

Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3

Visual Studio Code: SSH Keys, and Remote Building/Debugging

Below are the links for more robust guide:

<https://code.visualstudio.com/docs/remote/ssh>

https://code.visualstudio.com/docs/remote/troubleshooting#_installing-a-supported-ssh-client

<https://code.visualstudio.com/docs/cpp/cpp-debug>

<https://code.visualstudio.com/docs/cpp/launch-json-reference>

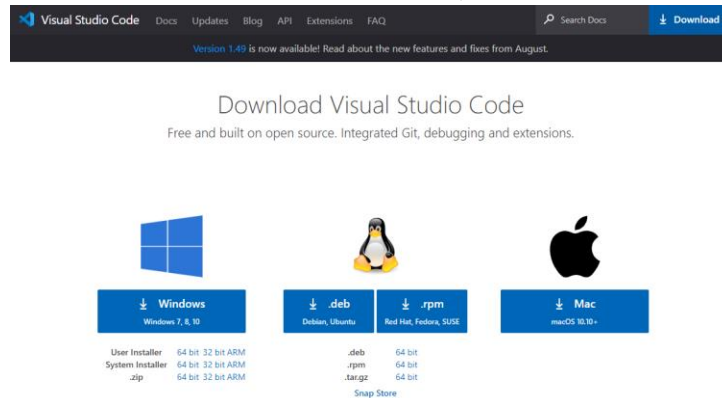
<https://code.visualstudio.com/docs/cpp/config-linux>

The intention of this guide is to simplify the process of connecting to your school server using visual studio code. This guide may also be applicable with other editors. Some notable editors are **Atom**, **Sublime**, **Notepad++**. You do not have to use Visual Studio Code. You do have to use the school's compiler, though.

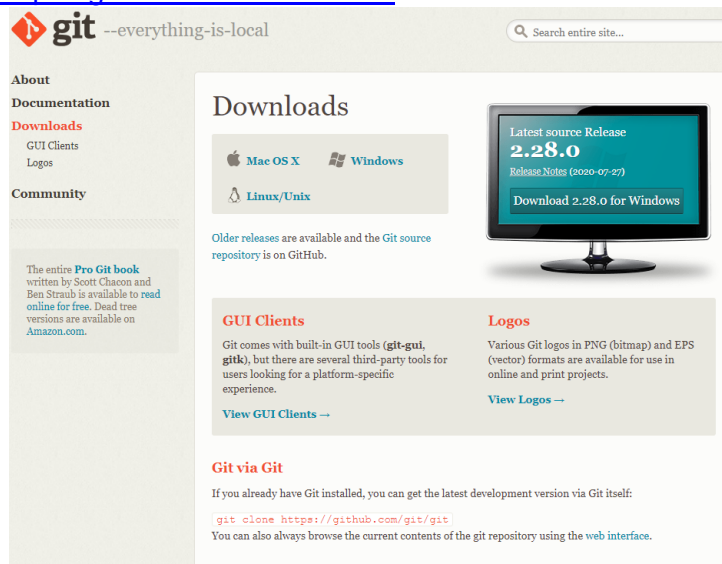
First Step: Install all the necessities

1. Install VS Code: <https://code.visualstudio.com/Download>

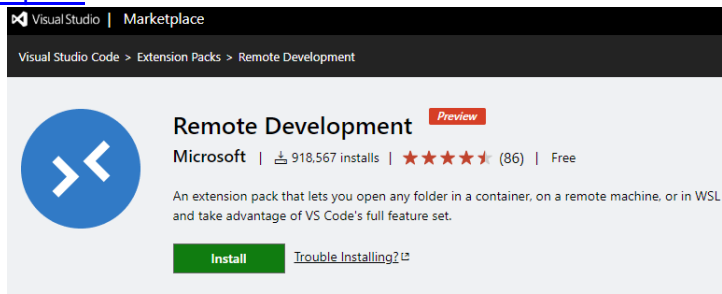
NOTE: You want to download **Visual Studio Code**, NOT Visual Studio



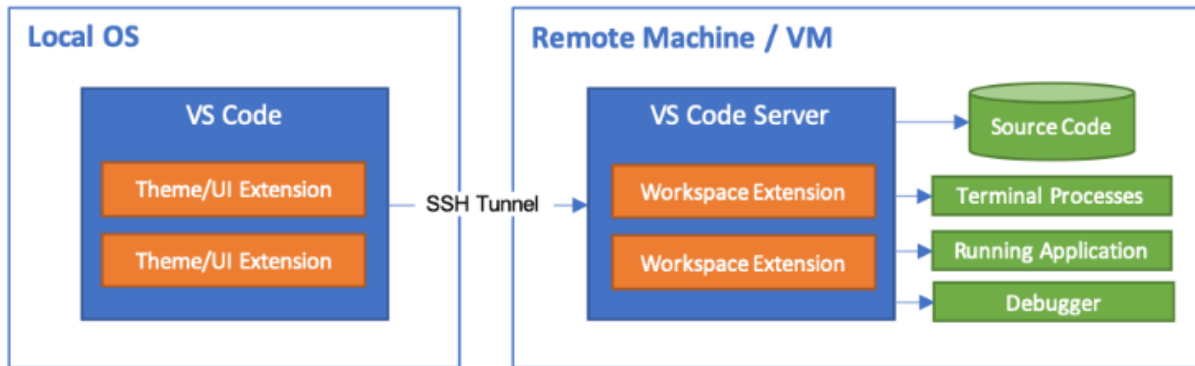
2. Install git bash: <https://git-scm.com/downloads>



3. Install Remote Development Pack:
<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack>



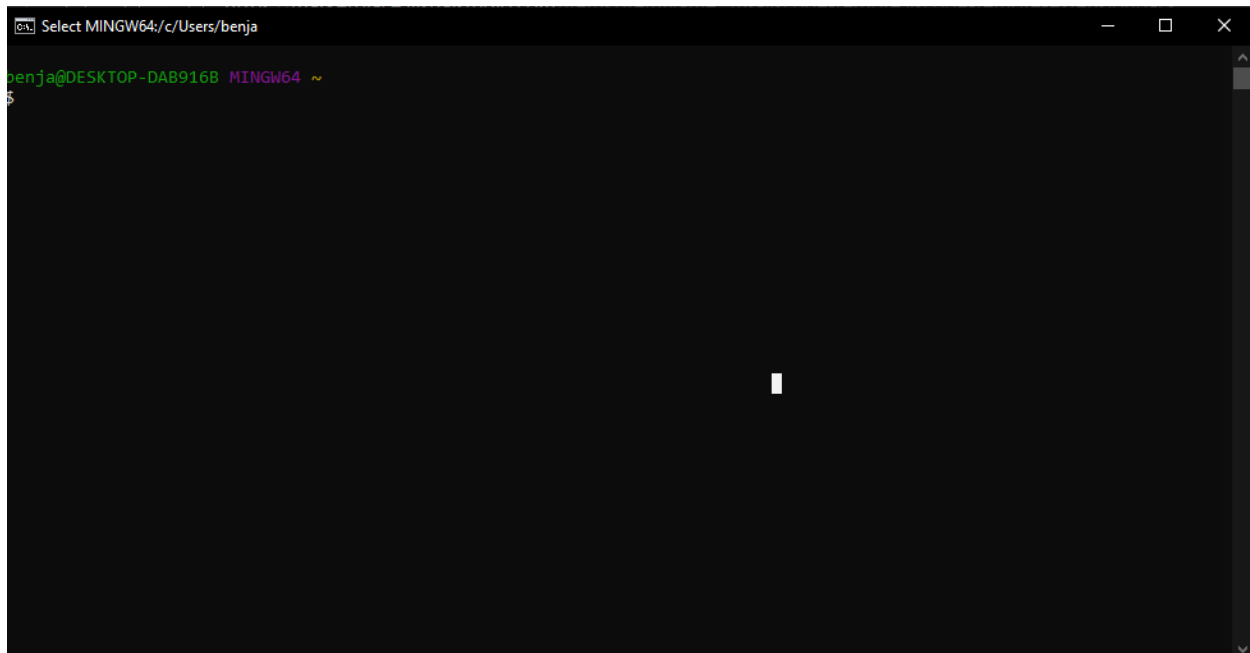
Second Step: Establish RSA SSH Connection (Encryption)



We must establish an SSH connection with the school server. This allows for high level encrypted security and speeds up the connection process without having to use a VPN and without having to enter a password/duo authenticate.

In order to do this, your computer needs two files to authenticate the connection: the **public** key which will be on the school server, and a **private** key which will be on the local machine (your computer).

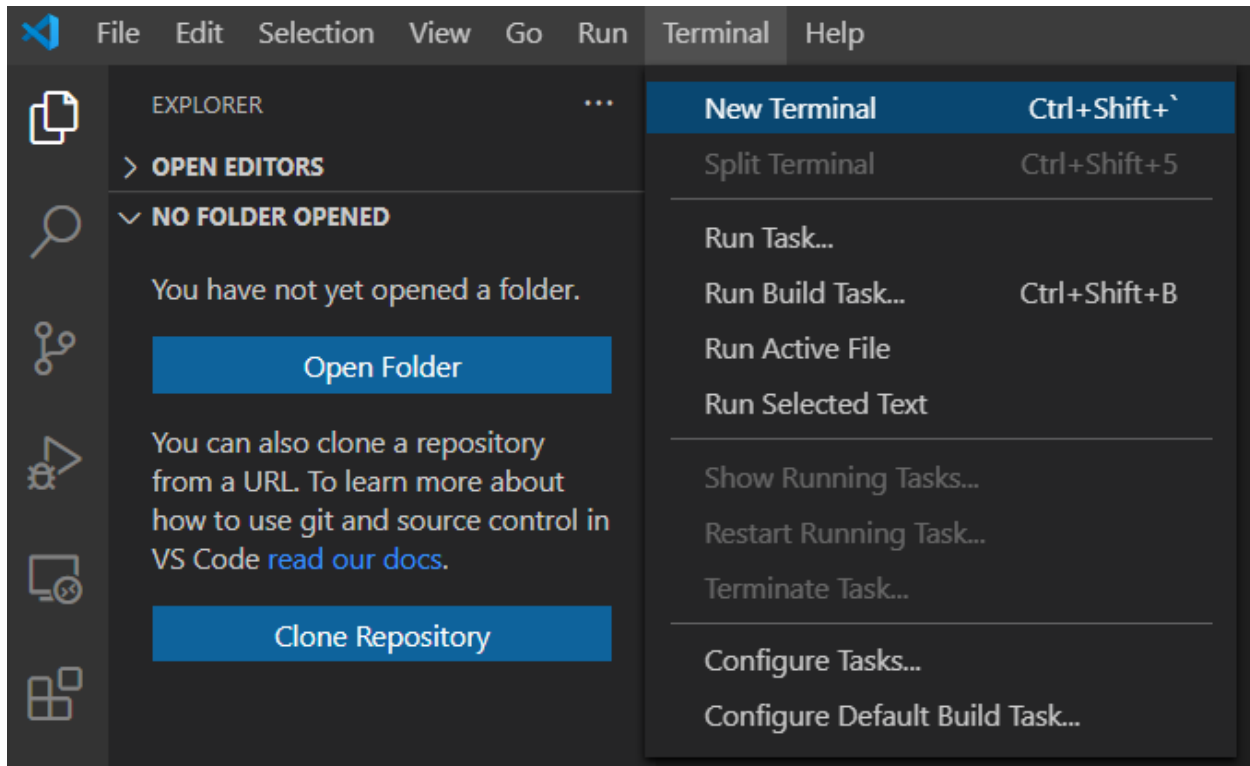
We need a supported OpenSSH compatible SSH client. The remote SSH server will be the school server. We will be using a **bash terminal**.



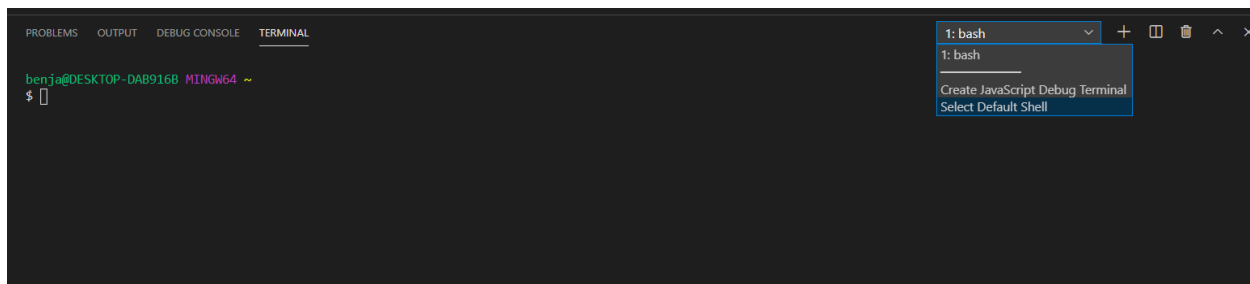
Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3

VS Code comes with an **integrated terminal**, meaning you can access the terminal inside VS Code. Because git bash is already installed, VS Code will find terminal through the installation path. You can open the bash terminal without VS Code (as shown the previous picture).

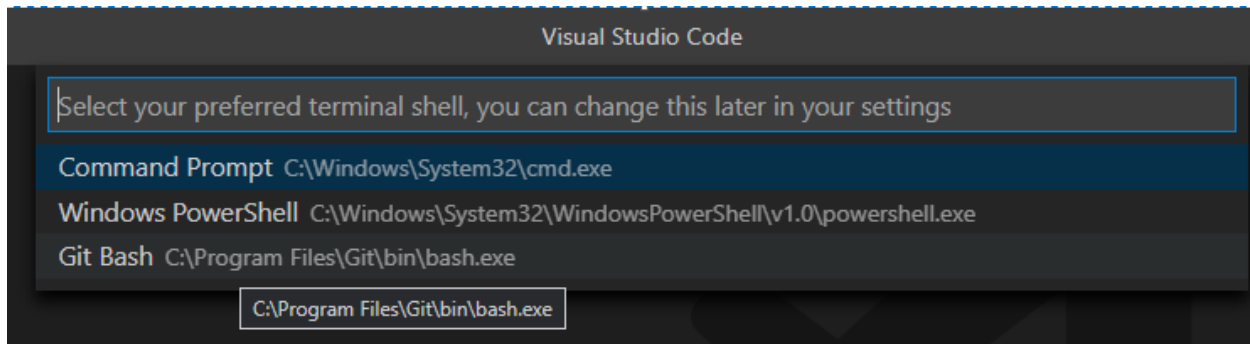
Click on the **terminal tab**, then select **New Terminal** (hotkey: Ctrl+Shift+`).



If it is not the bash terminal, click the terminal tab, then select **Select Default Shell**. Then select **Git Bash**.



Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3



Open new terminal. Ensure it is the bash terminal (shown above).

Now we need to create the ssh key files. If you do not have a key, run the following

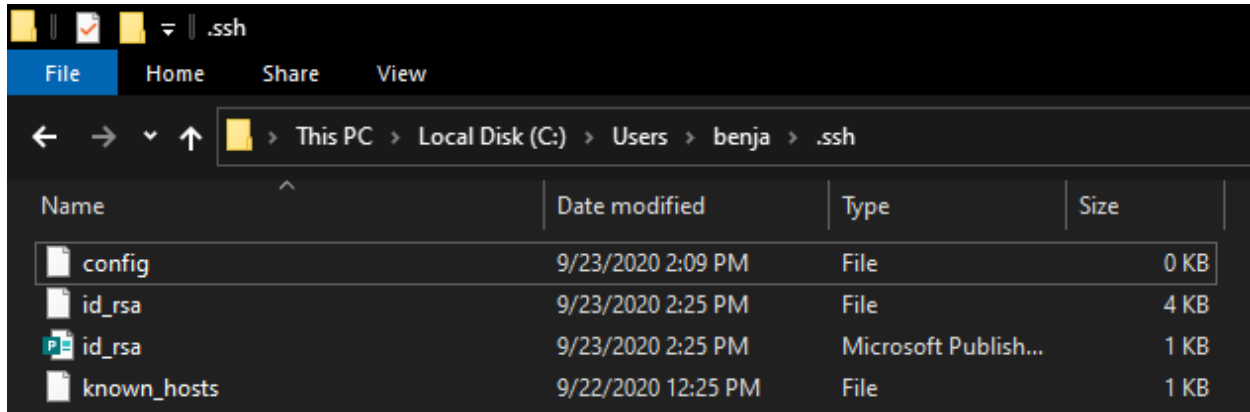
ssh-keygen -t rsa -b 4096

```
benja@DESKTOP-DAB916B MINGW64 ~  
$ ssh-keygen -t rsa -b 4096  
Generating public/private rsa key pair.  
Enter file in which to save the key (/c/Users/benja/.ssh/id_rsa):  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /c/Users/benja/.ssh/id_rsa  
Your public key has been saved in /c/Users/benja/.ssh/id_rsa.pub  
The key fingerprint is:  
SHA256:HQ7Ez20sp7xSZuPcxHjQB5IXRCDLJFpS5gvEBW+j1Rs benja@DESKTOP-DAB916B  
The key's randomart image is:  
+---[RSA 4096]-----+  
|o+B.o..==.          |  
|.B.+..oo.+          |  
|o .=oE o.*.         |  
|.+. . o .+0.        |  
|.. . SBo.           |  
|      B =           |  
|      B B           |  
|. = .               |  
|...                 |  
+---[SHA256]-----+  
  
benja@DESKTOP-DAB916B MINGW64 ~  
$
```

For the options, you can leave it blank (it defaults to what is in the parentheses).

Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3

This will make a private file and a public file. You can distinguish the public by the **.pub** extension. It should show in the terminal the path in which the files are saved.



Now we need the pub to be in our school server. This is done by adding the public key to the **school-user/.ssh/authorized_keys**. Luckily, there is an easy bash command to do this. In your bash terminal:

```
export USER_AT_HOST="your-school-username@access.engr.oregonstate.edu"
export PUBKEYPATH="$HOME/.ssh/file-name.pub"
ssh-copy-id -i "$PUBKEYPATH" "$USER_AT_HOST"
```

Example:

```
export USER_AT_HOST="condreab@access.engr.oregonstate.edu"
export PUBKEYPATH="$HOME/.ssh/id_rsa.pub"
ssh-copy-id -i "$PUBKEYPATH" "$USER_AT_HOST"
```

You do not want to copy the code above verbatim, but rather fill in the details of your school username and the ssh public file path. The **export** command is basically making a variable in the terminal scope. If you want to write the entire bash command in one line...

```
ssh-copy-id -i "$HOME/.ssh/file-name.pub" username@access.engr.oregonstate.edu
```

Example:

```
ssh-copy-id -i "$HOME/.ssh/id_rsa.pub" "condreab@access.engr.oregonstate.edu"
```

You will then be prompted with a password and DUO authentication. Simply enter 1 to run the DUO authentication. For security reasons, the password will not be displayed as you type it in.

Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3

```
benja@DESKTOP-DAB916B MINGW64 ~
$ ssh-copy-id -i "$PUBKEYPATH" "$USER_AT_HOST"
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/c/Users/benja/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
Password:
Duo two-factor login for condreab

Enter a passcode or select one of the following options:

1. Duo Push to XXX-XXX-8688

Passcode or option (1-1): 1

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'condreab@access.engr.oregonstate.edu'"
and check to make sure that only the key(s) you wanted were added.

benja@DESKTOP-DAB916B MINGW64 ~
$ ssh condreab@access.engr.oregonstate.edu
Last login: Tue Sep 22 22:34:32 2020 from 156.146.48.65
=====

Beginning Friday 12 July 2019 at 500pm, two-factor Duo authentication
will be required on the following College of Engineering linux SSH servers:

    access.engr.oregonstate.edu
    flip.engr.oregonstate.edu
    nome.eecs.oregonstate.edu

Students can sign up for two-factor Duo at:
    https://beav.es/duo

-----
If you have any problems with this machine, please mail support@engr.orst.edu
=====

[condreab@flip1 ~]$
```

[ssh-copy failed] If the above command in your bash terminal does not work, make sure your parentheses are included, your file path to your **public** key is correct, and your server username and login are correct.

[ssh-copy failed] If you are still not able to connect, then you can access the files directly by connecting to the server via ssh and editing/creating a directory and file for the ssh connection. Below are the step for new keys:

ssh username@access.engr.oregonstate.edu

Example:

ssh condreab@access.engr.oregonstate.edu

[ssh-copy failed] After we have put in the password and the DUO authentication, we need to create a folder **.ssh**, and a file inside called **authorized keys**. The data in your **public** file will go into this file. You can have multiple public keys in this one file (one each line). After having access to flip (not include parentheses):

Benjamin Condeara
Oregon State University – EECS TA
9/23/2020
v1.0.3

mkdir -p ~/.ssh (make the directory .ssh if it doesn't exist)
chmod 700 ~/.ssh (change the permissions of .ssh folder)
echo "your key here" >> ~/.ssh/authorized_keys (create and/or write key to file)
chmod 600 ~/.ssh/authorized_keys (change the permission of authorized keys file)

```
[condreab@flip1 ~]$ mkdir -p ~/.ssh
[condreab@flip1 ~]$ chmod 700 ~/.ssh
[condreab@flip1 ~]$ echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCMb/CpaTceAQQ4N9Vu8nkf/4zaYZi0+mDf8NzpFQ00h9Kes9swkoTxTCBmbROdFDWjAmlLoQjuoGqx7zZV06bamUcdMwuudQk6quvX47U
3L59hxfaxXncpmguLI97sXYEGs110MATPh804e0g5JgnMby//HhVC3F0S517ewDkPFgJ+OL0eDCKMbagS25LM0h1RXQzQF0h55mmzddpiQu978XYyuhuhwulnqo/LycxG8aHkQw1bSwQxPom7n6N2tc3iU3wG/bQEnehXeBuY1y
9yPKyeMddOf0aFgA/iZ9QXGuII/ASsnbaT4A5991FT51EG9aRVkYRipn3ZB59s8uQxh2e0Bj+14w+dKLfrCyM1NF1KQ8Guys21660G18wt6fzn6vFRkr1MQWzL14nEi3H/dliicdnLoxPt+0/yd919Xh11Nj/+H2j8Na3sYqY/
Rxd8uevFvHwWZE3Vj120kS9AX34rE0S8gmXX2/FLzKSul140LsMq4X3dy0MAyxtVcrQ8SXdhpxx8Fd9LDRet+YoNpRabGyS6bv5vUyrxovlvPrn9I9YEdTvcVlnT+EN7gYZQz3rSbijePfcaIABLUxp2TCBxz4LtQoIvd2
db81IxlR/D1o9QRiPEd5nKD1vNCdA578eHw1nuFcgpGsgc0vrr4fHveaBxArFv08MeT2gNzp13zw== benja@DESKTOP-DAB916B
> " >> ~/.ssh/authorized_keys
[condreab@flip1 ~]$ chmod 600 ~/.ssh/authorized_keys
[condreab@flip1 ~]$
```

Now you have successfully added the public ssh key on your school server. You should now be able to connect to the school server via ssh rsa encryption though the bash terminal (no need to put in password and DUO authentication).

```
benja@DESKTOP-DAB916B MINGW64 ~
$ ssh condreab@access.engr.oregonstate.edu
Last login: Wed Sep 23 21:08:20 2020 from 156.146.48.65
=====

Beginning Friday 12 July 2019 at 500pm, two-factor Duo authentication
will be required on the following College of Engineering linux SSH servers:

    access.engr.oregonstate.edu
    flip.engr.oregonstate.edu
    nome.eecs.oregonstate.edu

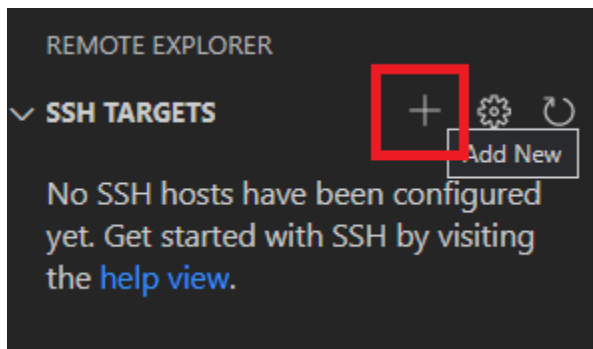
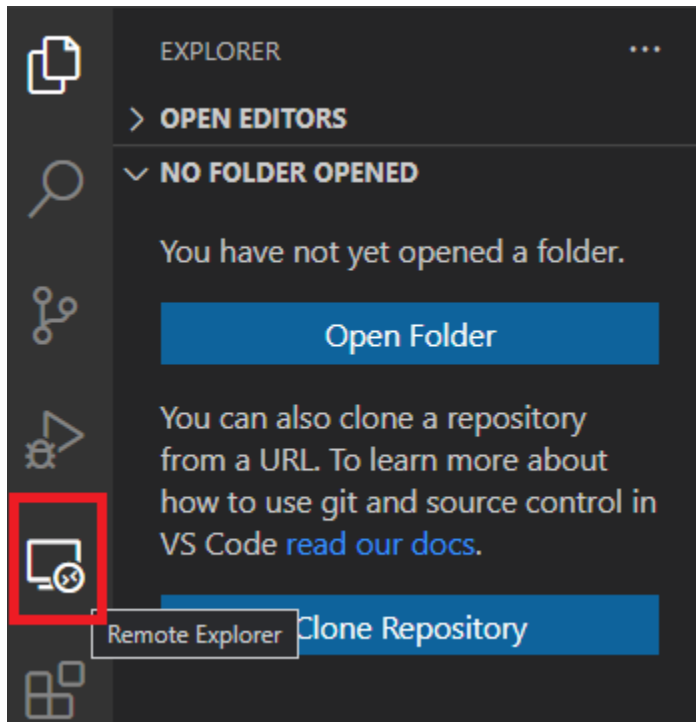
Students can sign up for two-factor Duo at:
    https://beav.es/duo

-----
If you have any problems with this machine, please mail support@engr.orst.edu
=====

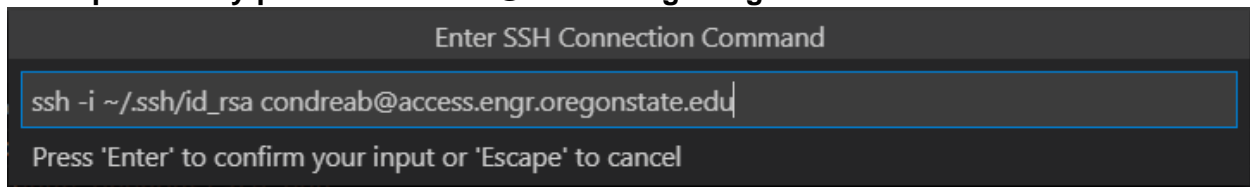
[condreab@flip2 ~]$
```


Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3

Now we need to set up a connection. Click on the **Remote Explorer** tab on the left. Enter in the **SSH Connection Command**:

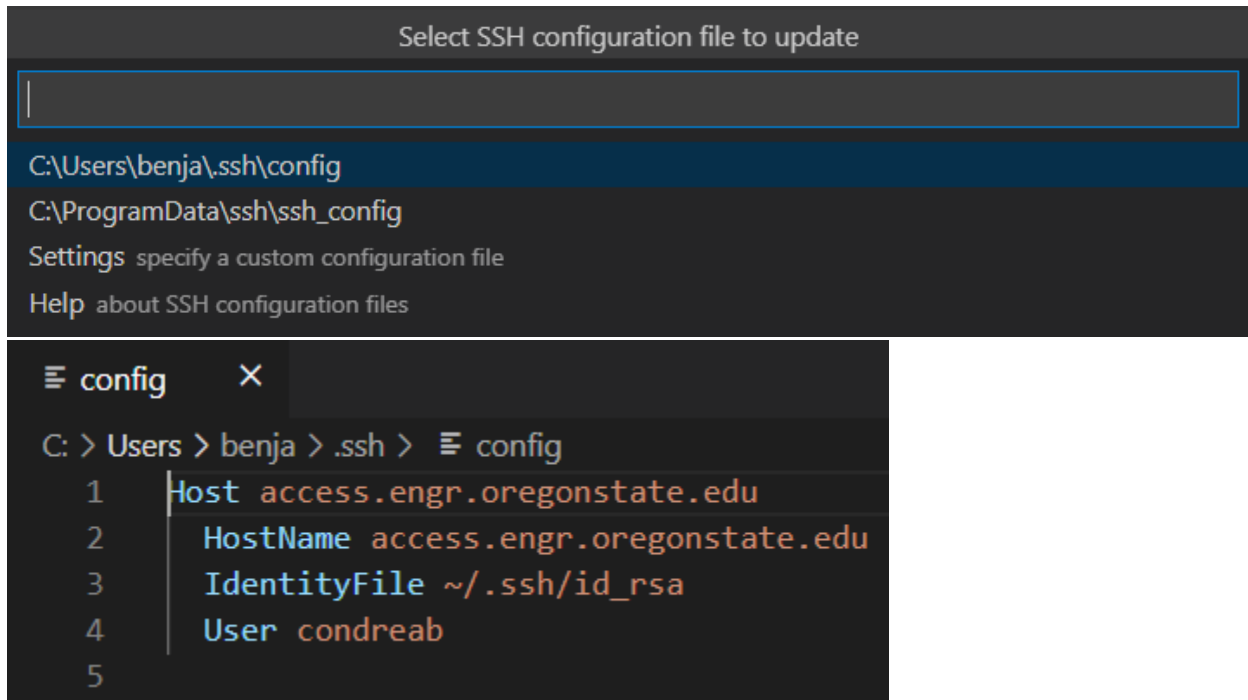


ssh -i private-key-path school-user@access.engr.oregonstate.edu

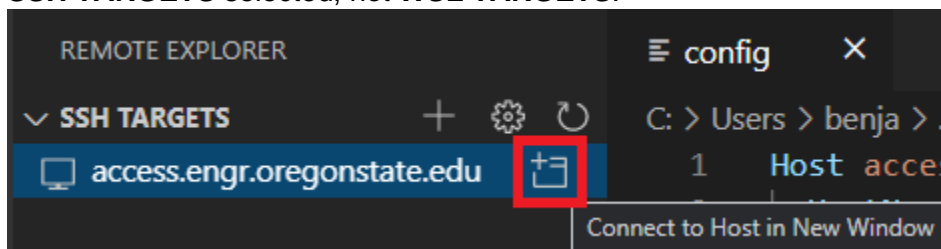


select the **SSH configuration file to update (usually the first one)**. Open the config file.

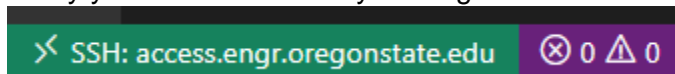
Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3



On the SSH TARGETS window, select **Connect to Host in New Window**. Make sure you have **SSH TARGETS** selected, not **WSL TARGETS**.



Verify you are connected by looking at the bottom.



Select **Linux** as the remote platform.

Create a file. This can be done by opening a new terminal. Notice that when you open a new terminal, you are now accessing the school **flip** server (instead of your local computer). To make a directory (folder), simply run this command. This is the **make directory** command for bash:

mkdir folder-name

Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3

```
[condreab@flip3 ~]$ mkdir testing123
[condreab@flip3 ~]$ cd testing123
[condreab@flip3 ~/testing123]$ mkdir homework
[condreab@flip3 ~/testing123]$ mkdir labs
[condreab@flip3 ~/testing123]$ mkdir recitation
[condreab@flip3 ~/testing123]$ pwd
/nfs/stak/users/condreab/testing123
[condreab@flip3 ~/testing123]$ ls
homework  labs  recitation
[condreab@flip3 ~/testing123]$
```

My suggestion is to do the following below (ignore the parentheses).

mkdir CS_162_F20

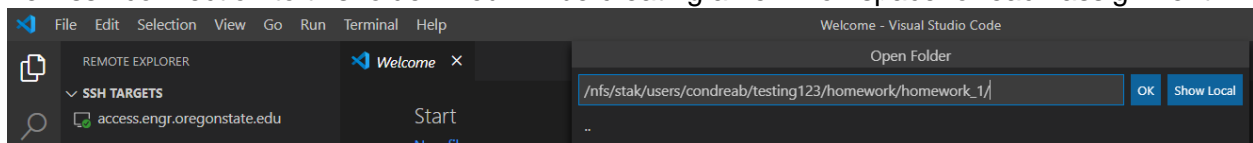
cd CS_162_F20 (change the current directory to the new folder)

mkdir assignments

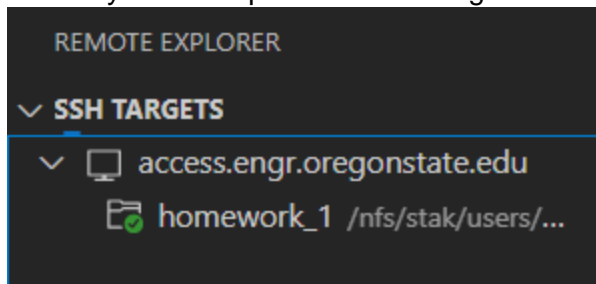
mkdir labs

mkdir recitations (probably not needed, but some students like to take notes)

Now select **Open folder**. This folder you open will become a **workspace folder**. It will create a new ssh connection to this folder. You will be creating a new workspace for each assignment.



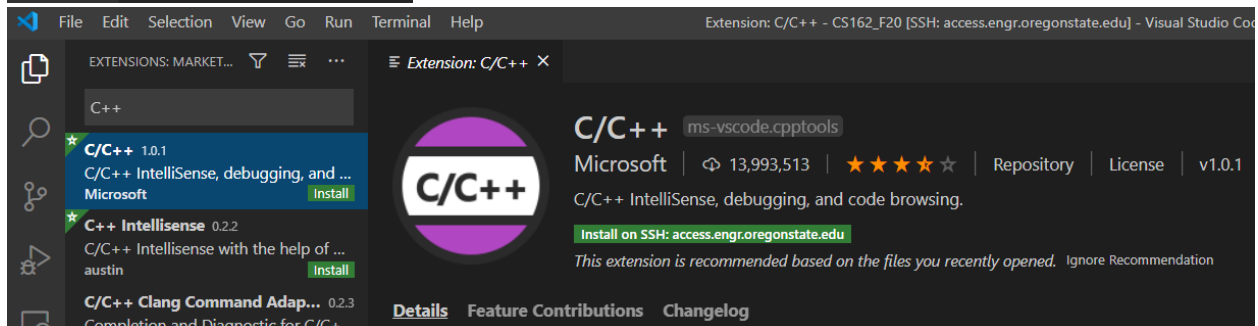
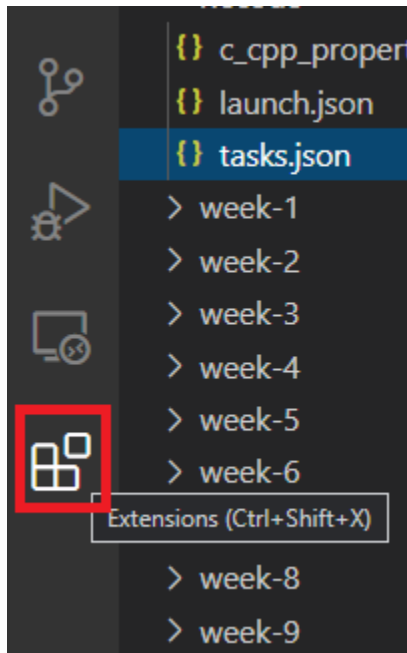
Notice now in the SSH TARGETS tab, you have the new folder you have connected to quickly access your workspace folder through the ssh connection.



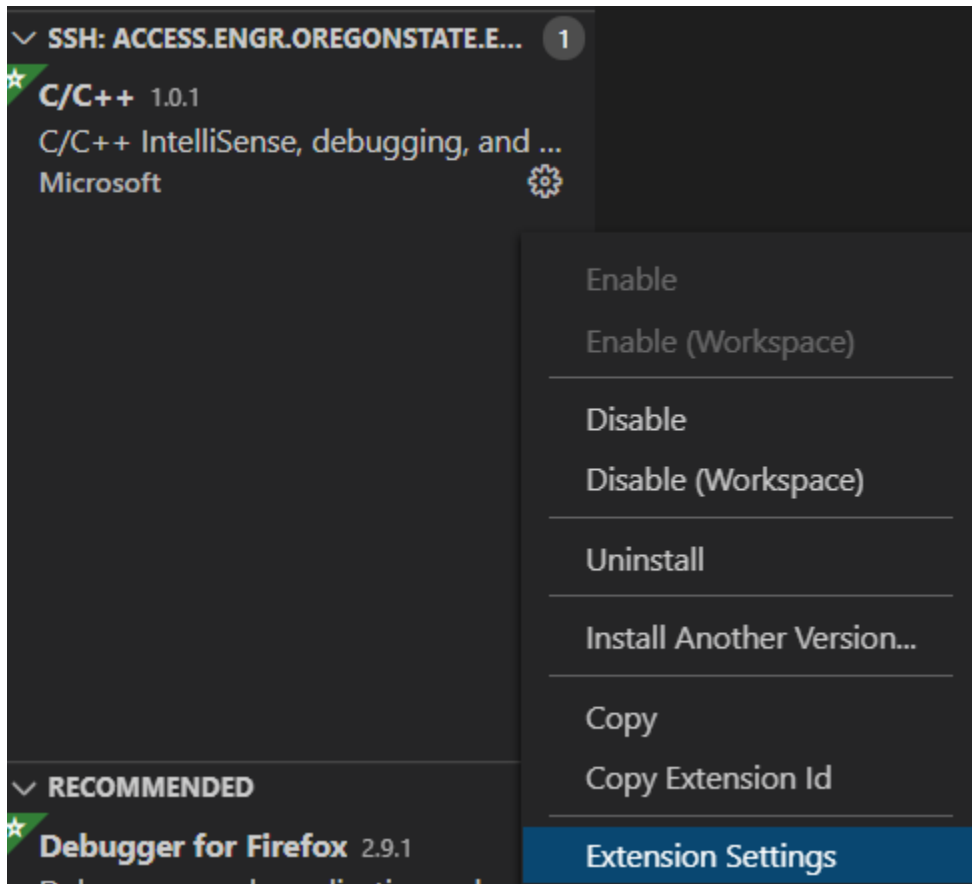
Third Step: Remote extensions/compiling/debugging

Now we need to install extensions. This can be done either locally on the UI / client side, or remotely on the SSH host. Most extensions will be on the SSH host to ensure a smooth experience for the workspaces. This allows you to pick up exactly where you left off *from a different machine*.

Install the C/C++ extension onto the school server.



You can edit the setting in the extension.



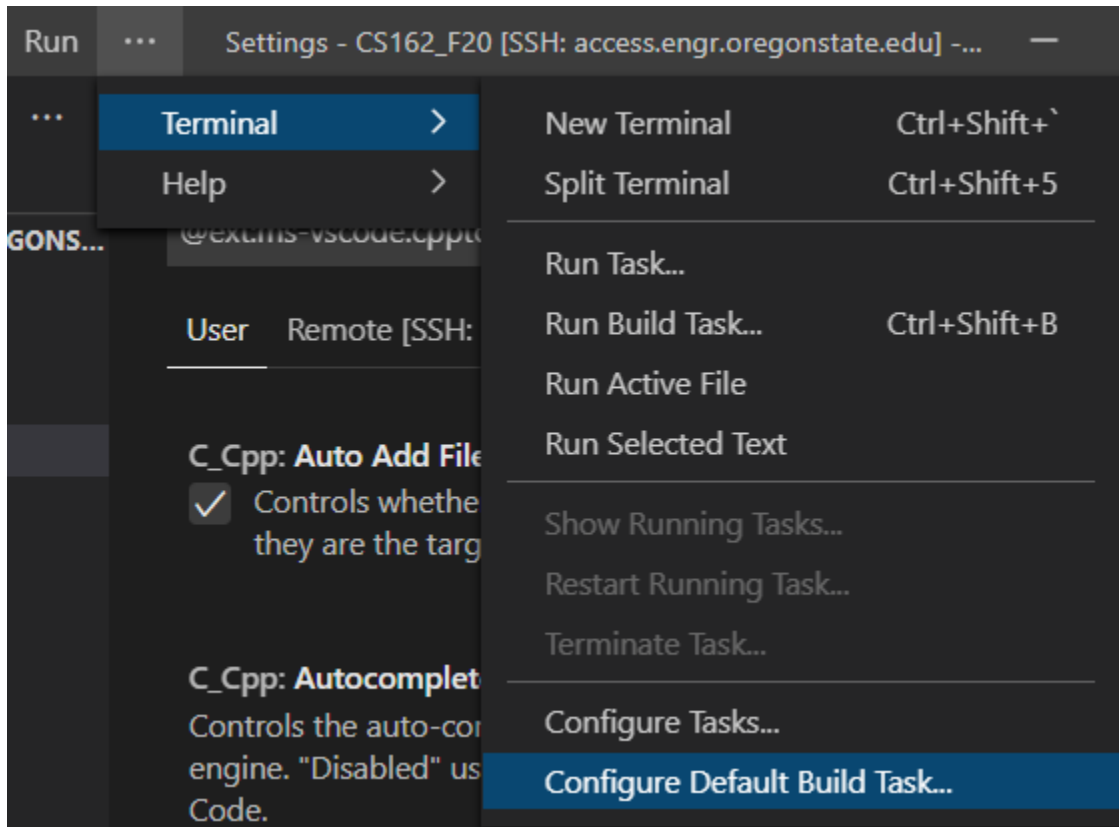
Compiling can be done in two ways: through the bash terminal or creating a task json file. A task file is a json file which allows you to quickly compile your workspace. A json file is a human readable way to transmit and store data objects with attributes. This way, we can store the settings for the launch.

The terminal command we want to run in a task file (you should be familiar with the command line below):

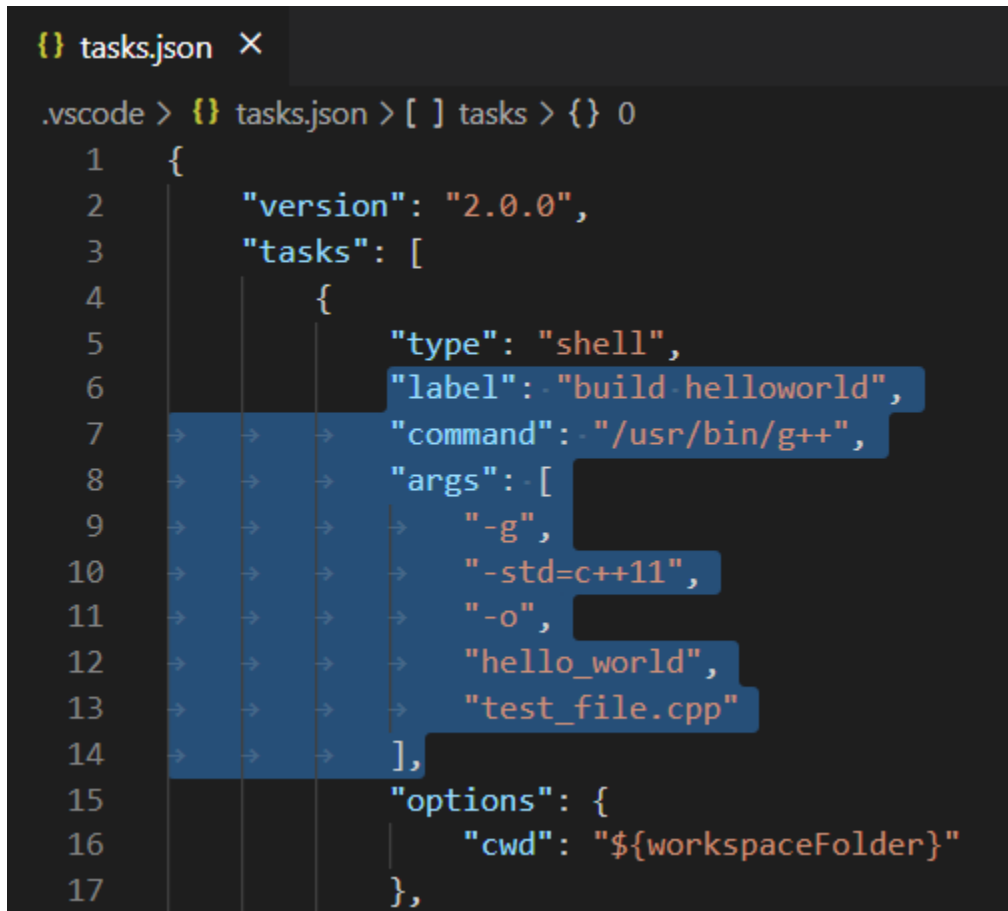
```
g++ -std=c++11 -g -o executable-file-name [source-files.cpp list]
```

The file we want to create is **tasks.json**

From the main menu, choose **Terminal** then select **Configure Default Build Task**. Choose **C/C++: g++ build active file**.



As you notice, the **command** is the location of the g++ compiler. The **args** array is the command-line arguments that will be passed to g++. The **label** can be anything you want. The **isDefault** specifies the task when **Ctrl+Shift+B** is run or **Tasks: Run Build Tasks** in the terminal menu.



```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "shell",
      "label": "build-helloworld",
      "command": "/usr/bin/g++",
      "args": [
        "-g",
        "-std=c++11",
        "-o",
        "hello_world",
        "test_file.cpp"
      ],
      "options": {
        "cwd": "${workspaceFolder}"
      }
    }
  ]
}
```

As before, you can run the program in your workspace directory by typing **./file-name** in your bash terminal or by running the default build with **ctrl+shift+B**. Below is a successful build.

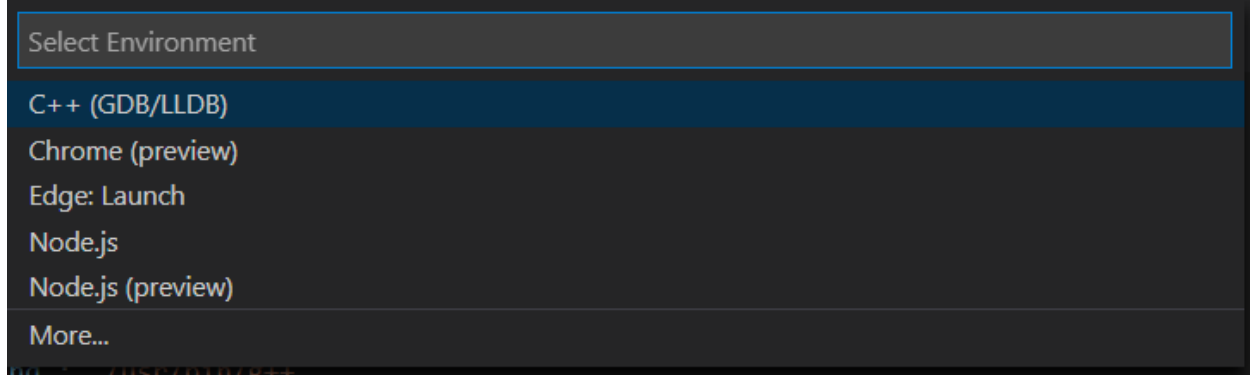
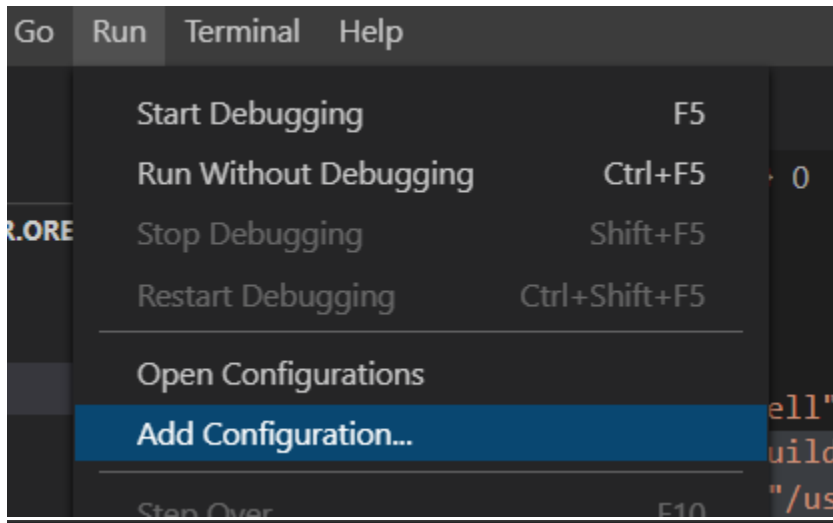


```
2: Task - build helloworl
The terminal process "/bin/bash '-c', '/usr/bin/g++ -g -std=c++11 -o test_file.cpp'" failed to launch (exit code: 4).
Terminal will be reused by tasks, press any key to close it.
> Executing task: /usr/bin/g++ -g -std=c++11 -o hello_world test_file.cpp
Terminal will be reused by tasks, press any key to close it.
> Executing task: /usr/bin/g++ -g -std=c++11 -o hello_world test_file.cpp
Terminal will be reused by tasks, press any key to close it.
```

Notice that it defaults to building the active file, meaning it will compile the file which is currently the active file. We need to change this if we have multiple files. This can be done by either hard-coding the filename or other methods like **"\${workspaceFolder}/*.cpp"** (compile all cpp files in the workspace folder).

Next, to debug the file, we need to create a **lauch.json** file. This file will launch the remote GDB debugger when you press **F5**.

From the main menu, choose **Run > Add Configuration...** and then choose **C++ (GDB/LLDB)**.



Change the **program** to the program executable.

```
"configurations": [  
  {  
    "name": "(gdb) Launch",  
    "type": "cppdbg",  
    "request": "launch",  
    "program": "${workspaceFolder}/hello_world",  
    "args": [],  
    "stopAtEntry": false,  
    "cwd": "${workspaceFolder}"
```

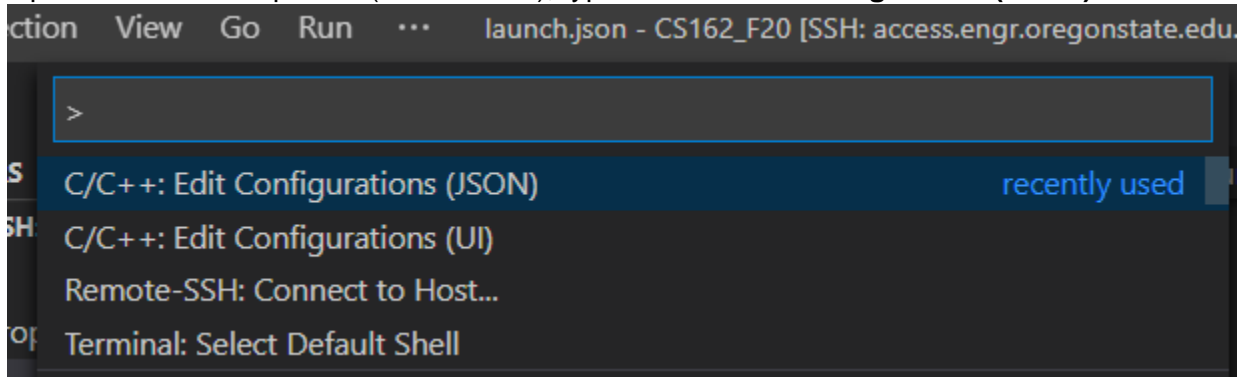
OPTIONAL: Add **prelaunchTask** as the **label** of the task in the tasks.json file (to compile before debugging).

OPTIONAL: Add **miDebuggerPath** to the path of the machine interface debugger (to specify the debugger).

Benjamin Condrea
Oregon State University – EECS TA
9/23/2020
v1.0.3

```
        "text": "-enable-pretty-printing",  
        "ignoreFailures": true  
    },  
    ],  
    "preLaunchTask": "g++ build active file",  
    "miDebuggerPath": "/usr/bin/gdb"  
}
```

Open the command palette (Ctrl+Shift+P), type **C/C++: Edit Configuration (JSON)**



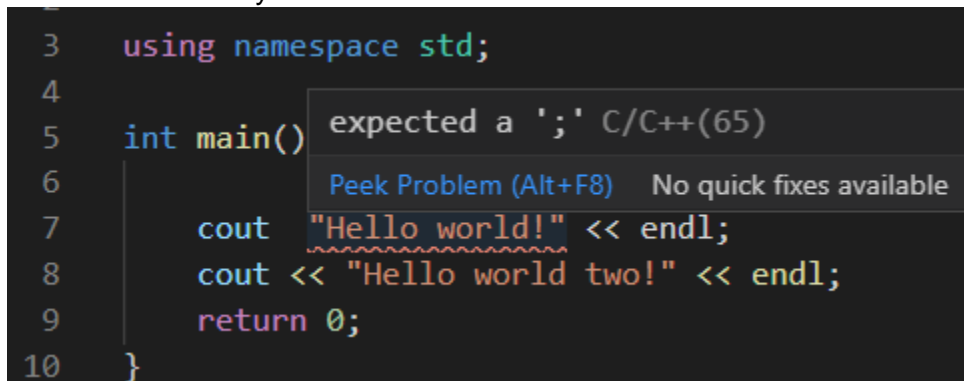
Change the compiler path to **g++** and change the cpp standard to **c++11**. Add **IntellisenseMode** to **gcc-x64** (the school uses older versions). You want to set it to the school's gcc compiler because you want the corrections to correlate with the school's compiler.



The screenshot shows the VS Code interface with the `c_cpp_properties.json` file open. The file is part of a workspace containing `tasks.json` and `launch.json`. The configuration is for a C/C++ environment on Linux. The `configurations` array contains one configuration named "Linux". The `compilerPath` is set to `"/usr/bin/g++"`, `cStandard` is `"c99"`, `cppStandard` is `"c++11"`, and `intelliSenseMode` is `"gcc-x64"`. The `version` is set to 4.

```
.vscode > {} c_cpp_properties.json > [ ] configurations > {} 0
1  {
2      "configurations": [
3          {
4              "name": "Linux",
5              "includePath": [
6                  "${workspaceFolder}/**"
7              ],
8              "defines": [],
9              "compilerPath": "/usr/bin/g++",
10             "cStandard": "c99",
11             "cppStandard": "c++11",
12             "intelliSenseMode": "gcc-x64"
13         }
14     ],
15     "version": 4
16 }
```

Intellisense is VS Code's auto correct. It formats your code to your likings through the C/C++ extension and many more features.

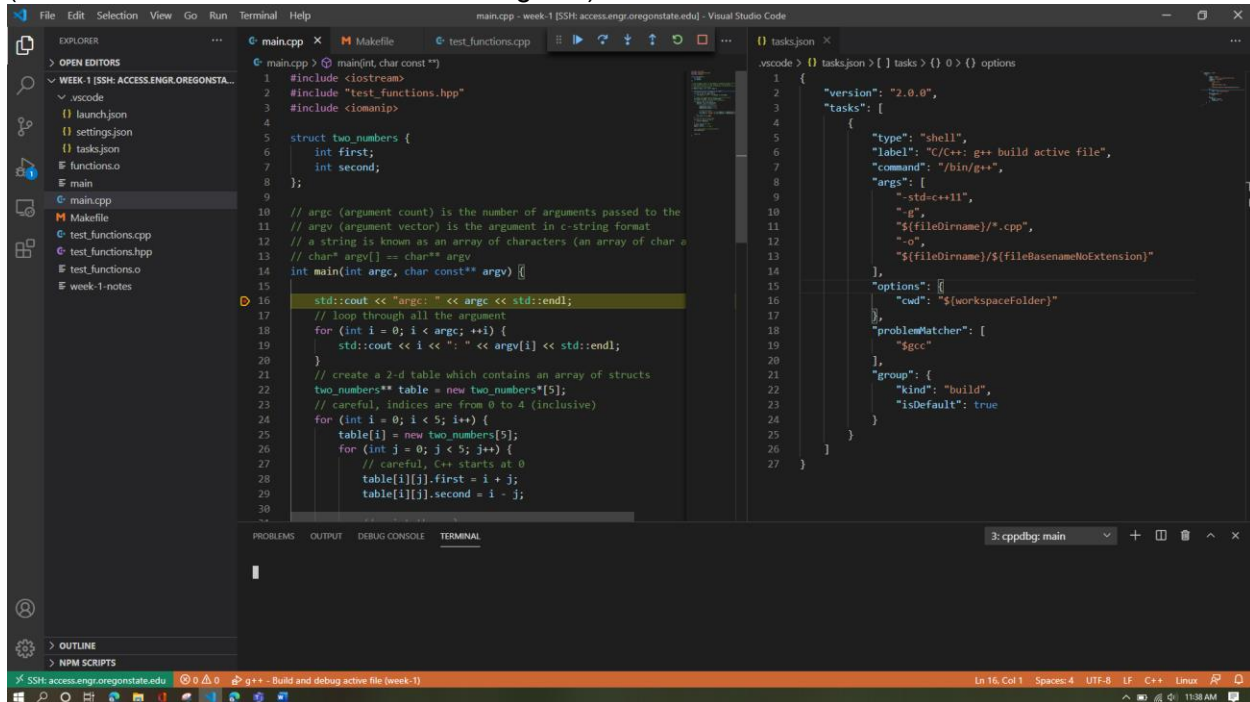


The screenshot shows a C++ code snippet in VS Code. The code is: `using namespace std;`, `int main()`, `cout << "Hello world!" << endl;`, `cout << "Hello world two!" << endl;`, and `return 0;`. An Intellisense error message is displayed over the first `cout` line, stating "expected a ';' C/C++(65)". The message also includes a "Peek Problem (Alt+F8)" button and a note that "No quick fixes available".

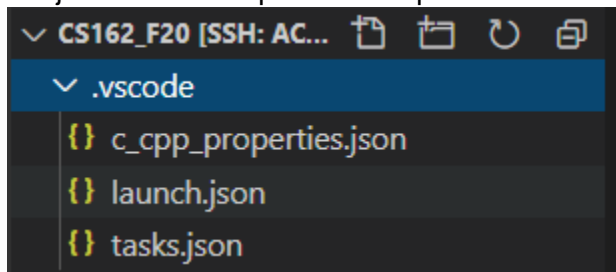
```
3  using namespace std;
4
5  int main()
6      cout << "Hello world!" << endl;
7      cout << "Hello world two!" << endl;
8      return 0;
9
10 }
```

Benjamin Condeara
Oregon State University – EECS TA
9/23/2020
v1.0.3

Use this guide to start a debugging session: <https://code.visualstudio.com/docs/cpp/config-linux>
(I do not wish to reiterate a well written guide).



Each workspace folder should have the **.vs-code** folder automatically created when you created the json files in the previous steps.



Changelog

9/23/2020 – Release
9/24/2020 – Added Intellisense description, fixed some grammatical errors.
9/25/2020 – Added more explicit examples for command lines
9/28/2020 – Added more gdb debugging.
9/30/2020 – Fixed some incorrect commands.