CS4414 Recitation 1

Course Introduction and C++ Setup

01/27/2022

Alicia Yang, Ricky Takkar

Recitation Overview

- Recitations introduction
- Programming environment setup
- Beginning C++ introduction, helloworld.cpp program demo and explain
- System performance

About TA --- Alicia

- 3rd year PhD student in CS
- Advised by Prof. Birman in the area of distributed system
- TA experience:
  - CS4320 Database System (Fall '19)
  - CS4412 System Programing (Fall '20, Fall '21)
  - CS5412 Cloud Computing (Spring '21, Spring '22)
- Office Hours :
  - **Thursday 4:15 - 5:15PM, Upson102**
  - **Saturday 7:30 – 9:30PM, Zoom**

About TA --- Ricky

- 2nd year PhD student in Systems Engineering
- Advised by Oliver Gao and Ken Birman. Topic: Verifiability via blockchain.
- TA experience:
  - CEE/ENMGT 5900 Project Management (Spring '22, Fall '22)
- Office Hours :
  - **Tuesday 2:00 - 3:00PM, CRP 104**
  - **Thursday 2:00 – 4:00PM, Zoom**

Goals

Develop **systems** in **C++** that **perform well**

Goals

Develop **systems** in **C++** that **perform well**

For the recitation:

- Basic C++ proficiency: Read, write, and debug C++ code
- Working knowledge of Linux: the Linux command line and the file system

Goals for the recitation

- Basic C++ proficiency: Read, write, and debug C++ code
  - Standard containers – std::vector<T>, std::map<K,V>
  - Memory management, RAII principle
  - gdb for debugging, gprof for profiling
  - Multi-threading, synchronization
- Working knowledge of Linux: the Linux command line and the file system

Secondary goals

- Make efficient use of hardware – learn to exploit CPU cores with threads
- Demystify systems' program
- Understand solutions to assignments/exams

Make recitations useful

- Ask questions
- Co-creating the recitations:
  - Post comments on Ed discussions(link) about the topics you find interesting and want to learn more in-depth about in recitations
- Try out the small puzzle in the end of the recitation
  - Test out the confusion with a simple runnable program

demo

C++ Environment Setup

C++ Coding Environment

- Compilation tools: GNU Compiler Collection(GCC) with **gcc-8 or recent**
  - Check your gcc compiler version: run command " g++ -v "
  - Most **linux** distrubutions have GCC
  - MacOS has Clang compiler (do **not** use this for this course, since Clang and GCC compiler have different compilation results on certain programs. The submitted assignments are run via GCC)
- C++ version: 20
  - Compile code with flag: -std=c++2a

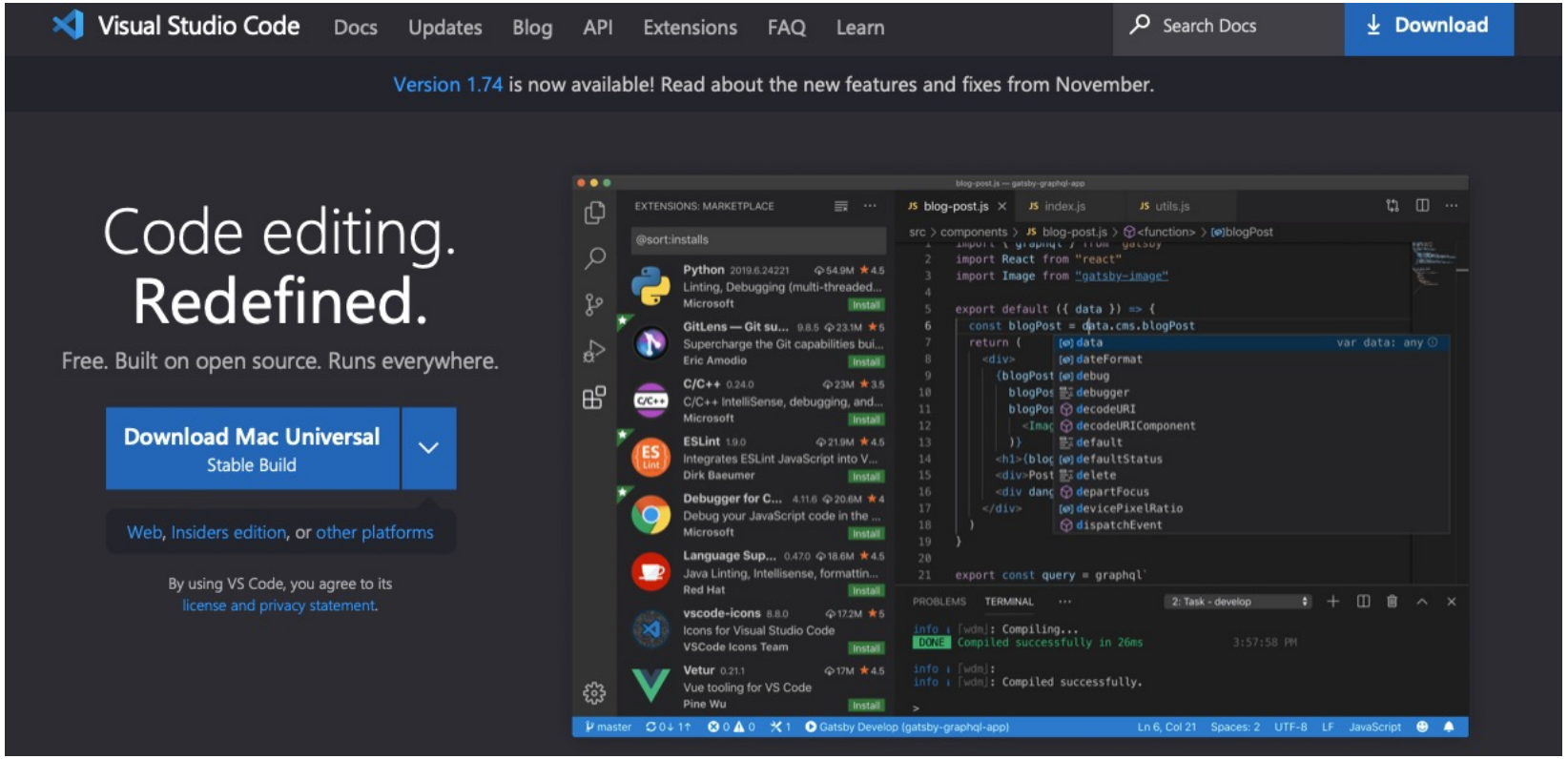
C++ Coding Environment

For the course assignments, it's optimal to use the following standard environment setup, which we have set up to have the required C++ and compiler version

- Server:
  - Cornell engineering ugclinux server remote access (link)
- Editing Tools:
  - Visual Studio Code with C/C++ extension

Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)



Coding Environment Setup Steps

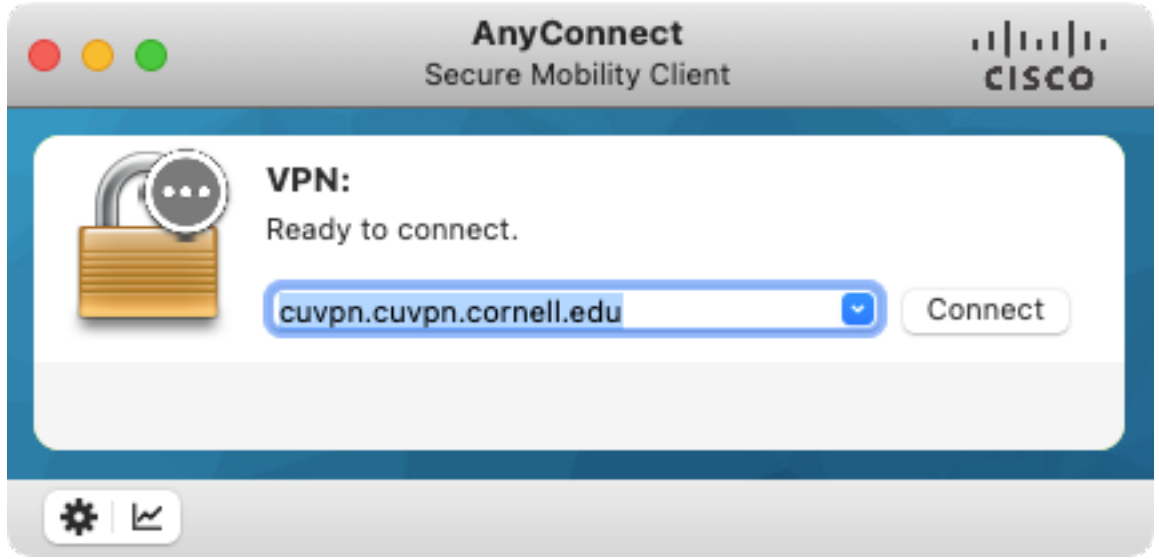
1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN (if you are on-campus using eduroam wifi, then skip this step)

Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
    1. Install CU VPN (link)
    2. Login using your Cornell id,

       Cornell password,

       and type ”push” for DUO confirm on your phone app(link)

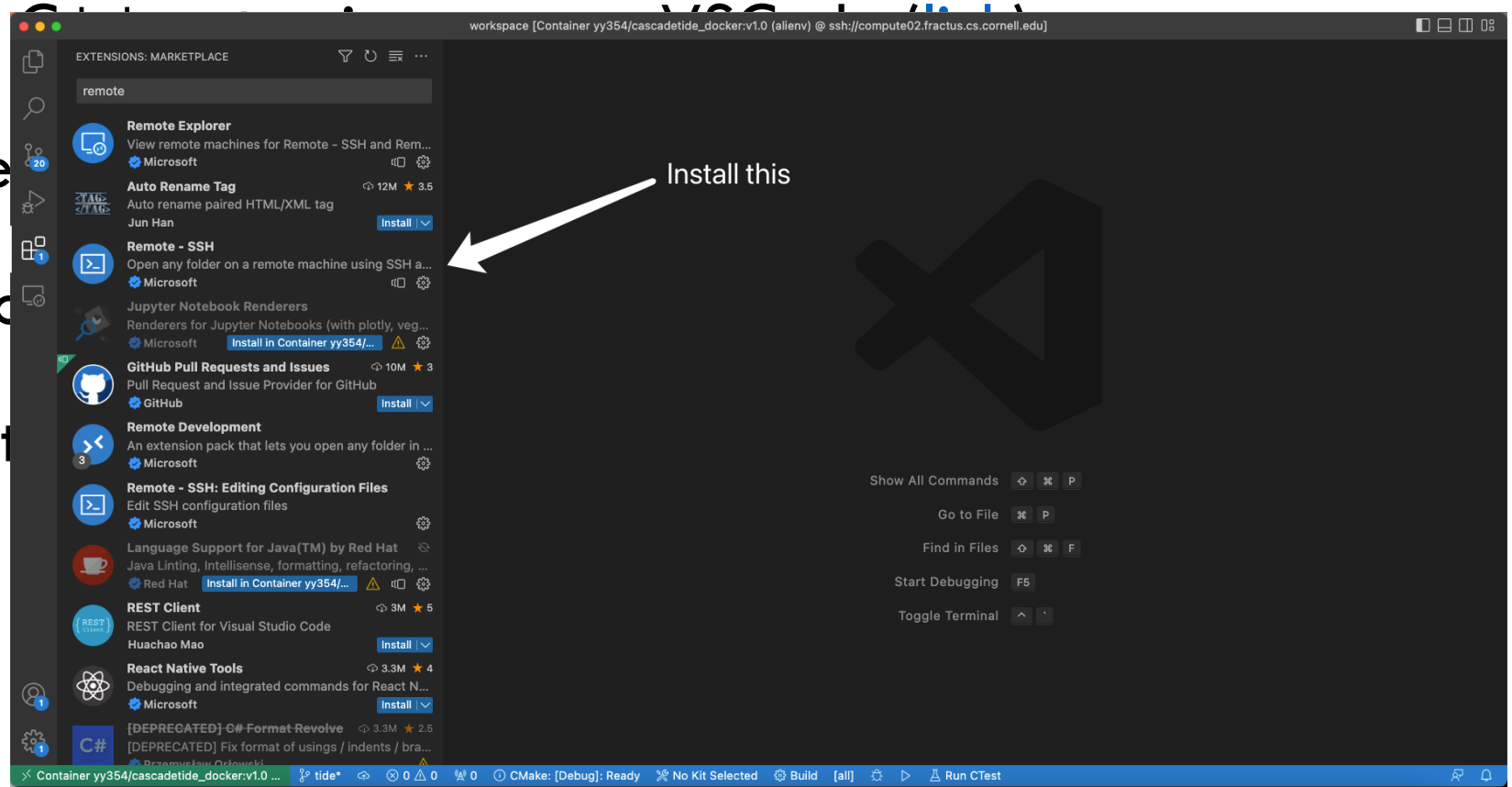


Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
3. SSH to your ugclinux server from VSCode
    1. Install Remote Explorer extension

Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Install C
3. Conne
4. SSH to
    1. Inst

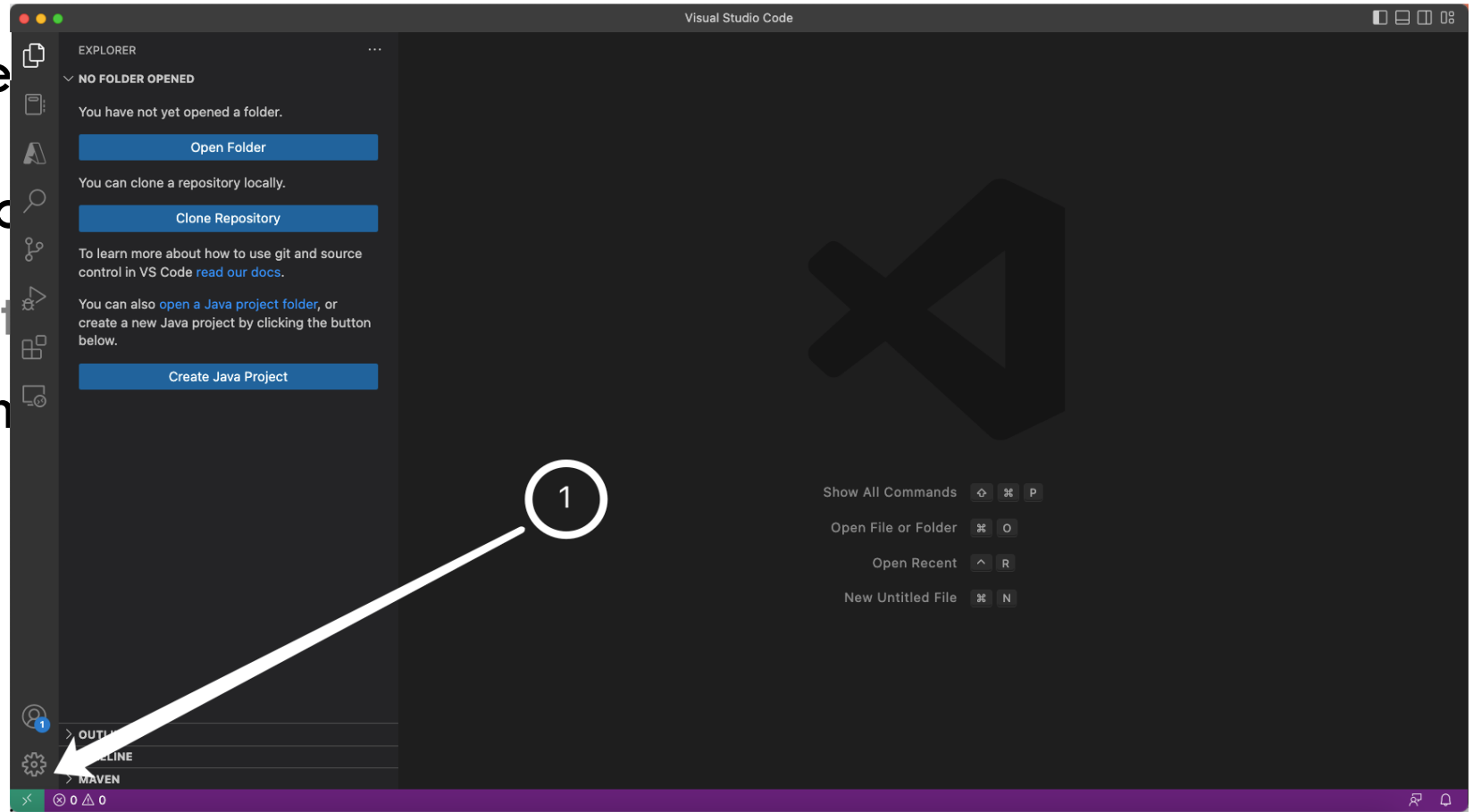


Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
3. SSH to your ugclinux server from VSCode
    1. Install Remote Explorer extension
    2. On VSCode: view -> command Palette

Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Conne
3. SSH to
    1. Inst
    2. On

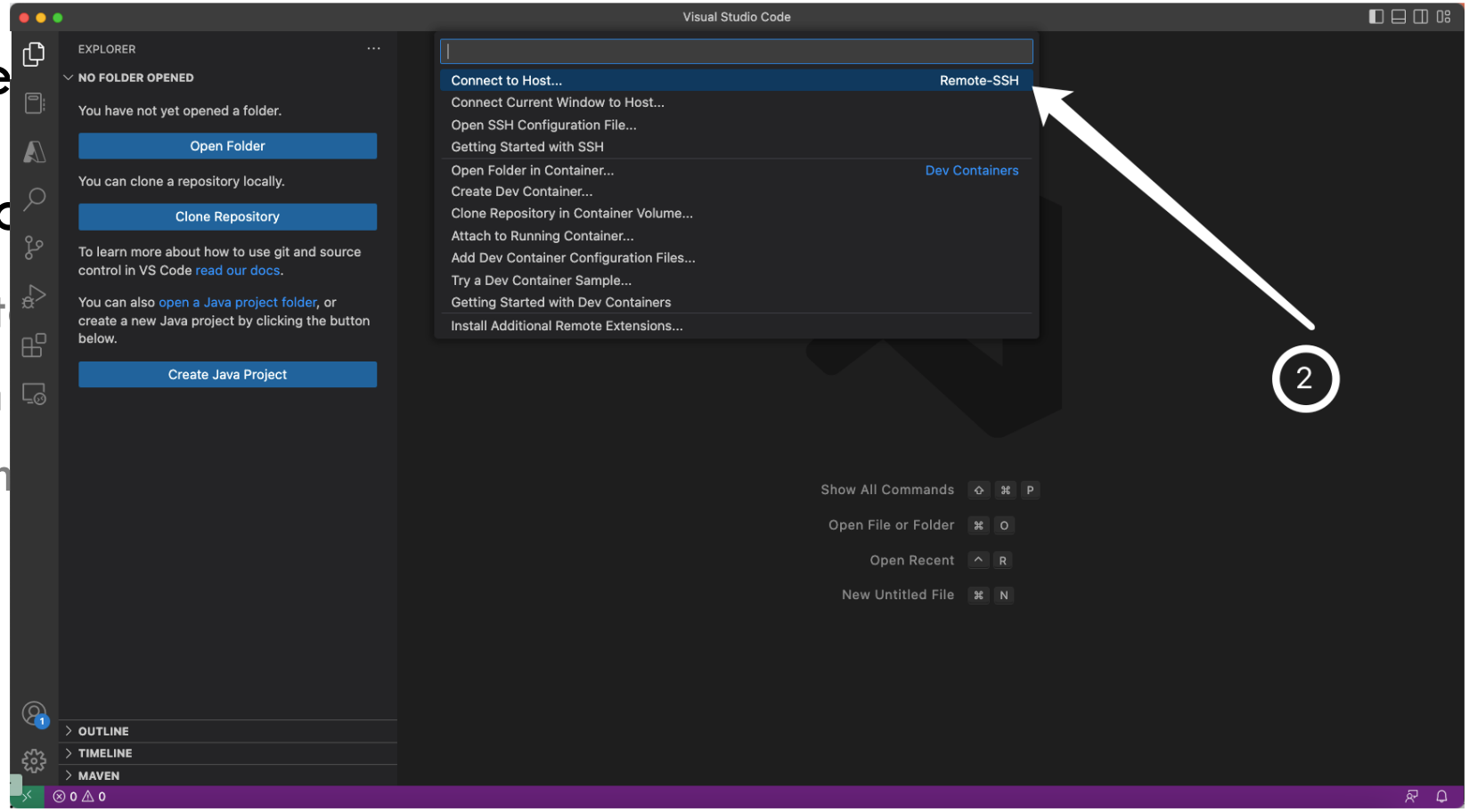


Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
3. SSH to your ugclinux server from VSCode
    1. Install Remote Explorer extension
    2. On VSCode: view -> command Palette
    3. Remote SSH: Connect to host

Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Conne
3. SSH to
    1. Inst
    2. On
    3. Ren

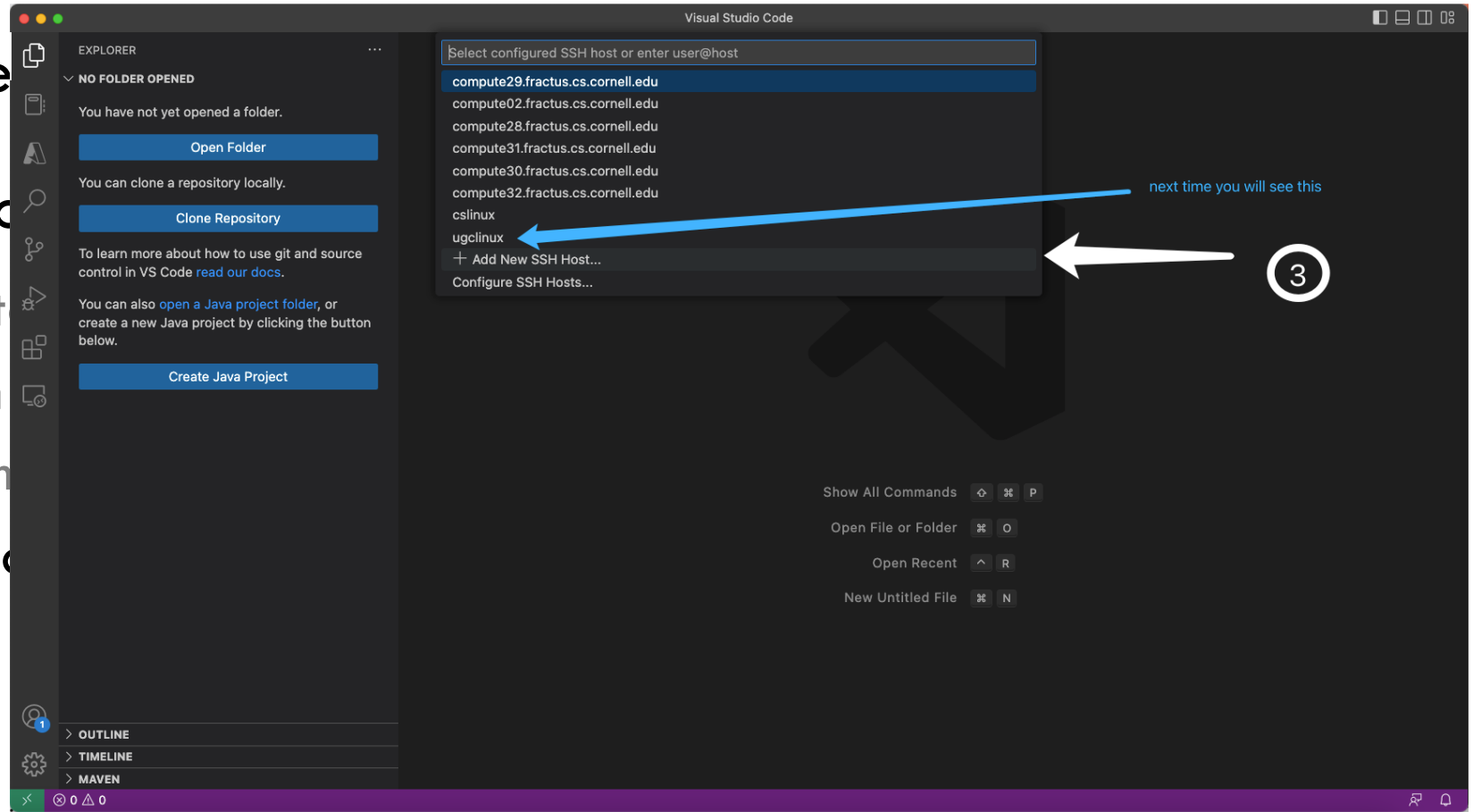


Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
3. SSH to your ugclinux server from VSCode
    1. Install Remote Explorer extension
    2. On VSCode: view -> command Palette
    3. Remote SSH: Connect to host
    4. Add New SSH Host

Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Conne
3. SSH to
    1. Inst
    2. On
    3. Ren
    4. Ad

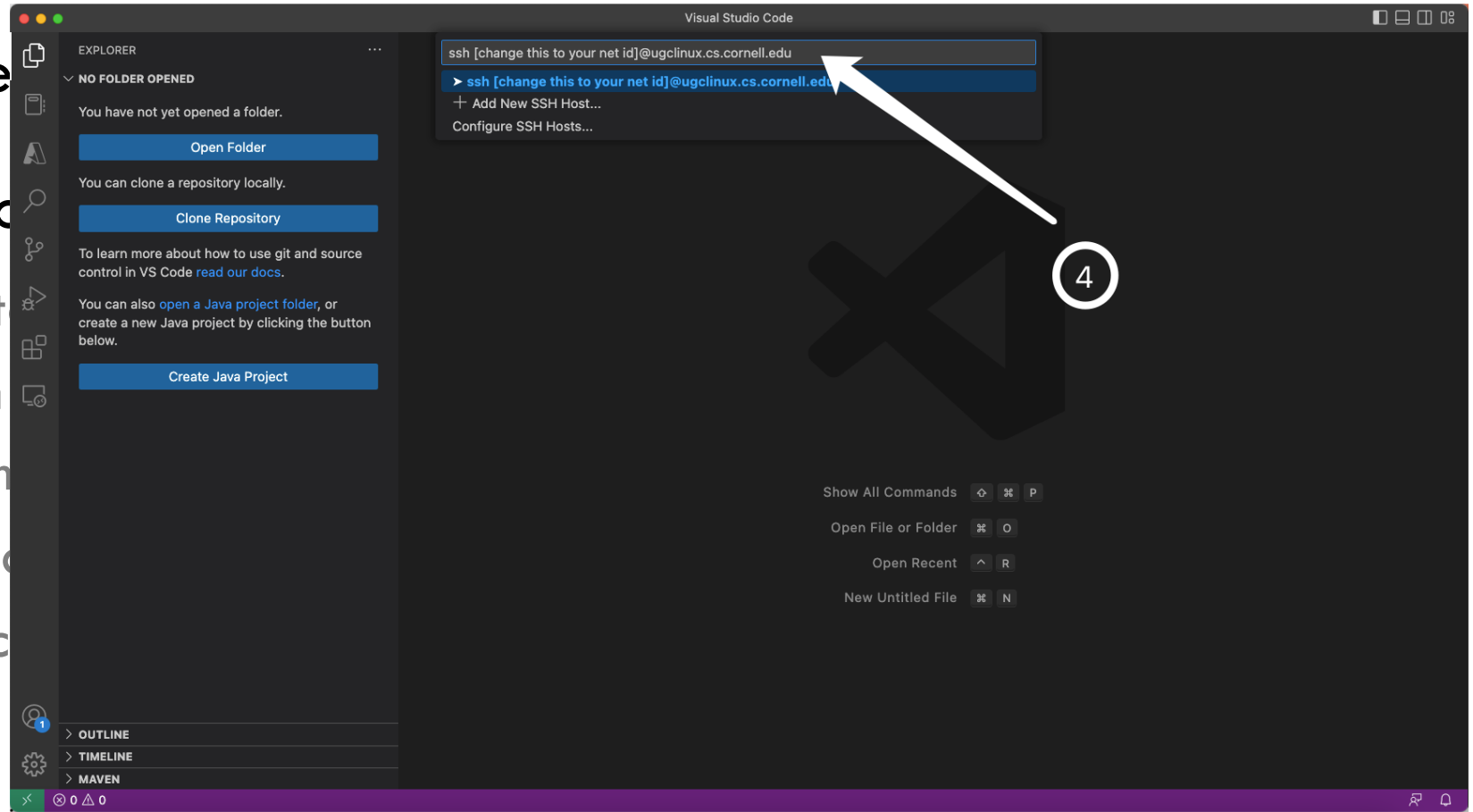


Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
3. SSH to your ugclinux server from VSCode
    1. Install Remote Explorer extension
    2. On VSCode: view -> command Palette
    3. Remote SSH: Connect to host
    4. Add New SSH Host
    5. In command palette, type: ssh [your netid]@ugclinux.cs.cornell.edu

Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Conne
3. SSH to
    1. Inst
    2. On
    3. Ren
    4. Ado
    5. In c

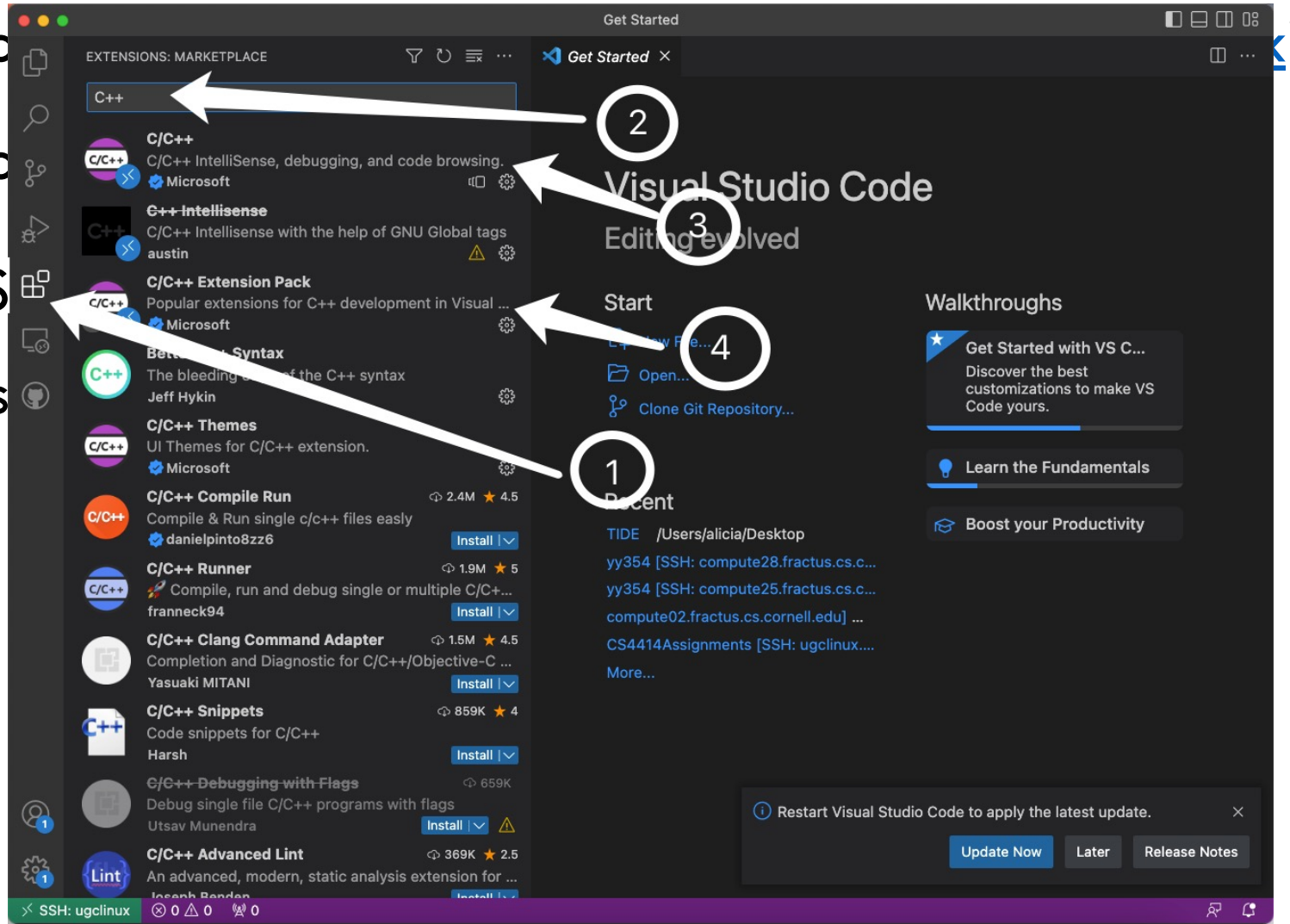


Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
3. SSH to your ugclinux server from VSCode
    1. Install Remote Explorer extension
    2. On VSCode: view -> command Palette
    3. Remote SSH: Connect to host
    4. Add New SSH Host
    5. In command palette, type: ssh [your netid]@ugclinux.cs.cornell.edu
    6. Type in the password related to your cornell netID, to access your ugclinux server

Coding Environment Setup Steps

1. Do
2. Co
3. SS
4. Ins



Coding Environment Setup Steps

1. Download Visual Studio Code on your computer (link)
2. Connect to Cornell VPN
3. SSH to your ugclinux server from VSCode
4. Install C++ extension on your VSCode (link)
    1. Click on extension tab on the left side of VSCode screen
    2. Search with key word C++
    3. Install package C/C++, for code browsing and debugging
    4. [optional] install package C/C++ Extension Pack for CMake tools

Coding Environment Setup Steps

Congratulations

You are all set to start your first program!

C++ Basics

What is C++?

A federation of related languages, with four primary sublanguages

- **C:** C++ is based on C, while offering approaches superior to C. Blocks, statements, processor, built-in data types, arrays, pointers, etc., all come from C
- **Object-Oriented C++:** "C with Classes", classes including constructor, destructors, inheritance, virtual functions, etc.
- **Template C++:** generic programming language. Gives a template, define rules and pattern of computation, to be used across different classed.
- **STL(standard template library):** a special template library with conventions regarding containers, iterators, algorithms, and function objects

demo

helloworld.cpp exampe

Helloworld.cpp example

```
#include <iostream>


int main() {
	std::cout << "Hello world!" << std::endl;
	std::cout << "Please type in your name: " << std::endl;
	std::string name;
	std::getline(std::cin, name);
	std::cout << "Hi " << name << std::endl;
	return 0;
}
```

Run your C++ Code

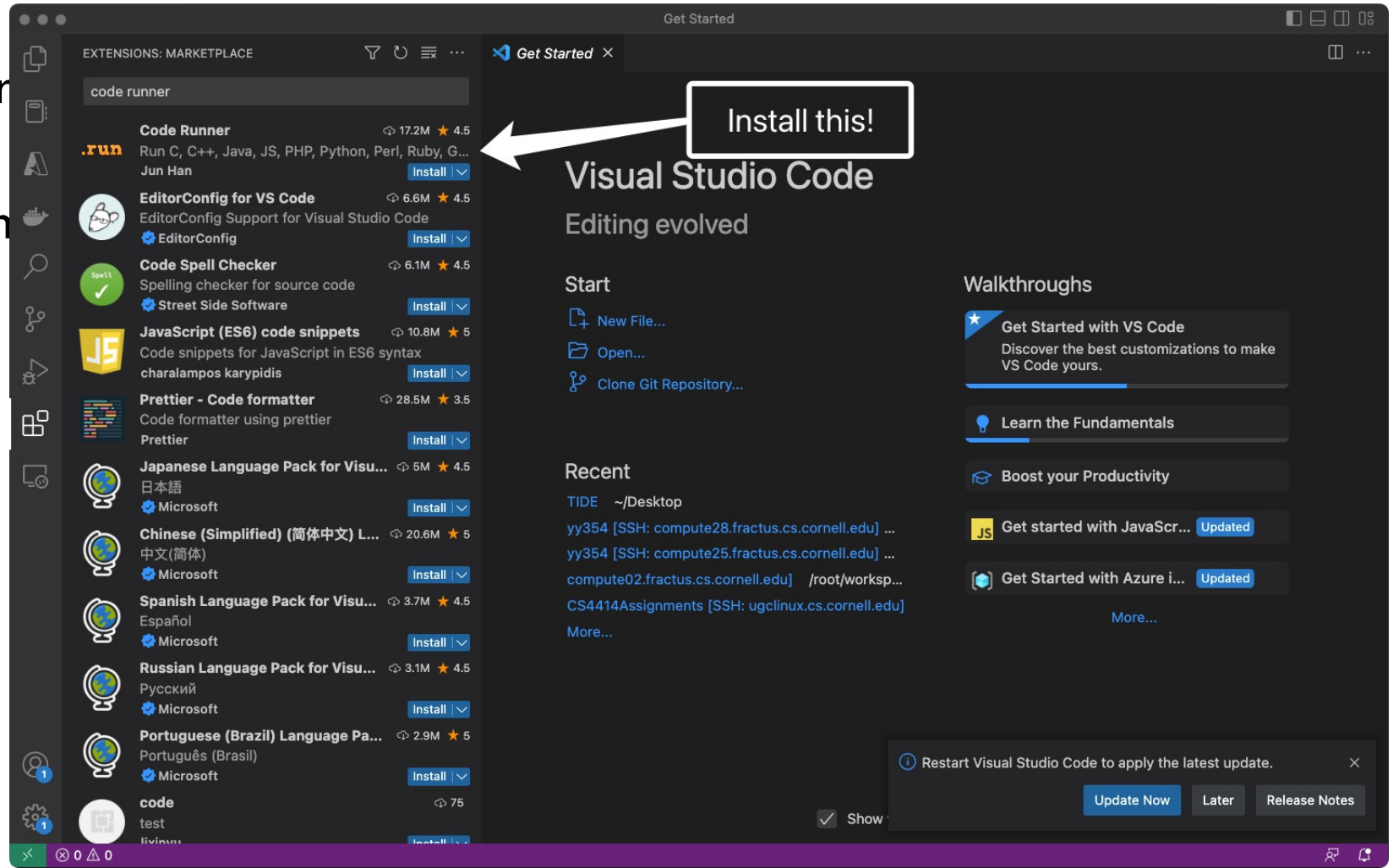
- Beginner: use <run> shortcut button on VSCode
  - You will be able to see the run button on the top right corner of vscode, after installing the C/C++ Compile Run extension (refer to step 4 in coding environment setup page)

Run your C++ Code

- Beginner: use <run> shortcut button on VSCode
  - Install an extension called code runner

Run your C++ Code

- Begin
  - In

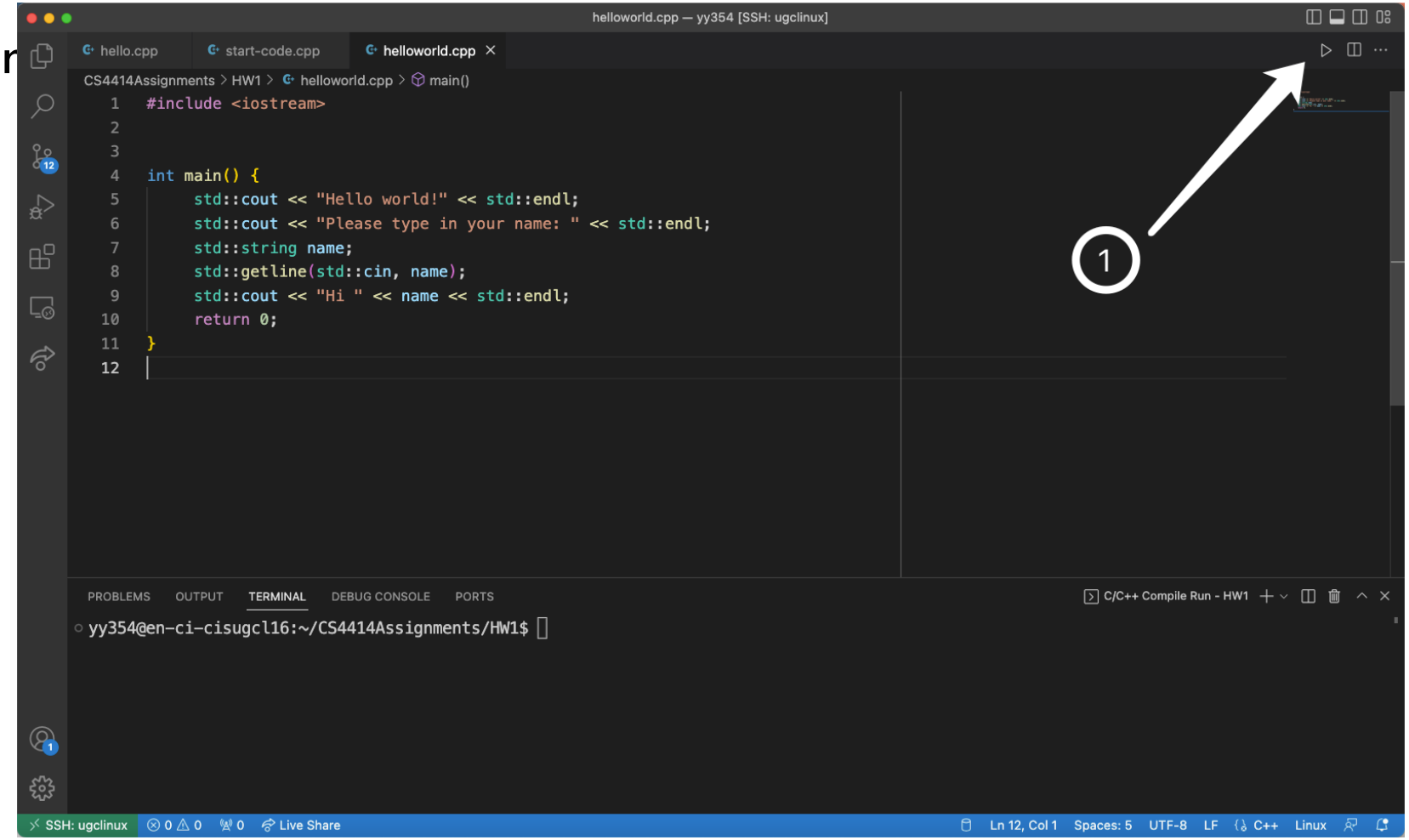


Run your C++ Code

- Beginner: use <run> shortcut button on VSCode
  - You will be able to see the run button on the top right corner of vscode, after installing the C/C++ Compile Run extension (refer to step 4 in coding environment setup page)

Run your C++ Code

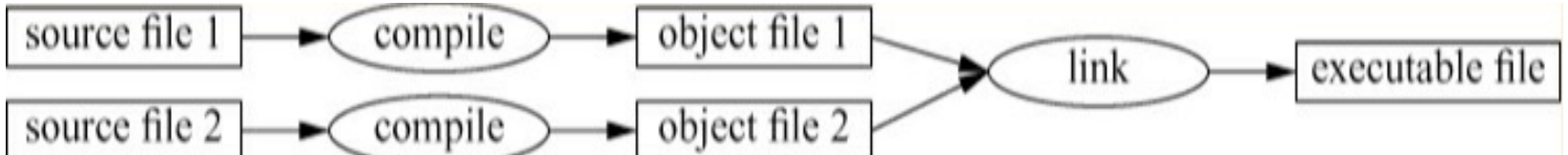
- Begin



What's under the hood when clicking run?

C++ is a compiled language.

- For a program to run, its source text has to be processed by a compiler, producing object files
- Linker combines the object files and generate an executable program



Compile and Run your C++ Code

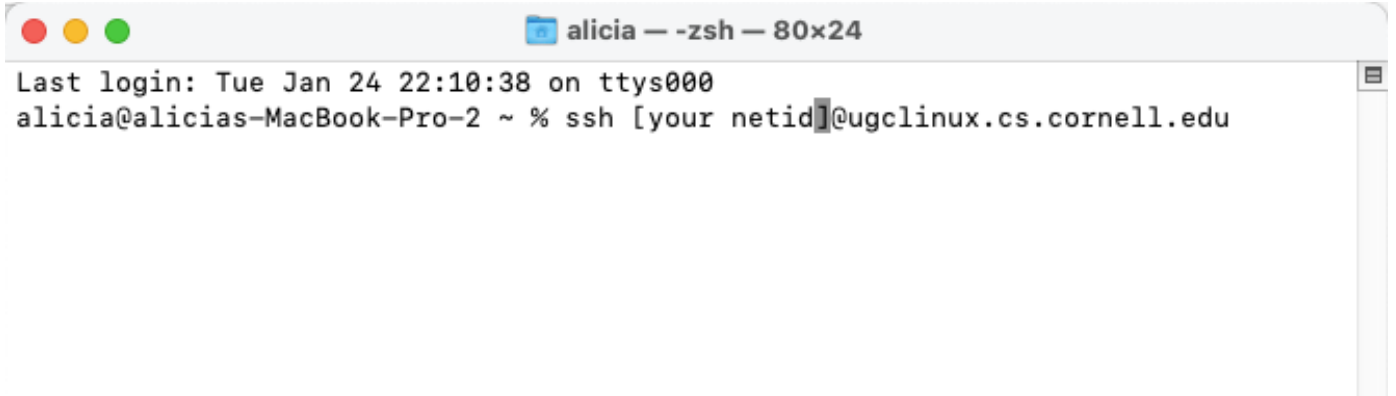
- Beginner: use <run> shortcut button on VSCode
- Recommended: use command line prompt to compile and run your C++ code

Compile and Run your C++ Code

- Beginner: use <run> shortcut button on VSCode
- Recommended: use command line prompt to compile and run your C++ code

1. From terminal login to ugclinux server, via ssh tunnel

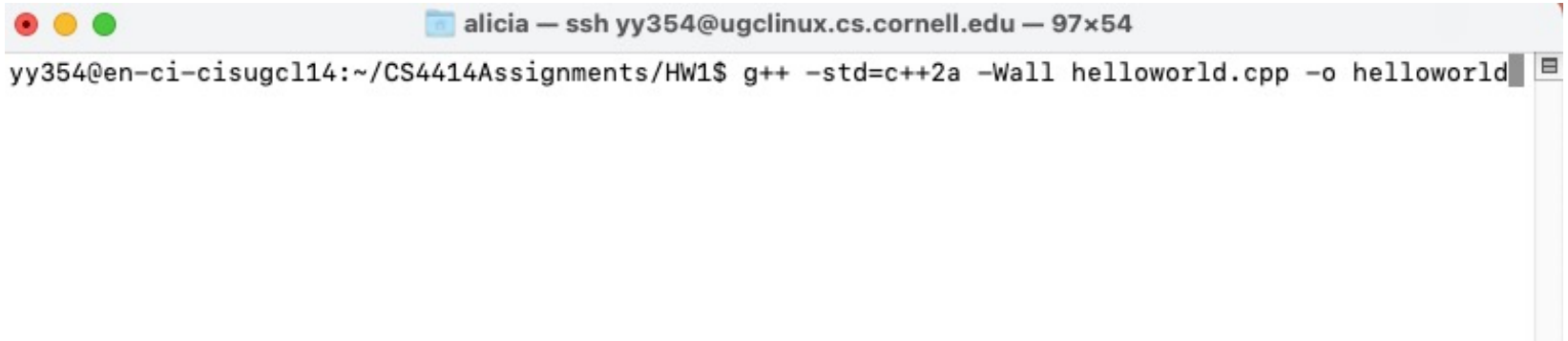
% ssh [your netid]@ugclinux.cs.cornell.edu



Compile and Run your C++ Code

- Beginner: use <run> shortcut button on VSCode
- Recommended: use command line prompt to compile and run your C++ code
  1. From terminal login to ugclinux server
  2. Compile your C++ program with simple line below

% g++ -std=c++2a -Wall helloworld.cpp -o helloworld



Compile and Run your C++ Code

- Beginner: use <run> shortcut button on VSCode
- Recommended: use command line prompt to compile and run your C++ code
  1. From terminal login to ugclinux server
  2. Compile your C++ program with simple line below

     % g++ -std=c++2a -Wall helloworld.cpp -o helloworld

     - Flags:
       - -std=c++2a: specify the compiler version to use C++20
       - -Wall: allow all compiler warnings to be printed out
       - -o: specify the name of the output executable

Compile and Run your C++ Code

- Beginner: use <run> shortcut button on VSCode
- Recommended: use command line prompt to compile and run your C++ code
  1. From terminal login to ugclinux server
  2. Compile your C++ program with simple line below
  3. Run the compiled executable program

     % ./helloworld

How to debug my code? --- GDB and example

demo

Gdb is a debugger tool, that allows us to

- See what is going on `inside' the program while it executes
- Checks what program was doing at the moment it crashed.

How to debug my code? --- GDB and example

1. Compile with –g flag:

```
%   g++  -std=c++2a  -Wall  helloworld.cpp -o helloworld
```

2. Run with gdb

```
%   gdb ./helloworld
```

3. Debug with gdb

How to debug my code? --- GDB and example

Useful commands in gbd

- **run** or **r** → execute the program from start to the end
- break or br → sets breakpoint
  - break *function* → stop at a particular function
  - break *linenum* → stop at a particular line
- next or n → execute next line of code
- step → go to next line of instruction
- print or p [variable] → print the stored value
- quit or q → exits out of gdb

More about GDB

Why I observe segmentation fault in execution but not in GDB?

- gdb default Disabling Address Space Layout Randomization. This can be solved by turn off this feature before run gdb

  (gdb) set disable-randomization off

- Optimization level inconsistency between runtime program, and debugging program

  compile the code with same level of optimization –O_ , more explanation (link)

- gdb set LINES and COLUMNS in program's environment, which will alter the size of environment, such as the stack size.

  (gdb) unset environment COLUMNS and (gdb) unset environment LINES

System Performance will be a mainstay of this course!

What do we mean by performance?

- **Latency**: time taken to compute
- **Throughput**: number of operations per second

Reasoning about system performance

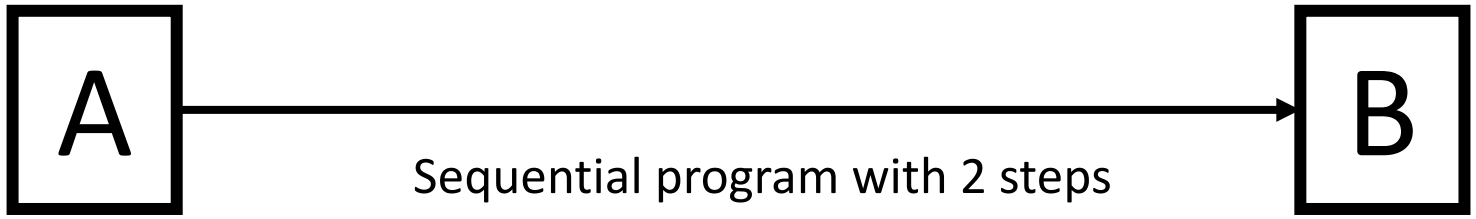
- Theoretical improvements don't always translate to better runtimes

Insertion sort outperforms quick sort in some cases
Why?

1. Insertion sort is iterative – no overhead from recursive calls (good for sorting a small set)
2. Insertion sort is fast when data is nearly sorted

Reasoning about system performance

- Theoretical improvements don't always translate to better runtimes
- Which algorithm? A system can be very complex with many features



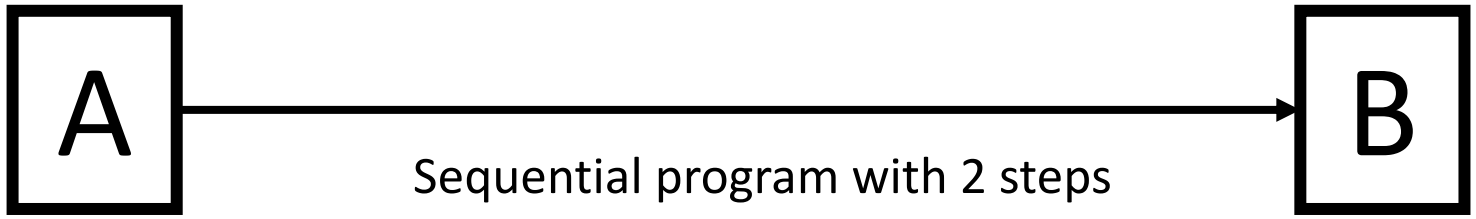
Fairly optimized code

Highly inefficient code

- A = processing files, B = printing 1 million lines of output

Reasoning about system performance

- Theoretical improvements don't always translate to better runtimes
- Which algorithm? A system can be very complex with many features



Fairly optimized code

Highly inefficient code

- What if step A takes about 99% of the total time? We need to profile and understand performance characteristics of code we write

Reasoning about system performance

- Theoretical improvements don't always translate to better runtimes
- Which algorithm? A system can be very complex with many features
- What if the code that implements the algorithm is inefficient?
- Sometimes heuristics work better

Reference

- Effective C++: 55 specific ways to improve your programs and designs, Scott Meyers, 3rd edition
- A Tour of C++, Bjarne Stroustrup
- Large Scale C++, Process and Architecture, John Lakos, Volume 1
- GDB documentation: https://www.sourceware.org/gdb/
- https://www.geeksforgeeks.org/gdb-step-by-step-introduction/
- GDB quickstart tutorial: https://web.eecs.umich.edu/~sugih/pointers/gdbQS.html
- How does gbd work? https://www.aosabook.org/en/gdb.html
- CS4414 recitation slides, from Sagar Jha, TA for this course in 2020, 2021