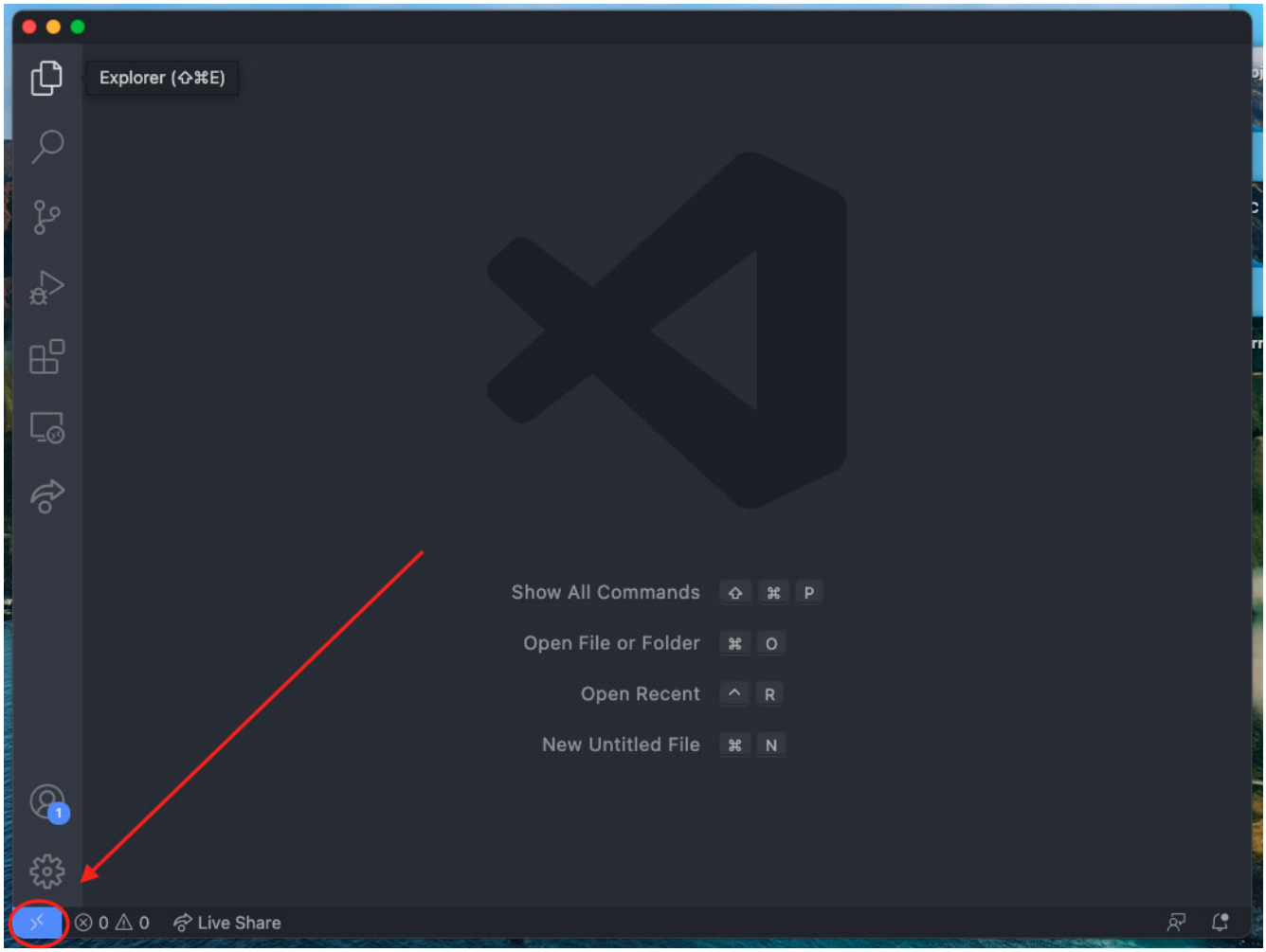
3. Select Extensions on the left side bar



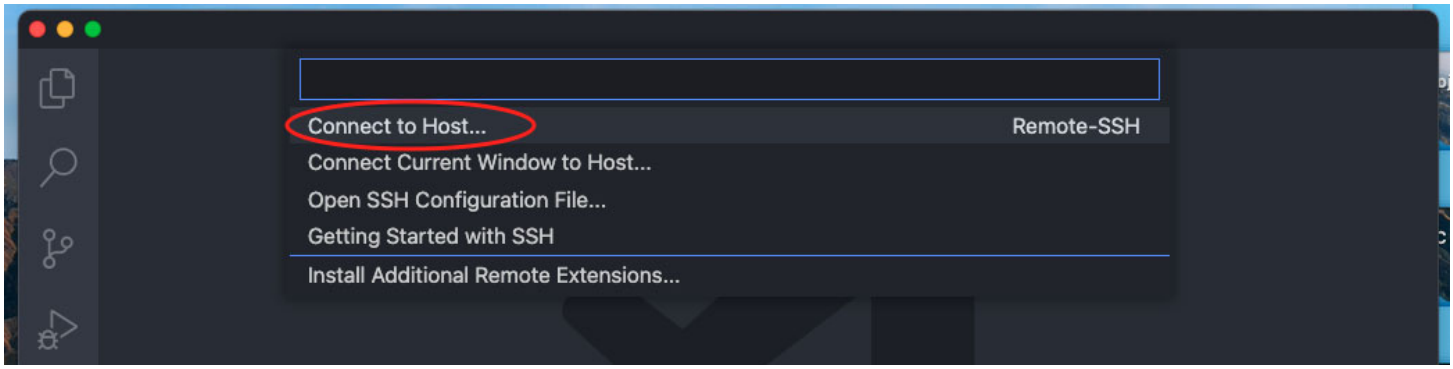
4. Using the Search Bar, type SSH - Remote and install the extension



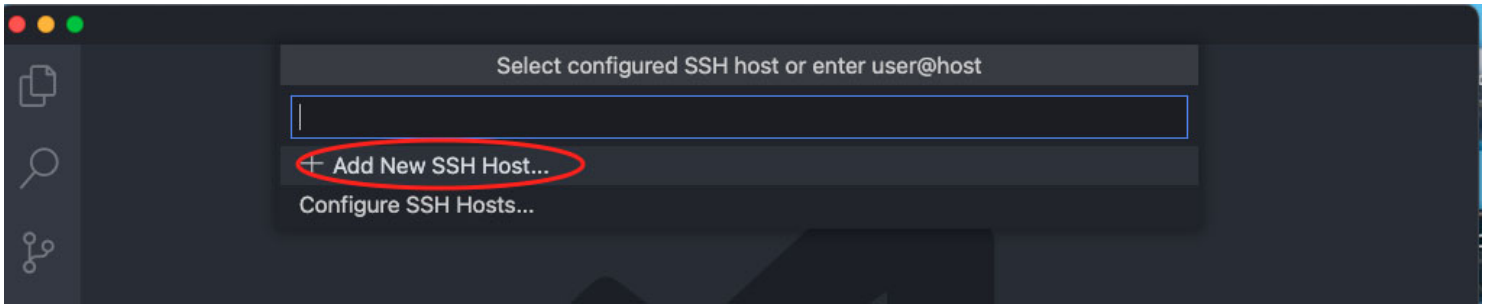
5. Once download is successful you should **press** on the new icon that will appear in the bottom left corner:



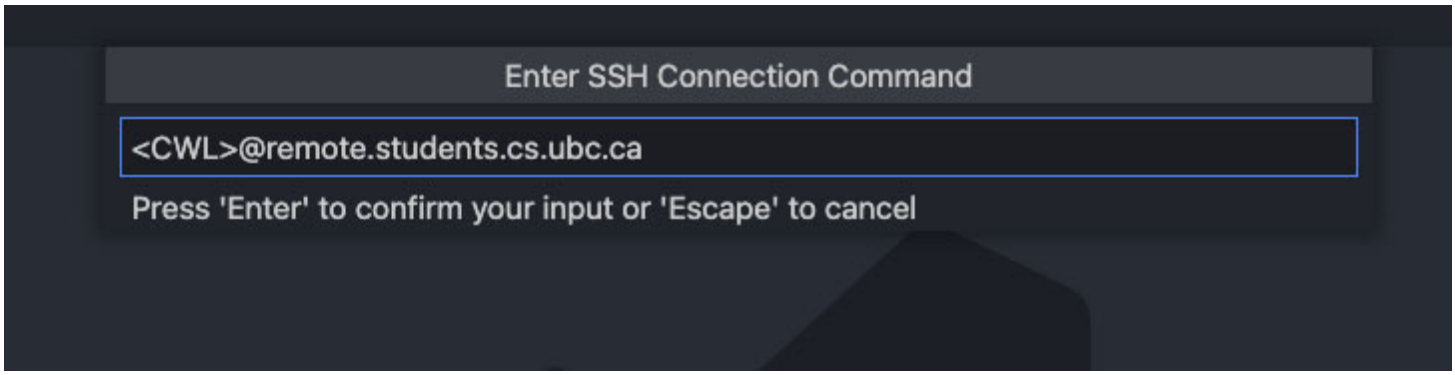
6. It will prompt you to connect to a host



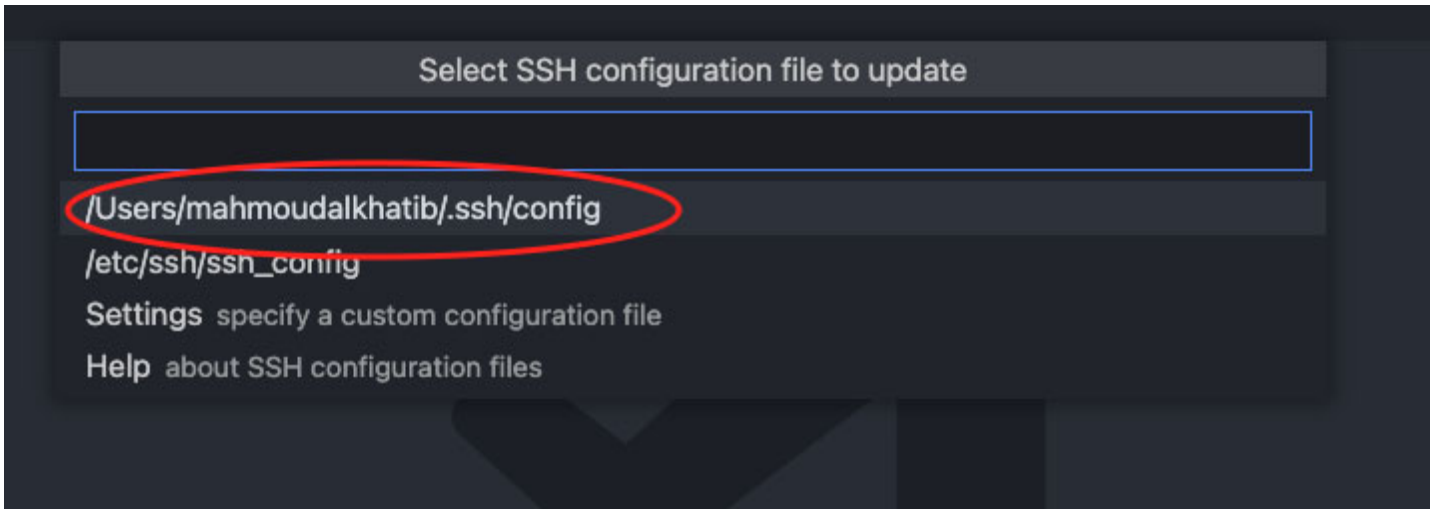
7. Select Add New SSH Host



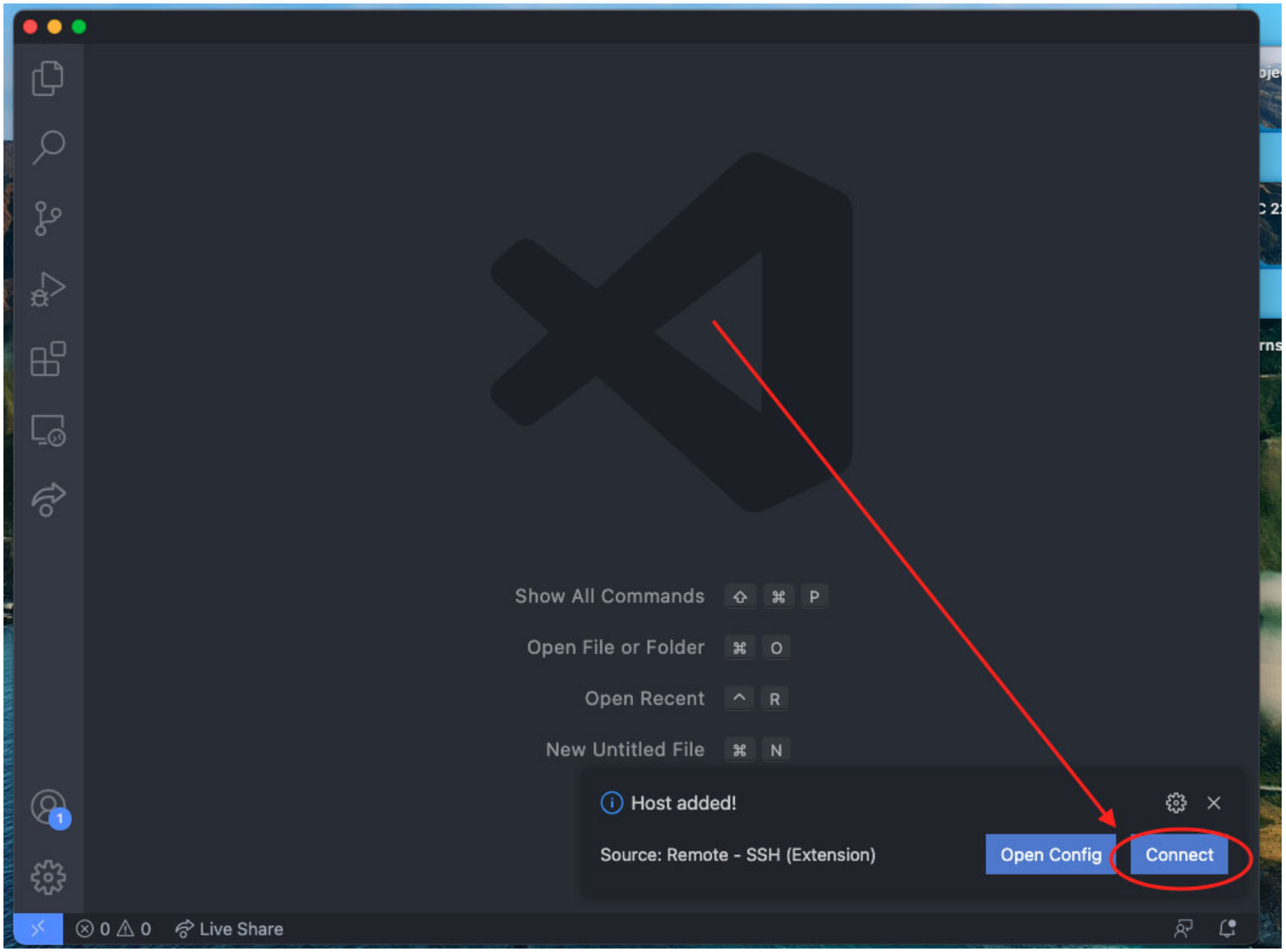
8. write `<your-CWL-id>@remote.students.cs.ubc.ca` note that you need your CWL ID not your CS ID. Also, there is plenty of other servers than remote but remote is just used for example (Ask your lab TAs for the other servers or check Piazza)



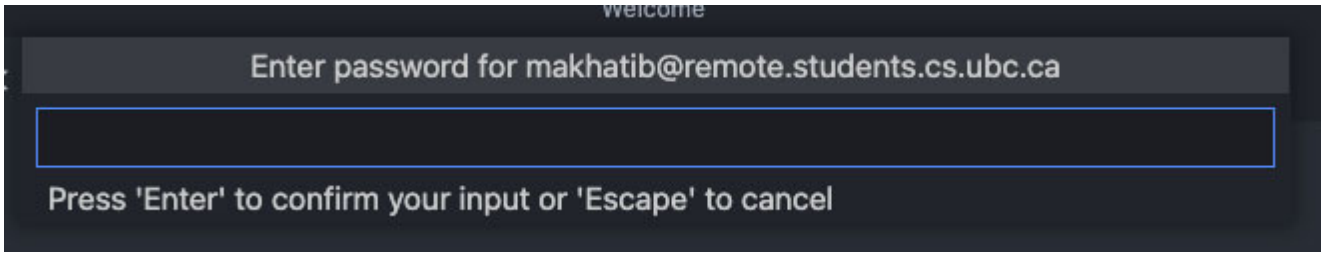
9. You can choose this option for config. If you are a Windows user, it could prompt you to choose between MacOS, Linux, Windows. Please choose Linux.



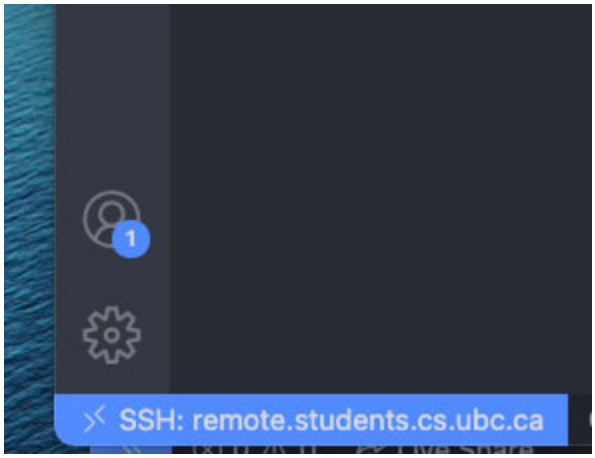
10. This window will pop up, choose Connect



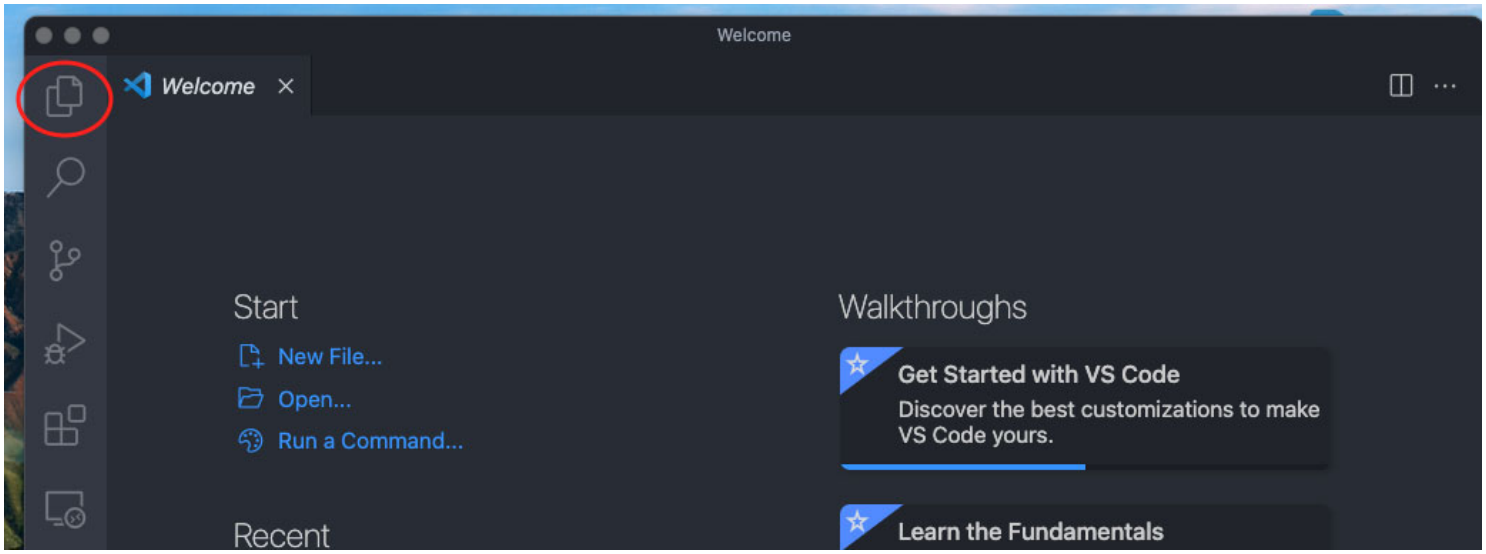
11. Add your CWL password



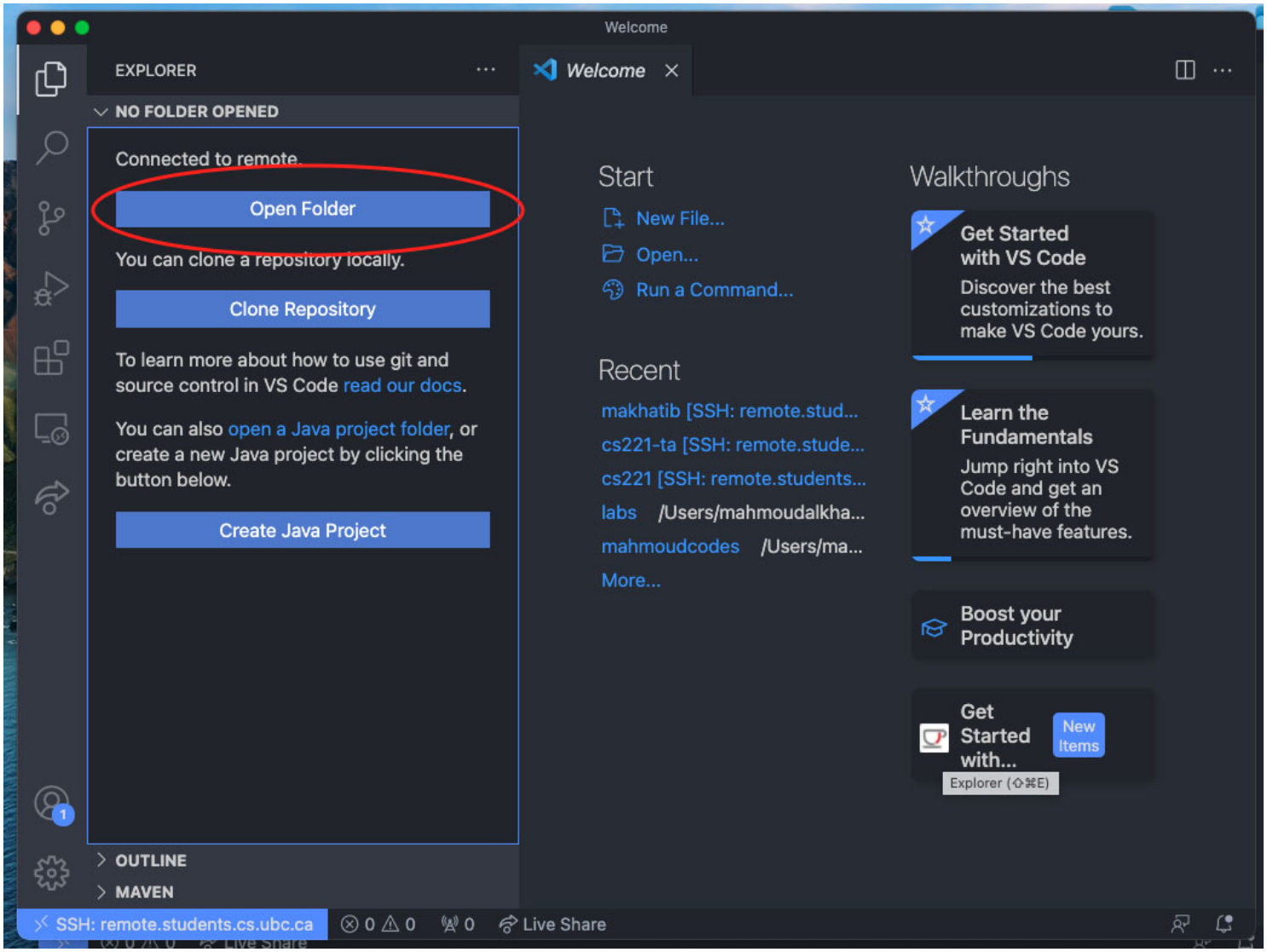
12. If you see this on the bottom left corner, it means your connection to our remote servers was successful



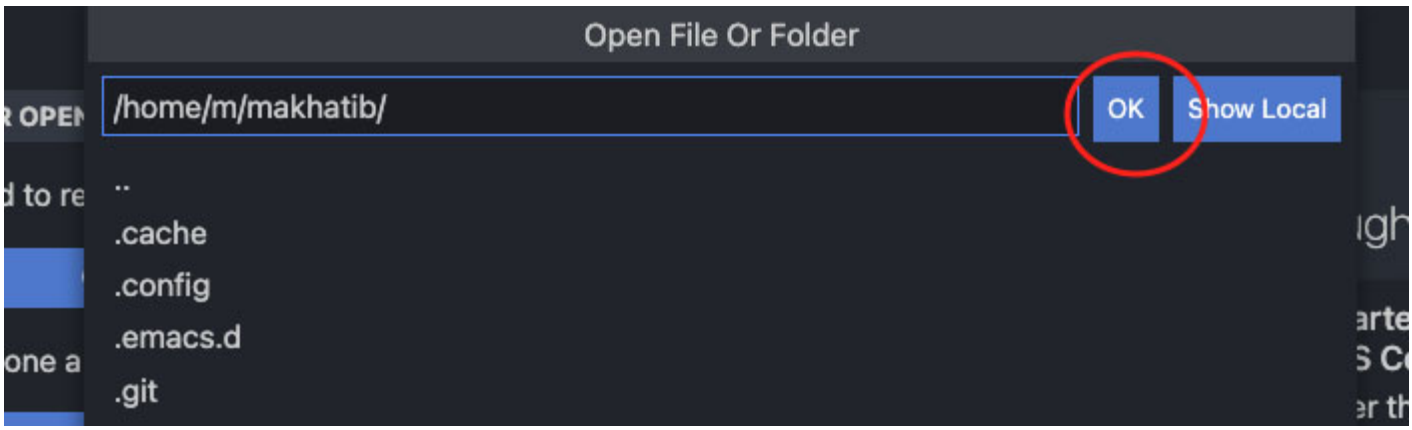
13. Now press on Files Explorer



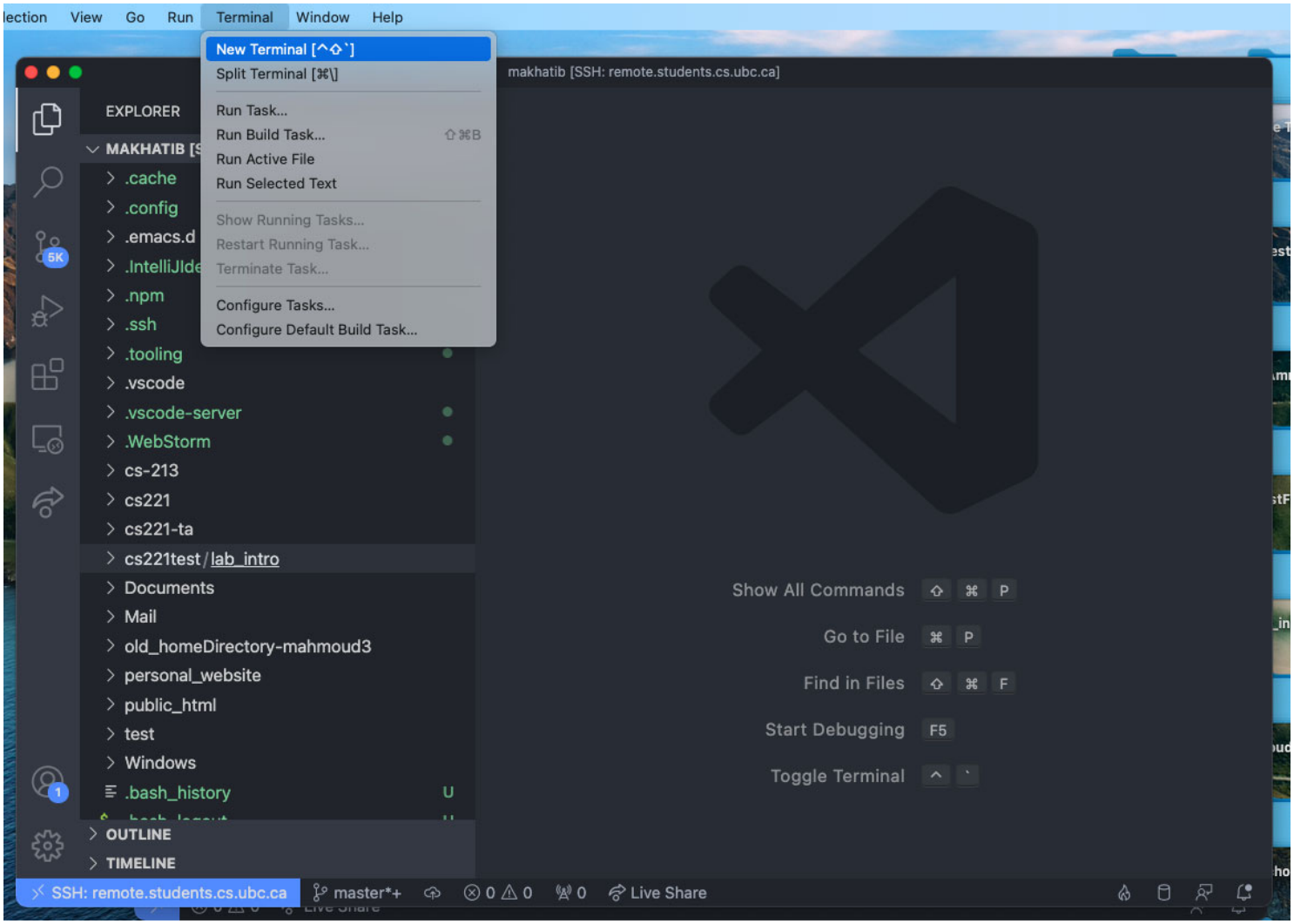
14. Select Open Folder



15. Choose OK



16. You can open a new terminal instance as shown below. You can now skip to "Command Line Tutorial" part of this Canvas page. Alternatively you can use VS Code graphical interface to create a folder and drop your *unzipped* lab/PAs files in there.



Windows

If you are on Windows, simply install **XManager** (https://my.cs.ubc.ca/docs/free-terminal-emulation-software-xmanager) . You'll be able to use **Xshell** like a UNIX terminal in the lab, and **Xftp** to transfer files. You can login into the ugrad servers either by creating a new remote connection (leave the remaining fields as default):

```
Name: ugrad
Host: remote.students.cs.ubc.ca
Protocol: SSH
```

Alternatively, type the following into XShell:

```
    ssh <your-CWL-id>@remote.students.cs.ubc.ca

    Example:
    ssh johndoe1@remote.students.cs.ubc.ca
```

Mac OS

If you are on Mac OS, download **XQuartz** (https://www.xquartz.org/) to enable graphics packages such as `gedit` and `xdg-open`. Mac OS has a built-in Terminal - navigate to `Applications > Utilities > Terminal` where you can login using:

```
ssh -Y your-CWL-id@remote.students.cs.ubc.ca
```

Linux

If you are on a Linux machine, you can find the Terminal app by navigating to `Applications > System Tools > Terminal` located at the top of your screen. (You can also drag this icon to the menu bar at the top of the screen for the future.) Login using:

```
ssh your-CWL-id@remote.students.cs.ubc.ca
```

Finally, if you navigate to a folder using Places and right-click inside it there is an option Open In Terminal which will open the terminal. The Terminal app is the command line interface for Linux, where you run programs by typing their names, and sometimes parameters such as files to open, instead of clicking on icons and buttons.

You should be familiar with the Linux command line, a useful skill not only useful for CPSC 221 but also for many other courses and a lot of power uses of operating systems.

Command Line Tutorial

After you log in to `remote.students.cs.ubc.ca` type:

```
pwd
```

and press enter. You'll see the folder you are currently located in, called the working directory. Since you haven't moved anywhere yet, this is your home directory.

Later on we'll make a directory (folder) to store your coursework in, but for now, let's create a test directory. Type the following in the terminal:

```
mkdir cs221test
```

Now let's look at the contents of the folder we're in (still your home directory):

```
ls
```

You'll probably see several folders (in blue). One of these should be `cs221test/`. Let's open `cs221test/`:

```
cd cs221test/
```

If you type `pwd` again, you will see you are now in `cs221test/` inside of your home directory. Finally, let's get back to your home directory. Run one of:

```
cd ..
cd
cd ~
```

All of them will return you to your home directory. The first (`cd ..`) navigates to the parent of the current working directory. The second is a shortcut builtin to `cd`: running `cd` without any parameters will navigate to your home directory. The third (`cd ~`) uses the `~` (a tilde), which is a standard way to say "the home directory of the current user".

Now, create the `cs221` directory to store your coursework in:

```
mkdir cs221
```

Useful Linux commands cheat sheet:

```
pwd             # Print Working Directory - shows the folder you are currently in

ls              # List - lists the files in your current folder

cd FOLDER       # Change Directory to FOLDER - opens FOLDER
cd ..           #  - opens the parent folder of your current location
cd              #  - opens  your home folder

atom FILE &     # Opens (or creates) FILE in atom, a text editor
gedit FILE &    # Opens (or creates) FILE in gedit, a simple text editor
vim FILE        # Opens (or creates) FILE in vim, a simple modal text editor

rm FILE         # Remove - deletes FILE
rm -r FOLDER    #        - deletes a folder and its contents
mv SRC DEST     # Move - moves/renames SRC to DEST
cp SRC DEST     # Copy - copies SRC to DEST

make PROGRAM    # Compiles PROGRAM using the rules in the Makefile file

xdg-open IMAGE  # Opens IMAGE in the default image viewer

clear           # Clear the terminal screen (Ctrl-l also works)
reset           # Reset and clear the terminal
```

Keyboard Shortcuts for Bash:

```
Ctrl + a : Navigates to the beginning of the line
Ctrl + e : Navigates to the end of the line
Alt + b  : Back (left) one word
Alt + f  : Forward (right) one word
Ctrl + u : Clears the line from the cursor to the beginning
Ctrl + c : Kills the current command being executed (useful if you run into an infinite loop)
tab      : Attempts to autocomplete a command or file
uparrow  : Retrieve previously entered command
Ctrl + r : Enter a search for a previously entered command
```

More commands can be found **here** (http://ss64.com/bash/syntax-keyboard.html). Note that some of these commands may be unavailable for Mac OS X.

Check out the list above for useful Linux commands, and try using them!

Downloading the lab files

You can download `lab_intro` here: **lab_intro.zip** (https://www.students.cs.ubc.ca/~cs-221/2021W2/labs/intro/lab_intro.zip).

Next, upload the `lab_intro.zip` file into your remote undergraduate lab account, in the `cs221` directory. There are two ways to do so:

Using the command line (Mac OS/Linux)

On a Linux or Mac OS machine, you can use the command line to do so. Note where the file is saved - you'll need to `cd` into this directory in the command line for the next step to work properly.

```
# Make sure you're in the directory that contains lab_intro.zip!
scp lab_intro.zip your-CWL-id@remote.students.cs.ubc.ca:cs221
```

Using Xftp or Cyberduck (Windows/Mac OS/Linux)

On a Windows machine, you should open **Xftp**. If you are on a Mac OS or Linux machine, and you do not wish to use `scp`, you can also download a file transfer client, such as **Cyberduck** (https://cyberduck.io/). Create a new connection to your remote ugrad account. Set the following parameters (leave the other fields as their default setting):

```
Host: remote.students.cs.ubc.ca
Protocol: SFTP
Username: your-CWL-id
Password: your-CWL-password
```

In the next dialog, click `Accept and Save`, and then drag the `lab_intro.zip` file from your local file directory into the `cs221` folder of your remote ugrad machine.

Unzipping the .zip file

Next login to the remote server, and unzip the zip file located in the `cs221` directory.

```
ssh your-ugrad-id@remote.students.cs.ubc.ca
cd cs221
unzip lab_intro.zip
```

To open this directory, use the `cd` (change directory) command:

```
cd lab_intro/
```

To display the files in this directory, use the `ls` (list) command:

```
ls
```

Opening a Text Editor

In order to open and edit a text file, you need to open a text editor. There are hundreds of such editors and many people have a favorite one for different reasons.

Command line editors

There are three standard editors that work within a command line interface:

- `vim`, powerful but very un-friendly to beginners
- `emacs`, powerful but also un-friendly to beginners
- `nano`, simple and user-friendly for quick edits

If you are writing a large program, graphical editors provide advanced features in a beginner friendly form. Some popular, free graphical editors that run on all major platforms include:

- `atom`, open-source editor developed by github
- `brackets`, open-source editor developed by adobe
- `visual studio code`, free light-weight editor developed by Microsoft
- `gedit`, simple, open-source editor default to most Linux distributions

Note that for Mac OS users, `gedit` will open only if the `-Y` flag was specified when running the `ssh` command to login to the remote ugrad server, and XQuartz is installed correctly (see Getting Started section above).

Using a file transfer client

On the other hand, you can also use your file transfer client (Xftp, Cyberduck etc.) to edit the files locally using your favourite text editor. Login to your file transfer client, and find the file to edit. Right-click the file, and click `Edit`. **(Do not click)** `Open`. This will open the file using the default text editor. Any changes made to this file will be automatically saved onto the remote server. You can change your default editor under `Tools > Options` in Xftp, or `Cyberduck > Preferences > Editor` in Cyberduck. Some good examples of text editors include Atom, Notepad++, and Sublime Text.

More tips and tricks & optional readings

The **Computing Environment** (https://www.students.cs.ubc.ca/~cs-221/2019W2/info/environment/) page has more information about machine setup. There are also important (but optional) readings to learn more about C++ coding.

Part 2 - Lab Assignment

Understanding HSL Color Space

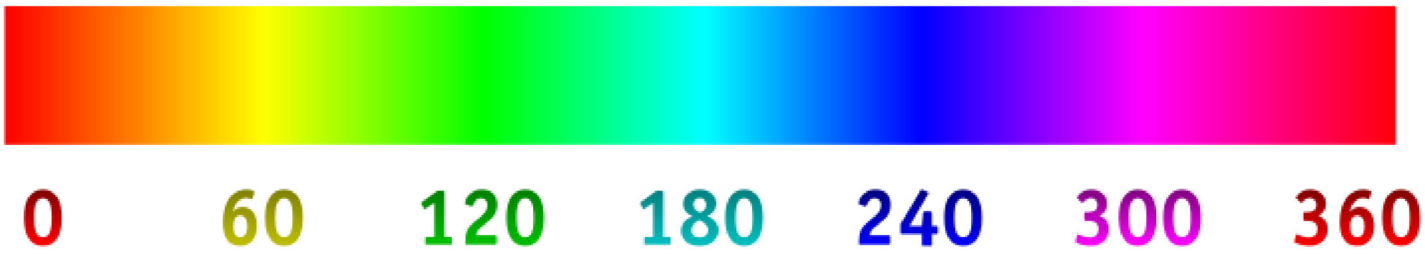
HSL – hue, saturation, and luminance – is a representation of color that tries to be intuitive and perceptually relevant. HSL can represent all of the colors that can be represented by RGB.

Hue

The **hue** of a color is the color component – red, green, blue, yellow, etc. Hues are represented on a circle in terms of degrees [0, 360]. For example:

- “UBC Yellow” has a hue of 40
- “UBC Blue” has a hue of 210

The full hue spectrum can be seen in this image from Wikipedia:



Saturation

The **saturation** is the intensity of a color, on a scale of [0, 1]. A saturation value of 0 is gray, without color.

Luminance

The **luminance** is brightness applied to the color, on a scale of [0, 1]. A luminance value of 0 is black and a luminance value of 1 is white. A higher luminance value results in a lighter/whiter version of the same color.

Writing your first class: HSLAPixel

Create a class called `HSLAPixel` whose functionality is described here:

A HSLA pixel is characterized by 4 doubles: `h` representing hue, `s` representing saturation, `l` representing luminance, and `a` representing the transparency value typically called alpha.

You will create 3 constructors:

- The default contstructor `HSLAPixel()` sets the default pixel to white, which is a pixel with any hue, saturation 0, luminance 1.0, and alpha 1.0.
- A 3 argument constructor whose parameters are used to set the hue, saturation, and luminance.
- A 4 argument constructor whose parameters are used to set the hue, saturation, luminance, and alpha.

A few important things to remember:

- All of the members of `HSLAPixel` are `public`,
- `HSLAPixel` is part of the `cs221util` namespace,
- Following convention, the class definition should be placed in a file called `HSLAPixel.h`, and the member function implementations should be placed in a file called `HSLAPixel.cpp`.
- `HSLAPixel.h` and `HSLAPixel.cpp` should both be created in the `cs221util` directory inside `lab_intro`.
- Make sure to add the include guards in `HSLAPixel.h`!

Compile it!

A `Makefile` has been provided for you for this lab (you'll make your own soon!). To compile your program, run:

```
make
```

If `make` fails, you will see error messages. We use `clang`, which aims to provides descriptive error messages that try to help you not only spot the error but also will provide a suggestion on how to fix the bug.

If `make` runs successfully, you will see three warning messages:

```
lab_intro.cpp:57:36: warning: unused parameter 'centerX' [-Wunused-parameter]
PNG createSpotlight(PNG image, int centerX, int centerY) {
                                   ^
lab_intro.cpp:57:49: warning: unused parameter 'centerY' [-Wunused-parameter]
PNG createSpotlight(PNG image, int centerX, int centerY) {
                                                ^
lab_intro.cpp:91:35: warning: unused parameter 'secondImage'
[-Wunused-parameter]
PNG watermark(PNG firstImage, PNG secondImage) {
                                  ^
```

This is expected – you have not written these functions yet.

Writing the PNG manipulation functions

The rest of this lab assignment uses the `HSLAPixel` class to manipulate an image. A small program has been written in `main.cpp` that loads `rosegarden.png`, calls various manipulation functions, and writes out the images as `out-_____.png`.

All of these manipulation functions are in `lab_intro.cpp`. The first one, `grayscale`, has been written for you already and transforms `rosegarden.png` to grayscale:

`rosegarden.png`

out-grayscale.png

You should view these files for yourself from your own program to verify they look the same! Continue working through `lab_intro.cpp` to complete the remaining functions.

Helpful Functions:

```
image.resize(1024, 768);    // Changes the size of the PNG image to be 1024 (width) by 768 (height) p
ixels.
image.height();             // Returns the height of the PNG image.
image.width();              // Returns the width of the PNG image.
HSLAPixel *pixel = image.getPixel(x, y);  // Gets a pointer to the memory storing the pixel at (x, y)
```

Compiling the Code

To compile your code, run the following:

```
make
```

Testing the Code

After compiling your code, an executable named `lab_intro` should be located in your working directory. To test your code, run `lab_intro`:

```
./lab_intro
```

This will make several png images as `out-*.png` in your current directory, where `*` denotes the manipulation done to the image. You can view it by opening the image or, on the command line, by running:

```
xdg-open out-grayscale.png
xdg-open out-ubcify.png
xdg-open out-spotlight.png
xdg-open out-watermark.png
```

(Once again, Mac OS users should specify the `-Y` flag when `ssh`-ing into the remote ugrad server.)

Marking Your Work

We'll be grading all of the labs this semester using an autograder provided by PrairieLearn. The lab will appear under the Assessments heading, and you will be able to drag and drop your required files into that space.

For this lab, please submit files:

- `HSLAPixel.h`
- `HSLAPixel.cpp`
- `lab_intro.cpp`

Note that the test cases we run for lab work are minimal. They are mainly a mechanism for letting us know that you have engaged in the lab material, and not an accurate reflection of the quality of your work. Your mark for the lab depends only on the submission score, and not passing even those minimal tests.