Linux & Testing on SEASnet Server and Debugging

Wenjie Mo
OH: Monday 12:30-1:30PM
Email: martinmo@ucla.edu

Yichen Zhou
OH: Tuesday 12:30-1:30PM
Email: yichenzhou@g.ucla.edu

Follow this presentation at
https://tinyurl.com/CS32W22LAWorkshop1

Overview

1. SEASnet and UCLA VPN Setup
2. Linux Server Login
3. Linux Server Commands
4. Move files over to Linux server
5. Debuggers (XCode and VS)

SEASnet & VPN Setup

What is UCLA’s SEASnet Linux Server?

- Linux is an operating system, kind of like Windows or Mac
- The Linux server is a group of machines that will be used to test your code
- You can log into the Linux server to test your code
- In CS 31 and 32, it’s especially important to test your code on the server because it can catch errors that Xcode/ Visual Studio might not catch

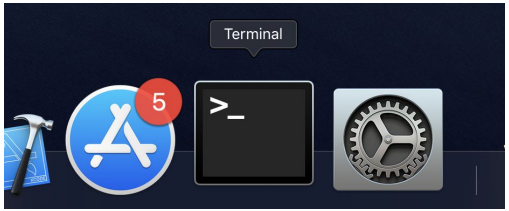
Accessing Linux Server

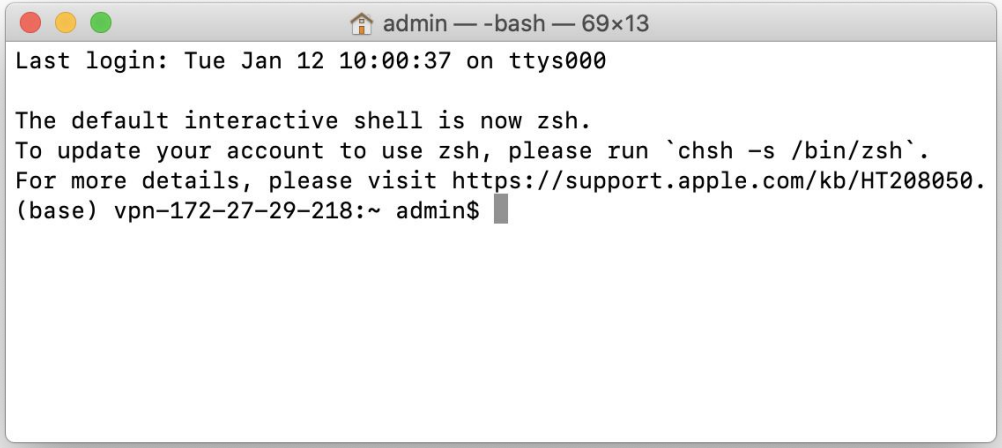
Accessing SEASnet Linux Server

- Create an account if you don't have one already, it will usually take 24 hours/ 1 business day for it to generate, so make an account ASAP!
  - https://seas.ucla.edu/acctapp/
- Access in Windows:
  - PuTTY (recommended)/ Windows Terminal with WSL2 (recommended) or Command Prompt
  - Remote Desktop Connection
- Access in Mac:
  - Terminal
- If connecting from a computer off campus, you need to connect to the campus VPN here:
  - https://www.it.ucla.edu/bol/services/virtual-private-network-vpn-clients
- If connecting on campus, connect to either one of these secure WiFis:
  - eduroam
  - UCLA_WIFI
  - UCLA_WIFI_RES
  - UCLA_SECURE_RES

Logging Into the Linux Server (Mac)

- Launch the Terminal Application





Logging Into the Linux Server (Mac)

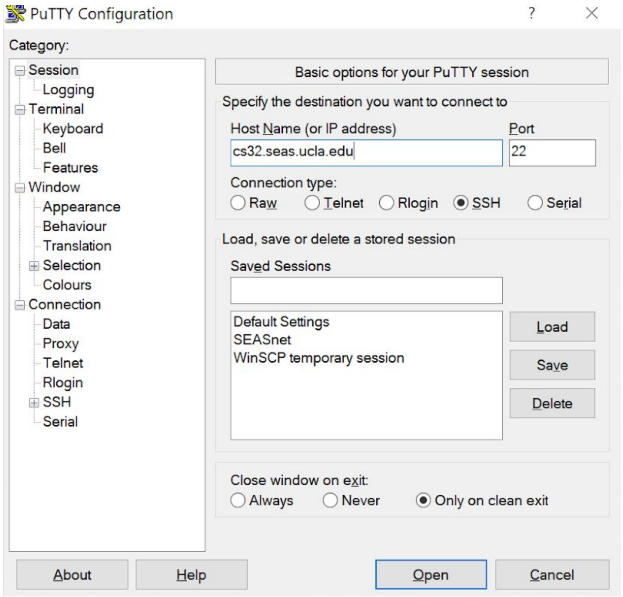
admin — ssh classjzh@cs32.seas.ucla.edu — 81×37

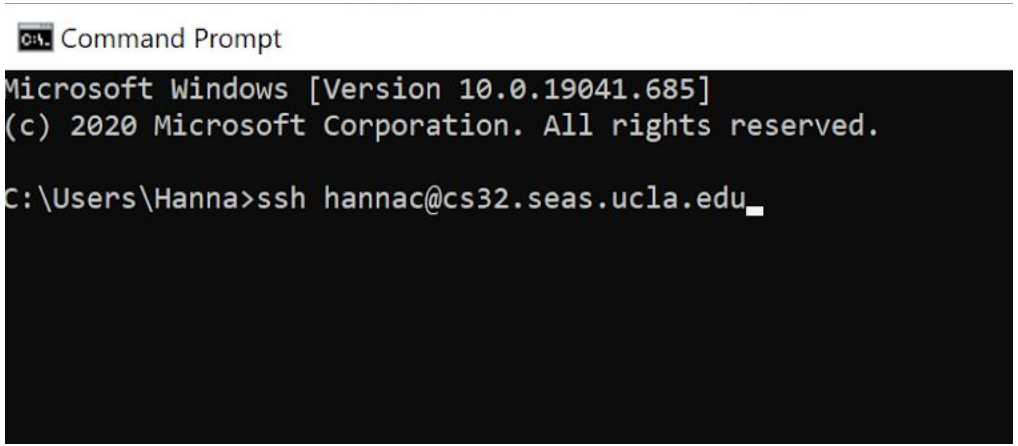
```
Last login: Tue Jan 12 09:43:00 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) vpn-172-27-154-54:~ admin$ ssh classjzh@cs32.seas.ucla.edu
The authenticity of host 'cs32.seas.ucla.edu (164.67.100.236)' can't be establish
ed.
ECDSA key fingerprint is SHA256:kk4Uglj/TTk9Be7q7acNGSjdV1Sk+bsymPopyskF9H4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'cs32.seas.ucla.edu' (ECDSA) to the list of known host
s.

Last login: Tue Nov 10 02:41:46 2020 from 172.27.141.227
*******************************************************************************
                        lnxsrv06.seas.ucla.edu RHEL 7
*******************************************************************************

        * User processes older than 36 hours will be cleaned up

*******************************************************************************
*******************************************************************************
* SEASnet Computing Access                                                    *
*                                                                             *
* Priority is given both on the servers and in the student labs to those     *
* students doing coursework. Computing support for research is provided by   *
* each department.                                                            *
*******************************************************************************
* For assistance please contact help@seas.ucla.edu or call 206-6864.          *
*******************************************************************************

[classjzh@lnxsrv06 ~]$
[classjzh@lnxsrv06 ~]$
```

- Type the command:
  - `ssh yourSEASaccount@cs32.seas.ucla.edu`
- Type ‘yes’ when prompted

Logging Into the Linux Server (Windows)

- Open putty.exe and enter the hostname “cs32.seas.ucla.edu”
- Open Command Prompt, or Windows Terminal (if installed) and follow the ssh instructions in the last slide





Important Linux Commands

Important Commands

We will not talk about Linux commands in detail in this workshop. See the main site of CS32 http://web.cs.ucla.edu/classes/winter22/cs32/linux.html for more useful commands.

- `ls` list files in a directory (default: the directory you're on)
- `mkdir` make a directory
- `mv` Move files from one directory to another
- `cd` change directory to move in/ out of a directory
- `cp` copy files (either to another filename or the same one)
- `rm` remove files/ directory.

Please note removed/moved files **CANNOT** be recovered or undoed, so use the commands carefully!!

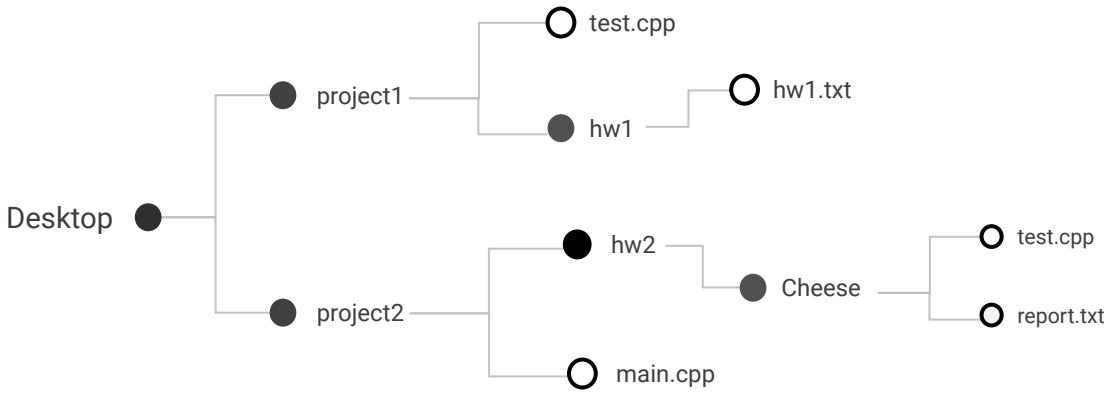
Important Commands

command [-flag] [arguments]

- ls
  - list files in a directory (default: the directory you're on)
  - `ls [directory]` / `ls`
    - `ls Desktop` – list all files in Desktop
    - `ls` – list all files in your current directory
- mkdir
  - make a directory
  - `mkdir [directory_name]`
    - `mkdir cs32_project1`
- mv
  - Move files from one directory to another
  - `mv [filename] [directory]`
    - `mv zombie.cpp cs32_project1`
- cd
  - change directory to move in/ out of a directory
  - `cd [directory]`
    - `cd Desktop`
- cp
  - copy files (either to another filename or the same one)
  - `cp [filename]`
  - `cp [filename] [new_filename]`
    - `cp zombie.cpp zombie2.cpp`
  - `cp [filename] [directory]`
    - `cp zombie.cpp cs32_project1`
- rm
  - remove files/ directory
  - rm [filename]
    - `rm zombie.cpp`
  - rm -r [directory]
    - `rm -r cs32_project1`

Important Commands: `cd`

- `cd` is used to change the directory you're in (change current directory)
- To move out one directory: `cd ..`
- To move out >1 directories: `cd ../../..` (add as many /.. as you need)
- To go back to your home directory: `cd`



Some things to think about:
- Which ones are directories and which ones are files?
- If we're in the project2 directory and we do 'ls', what's the output?
- How to find hw1.txt from Desktop
- How to get back to Desktop from hw1 directory
- If we're currently at the Cheese directory, where would we be if we do the command ../../..

Now, try it yourself!!

1. Log into your Linux server (what's the command?)
2. Make a new directory called `linux_practice`
   - `mkdir linux_practice`
3. Pick two files that you already have (unused ones preferably), make a copy of it and move it into the `linux_practice` directory
   - `cp [file1] [file2] linux_practice`
4. Move to the `linux_practice` directory
   - `cd linux_practice`
5. Make a copy of each file, renaming them to `one` and `two`, respectively
   - `cp [file1] one`
   - `cp [file2] two`
6. Remove the two old files
   - `rm [file1] [file2]`
7. Go back one directory (the previous directory you were on)
   - `cd ..`
8. Remove the whole `linux_practice` directory
   - `rm -r linux_practice`

Moving Files to Linux Server

Move Files Over to Linux Server (Mac OS & Windows)

(If connecting from a computer off campus, you need to connect to the campus VPN first)

Download [FileZilla]

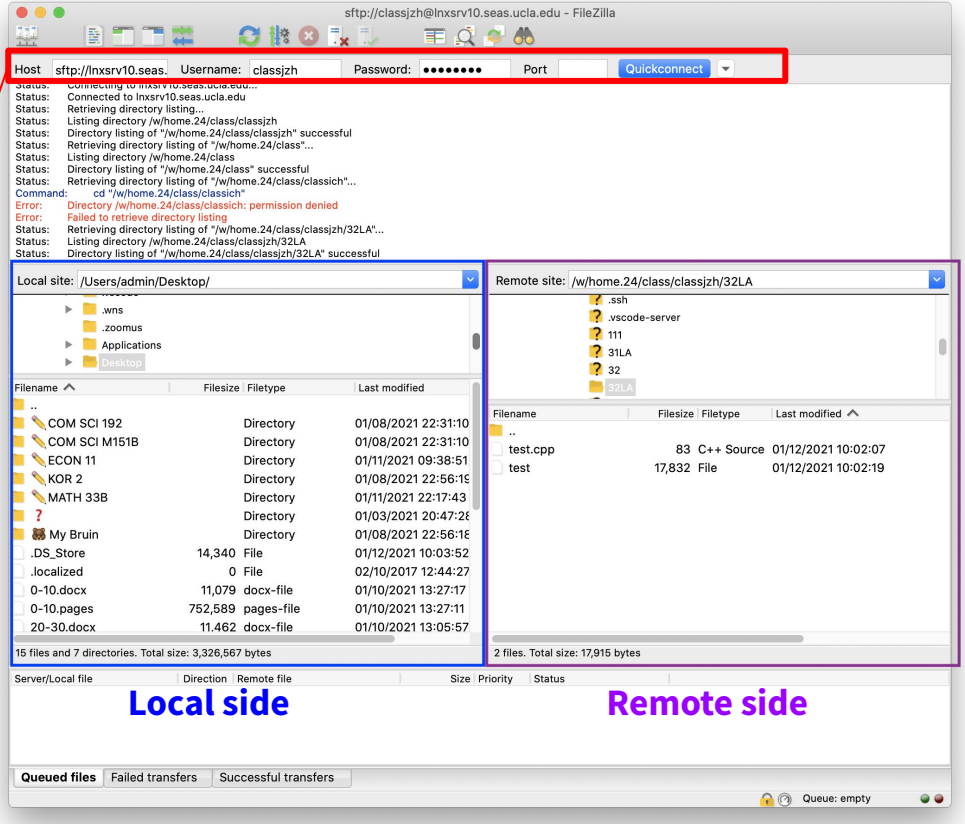
Host: cs32.seas.ucla.edu

Username: Your username

Password: Your password

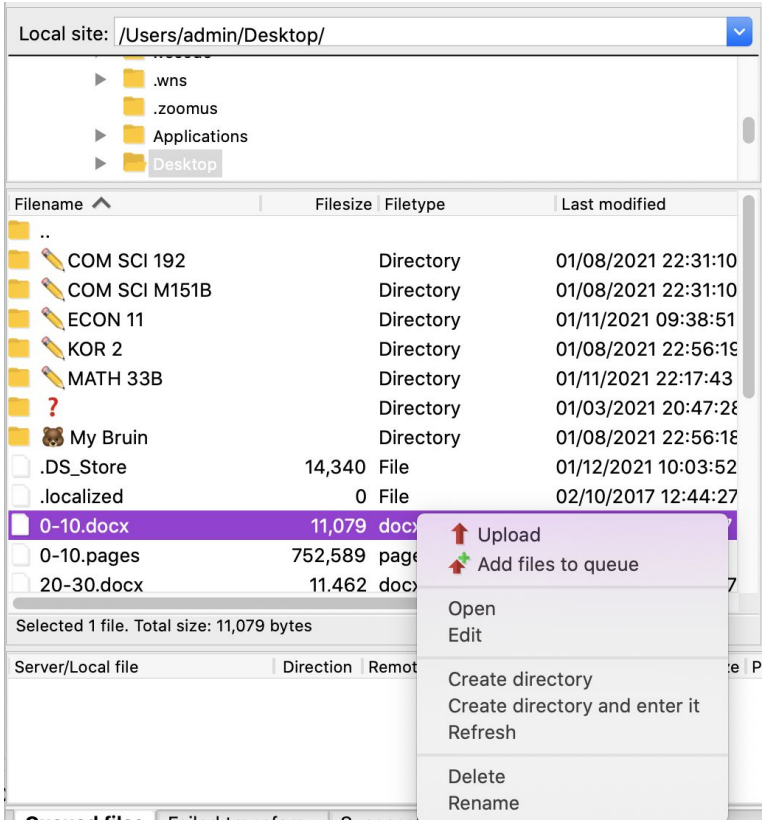
Port: 22

Click Quickconnect

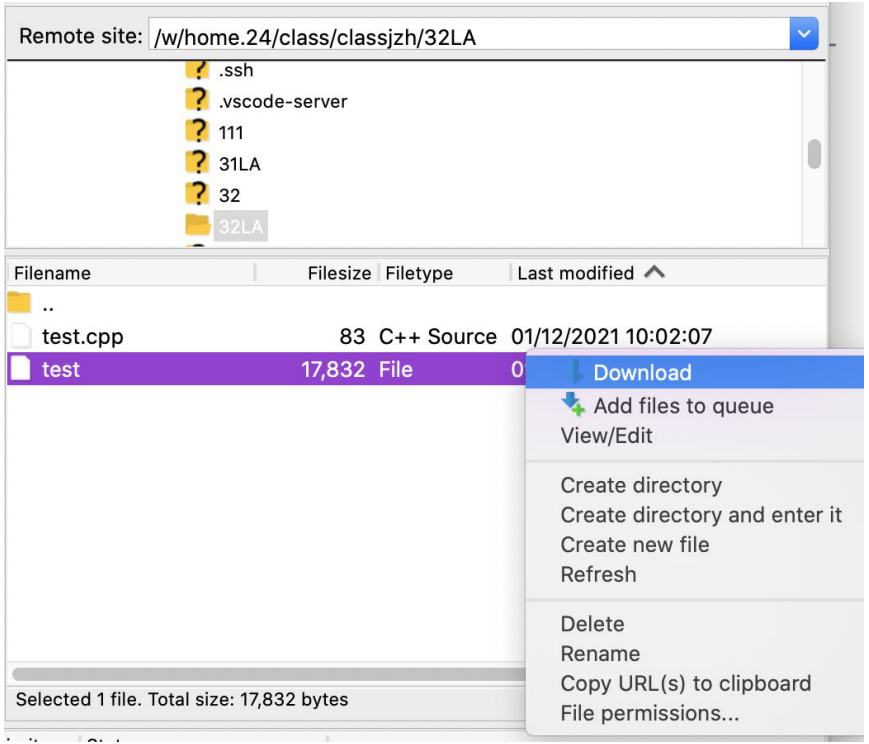


Move Files Over to Linux Server (Mac OS & Windows)

Local side



Remote side



Move Files Over to Linux Server (Mac OS & Windows)

(If connecting from a computer off campus, you need to connect to the campus VPN first)

Download CyberDuck

Protocol: SFTP

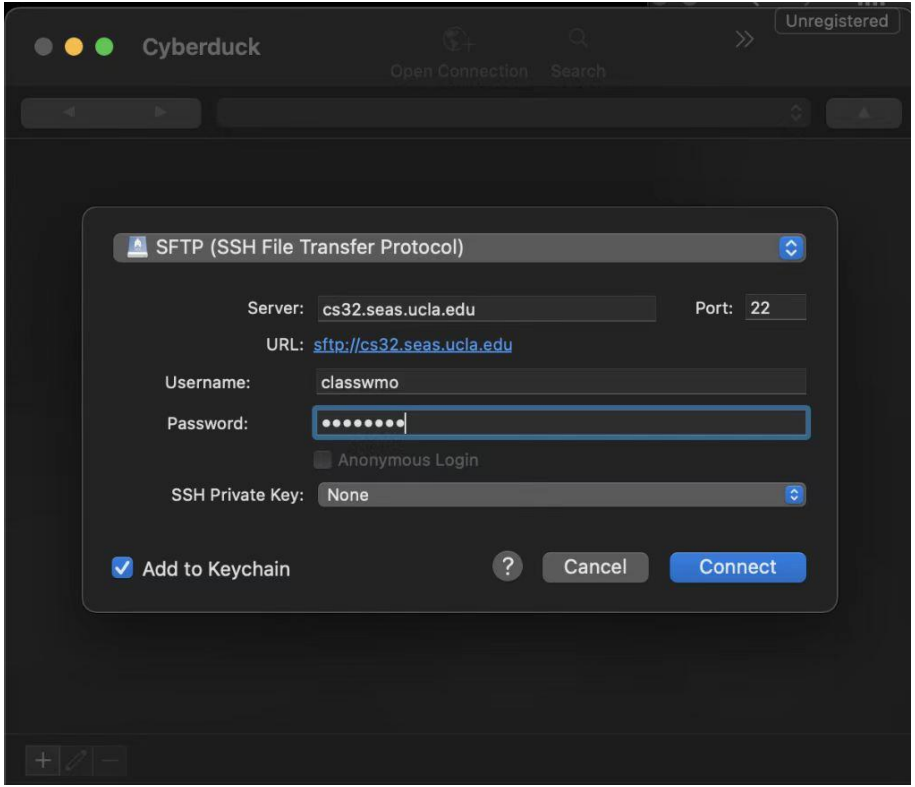
Server: cs32.seas.ucla.edu

Username: Your username

Password: Your password

Port: 22

Click Connect



Move Files Over to Linux Server (Windows)

(If connecting from a computer off campus, you need to connect to the campus VPN first)

Download WinSCP

Host: cs32.seas.ucla.edu

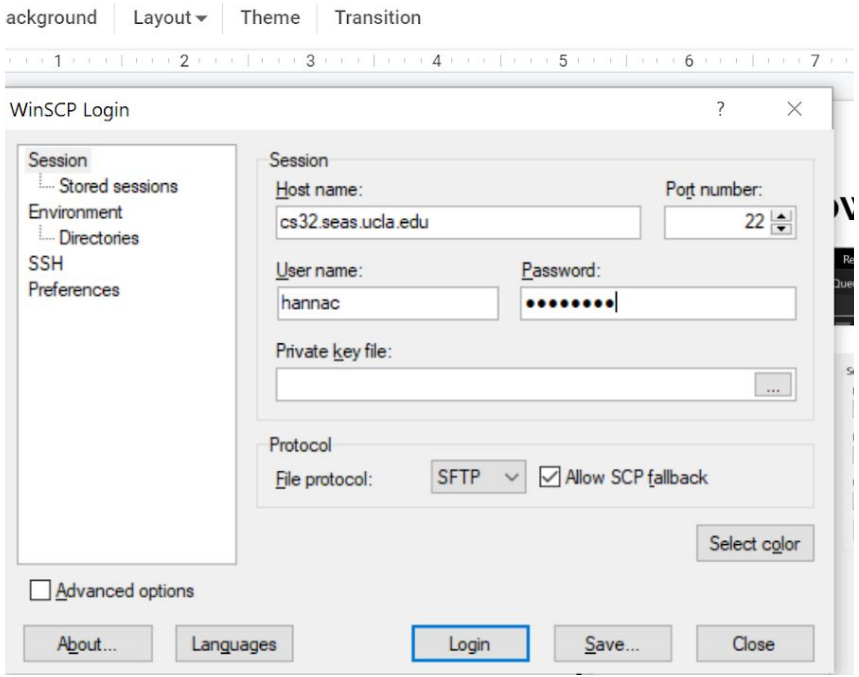
Port number: 22

Username: Your username

Password: Your password

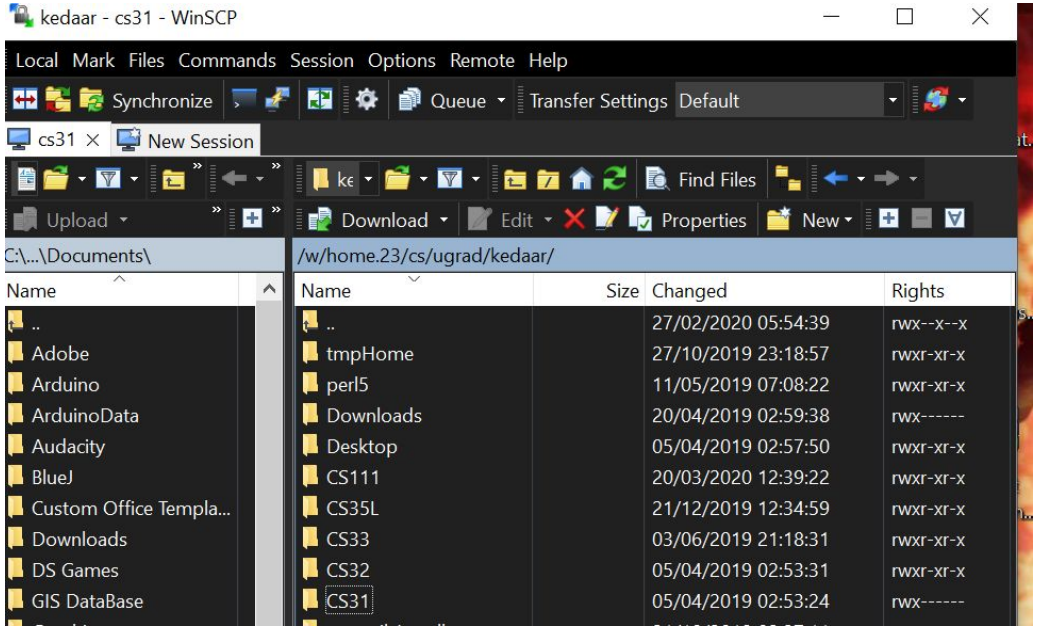
File Protocol: SFTP

Click Login/Save



Move Files Over to Linux Server (Windows)

- Can drag and drop your .cpp or project files from your computer into the Linux server (shown below)
- Can then login into PUTTY to SSH into the server and see the copied files



Move Files From Linux Server to Local (Windows)

- Moving files from Linux server to local system is the same as moving from local to the Linux server
- Just use the WinSCP/CyberDuck interface to copy files from Linux server to local system
  - Drag and drop or copy and paste

Move Files Over to Linux Server

- **Quick recap**
  - Connect to VPN if you are off-campus
  - Use file transfer applications (FileZilla, WinSCP, CyberDuck, etc.)
  - Can also use scp to transfer files (see more on the main site of CS32, Linux man page)
- **Next steps**
  - Log into SEASnet Linux server via SSH
  - Compile the program and test it! (Still need to use command line tools)

Debugging

See the complete demo at https://tinyurl.com/debuggingdemo

Why do we need a debugger?

We could just stare at the code, but this might take a long time (or possibly forever). We could also use print statements. But this lacks many of the useful features of a debugger, which include:

- Stepping through the code **line by line**
- **Stopping** the program at particular points called breakpoints
- **Inspecting the value** of variables at those points

Breakpoints

Stop the program at places where you might think there is a problem. Click next to the line number to set a breakpoint at that line. You can disable the breakpoint by clicking again.

```
9   #include "practice.hpp"
10  #include <iostream>
11
12  using namespace std;
13
14  void printLine(int width, bool fill) {
15      for (int i = 0; i < width; i++) {
16          if( i != 0 && i != width && !fill) {
17              cout << " ";
18          } else {
19              cout << "*";
20          }
21      }
22      cout << endl;
23  }
24
25  void printRectangle(int height, int width) {
26      for(int i = 0; i < height; i++) {
27          bool fill = false;
28          if(i == 0 || i == height - 1) {
29              fill = true;
30          }
31          printLine(width, fill);
32      }
33  }
34
35
36
37  int main() {
38      printRectangle(5, 8);
39      return 0;
40  }
41
```

```
1   #include <iostream>
2
3   using namespace std;
4
5   void printLine(int width, bool fill) {
6       for (int i = 0; i < width; i++) {
7           if (i != 0 && i != width && !fill) {
8               cout << " ";
9           }
10          else {
11              cout << "*";
12          }
13      }
14      cout << endl;
15  }
16
17  void printRectangle(int height, int width) {
18      for (int i = 0; i < height; i++) {
19          bool fill = false;
20          if (i == 0 || i == height - 1) {
21              fill = true;
Location: Source.cpp, line 21 ('printRectangle(int height, int width)')
23          printLine(width, fill);
24      }
25  }
```

What can we see once we've stopped at a line?

Values of variables: There should be a box at the bottom that tells you the value of variables in the current scope. You can also hover over variables in your program to get the value. No more print statements whew!

```
23  }
24
25  void printRectangle(int height, int width) {
26      for(int i = 0; i < height; i++) {
27          bool fill = false;
28          if(i == 0 || i == height - 1) {
29              fill = true;
30          }
31          printLine(width, fill
32      }
33  }
34
35
36
37  int main() {
38      printRectangle(5, 8);
39      return 0;
40  }
41
```

false

debugging-w

height = (int) 5
width = (int) 8
i = (int) 0
fill = (bool) false

```
17  void printRectangle(int height, int width) {
18      for (int i = 0; i < height; i++) {
19          bool fill = false;
20          if (i == 0 || i == height - 1) {
21              fill = true;
22          }
23          printLine(width, fill);
24      }
```

height 5

≤1ms elapsed

100 %

Autos

Name	Value	Type
fill	false	bool
height	5	int
i	0	int

What can we see once we've stopped at a line?

Stack: You can see what function(s) you're inside of.

debugging-workshop-cs32 〉 Thread

0 printLine(int, bool)
1 printRectangle(int, int)
2 main
3 start

```
(lldb)
```

```
12        }
13    }
14    cout << endl;
15 }
16
17 void printRectangle(int height, int width) {
18     for (int i = 0; i < height; i++) {
19         bool fill = false;
```

Summary | Events | Memory Usage | CPU Usage

Events

Show Events (3 of 5)

Memory Usage

Take Snapshot

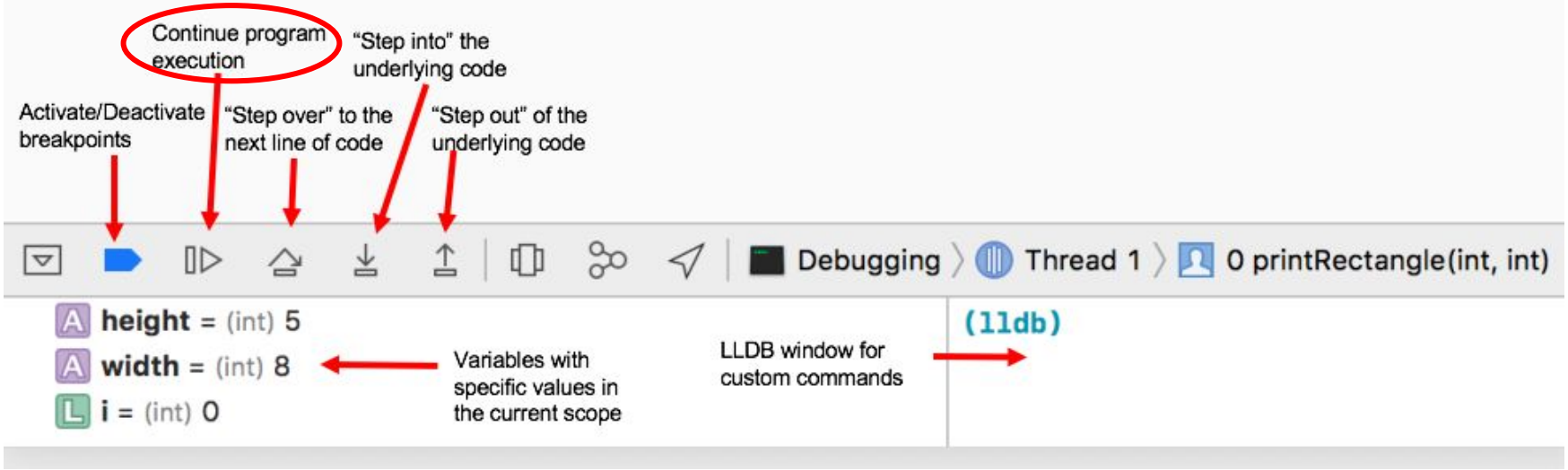
Value	Type
true	bool
8	int

Call Stack

Name	Lan
Project1.exe!printLine(int width, bool fill) Line 5	C++
Project1.exe!printRectangle(int height, int width) Line 23	C++
Project1.exe!main() Line 30	C++
[External Code]	
[Frames below may be incorrect and/or missing, no symbols loaded for kernel32.dll]	Un..

Continue

To continue to the next breakpoint, click "continue." Everything in between will be run, but you don't stop at each line.



For example, if we are at the first breakpoint in the following code snippet and we click "continue," we will run the code until the second breakpoint where we stop.

```
void printRectangle(int height, int width) {
    for (int i = 0; i < height; i++) {
        bool fill = false;
        if (i == 0 || i == height - 1) { // first breakpoint set here
            fill = true;
        }
        printLine(width, fill); // second breakpoint set here
    }
}
```

stop here

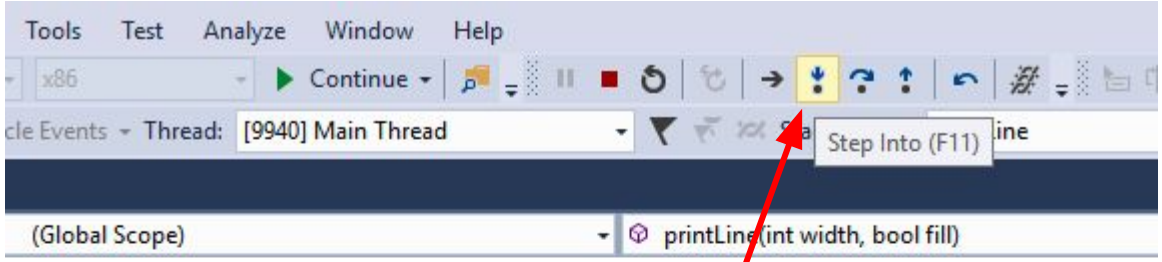
if you click continue...

keep running until the next breakpoint is reached

Step into

```
13
14 void printLine(int width, bool fill) {
15     for (int i = 0; i < width; i++) {
16         if( i != 0 && i != width && !fill) {
17             cout << " ";
18         } else {
19             cout << "*";
20         }
21     }
22     cout << endl;
23 }
```

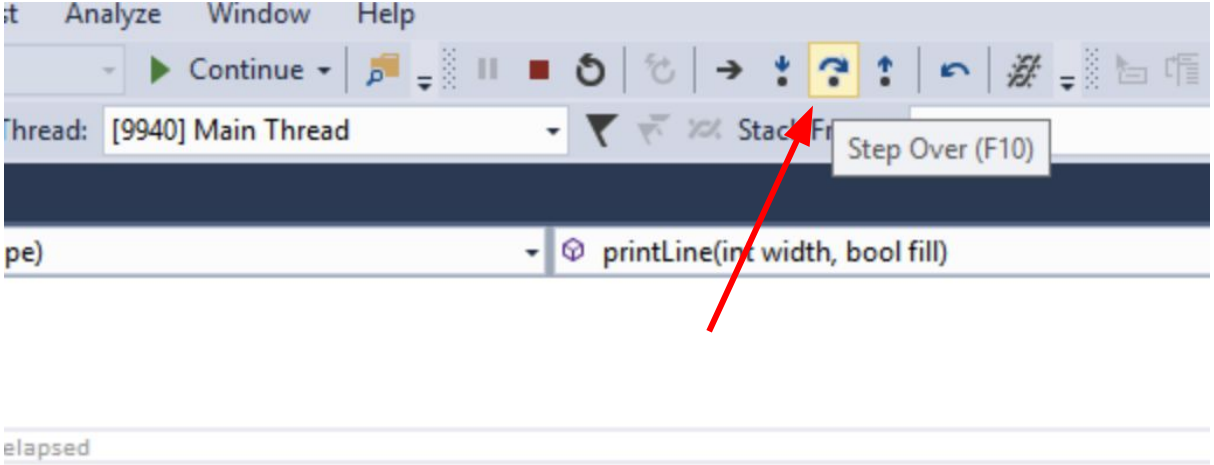
Thread 1: step in



If you come to a line where you call a function, "step into" will take you into that function and start at that function's first line. For example, if we are in `printRectangle` and we step into `printLine`, we will stop at the first line of `printLine`.

Step over

If you come to a line where you call a function, "step over" will continue **<u>until after that function has finished</u>** being run.



For example, if we are in `printRectangle` and we step over `printLine`, we will continue to the top of the for loop to check the condition `i < height`.

```
void printLine(int width, bool fill) {
    for (int i = 0; i < width; i++) {
        if (i != 0 && i != width && !fill) {
            cout << " ";
        } else {
            cout << "*";
        }
    }
    cout << endl;
}

void printRectangle(int height, int width) {
    for (int i = 0; i < height; i++) { // all of printLine will be run and we will stop here
        bool fill = false;
        if (i == 0 || i == height - 1) {
            fill = true;
        }
        printLine(width, fill); // breakpoint set here
    }
}
```

the whole printLine function will be run

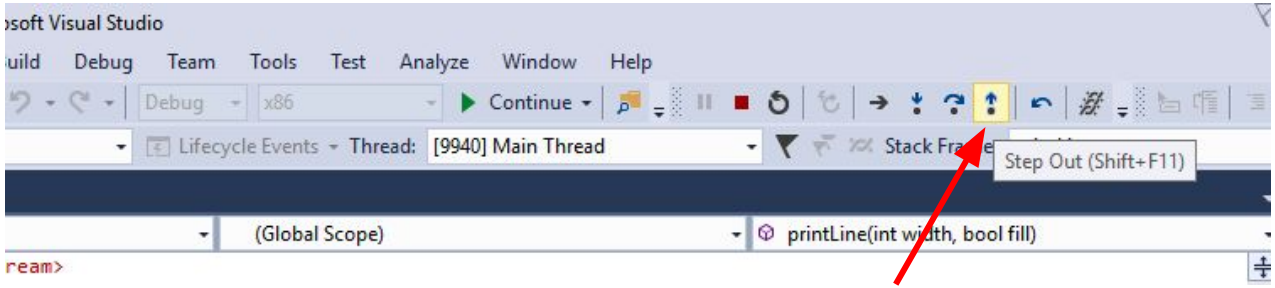
keep running until we leave printLine

stop here

if you click step over...

Step out

If you are inside a function and would like to continue until you exit that function, use "step out."



For example, let's say you have stepped into the `printLine(width, fill)` function while debugging `printRectangle`. Now, as you are inside printLine, use "step out" if you want to skip the rest of printLine and continue to what happens after the `printLine(width, fill)` line finishes, back in the function `printRectangle`.

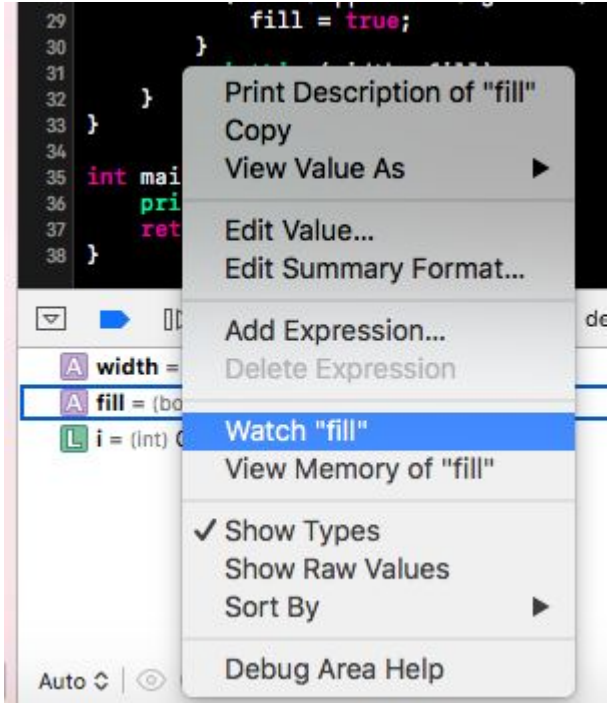
Watchpoints

Watchpoints are used to track any changes to the value of a variable. Instead of stopping execution when execution hits a particular line or function like a breakpoint does, watchpoints stop execution when the data of interest changes. Watchpoints are especially useful when you have variables that are changed on multiple lines or multiple if statements, or when you think you have a bug that relates to specific variables.

Note: you must set a breakpoint and start debugging the program before setting a watchpoint.

Watchpoints (XCode)

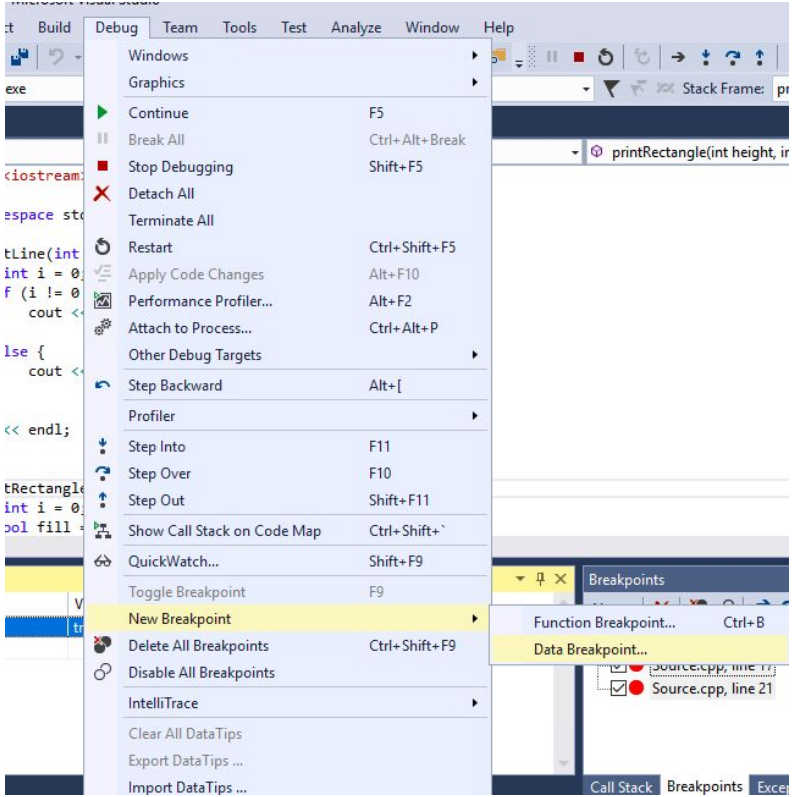
In XCode, while debugging, right-click the variable of interest listed in the bottom left window and select watch "fill".



Watchpoints (VS)

Two ways to watch a variable:

- Debug tab -> New Breakpoint -> Data Breakpoint
    - Put the address of the variable you are interested in (i.e. &fill). By default this will be a regular watchpoint that breaks execution when value changes, but if you're interested, there are additional options
- Or, right-click the variable (in the bottom left window or in the code) and select Add Watch, which sets a default watchpoint that breaks when the value changes.



Demo time!

Questions?