Lecture 1: Course Overview & Introducing C

Sarah Nadi
nadi@ualberta.ca
Department of Computing Science
University of Alberta

CMPUT 201 - Practical Programming Methodology
Winter 2018

Agenda

- Course Overview
- Introducing Unix & C
- Introducing version control using git

Readings

- Look at eClass course page
- Textbook: Ch 1 & 2

Course Overview

Course Objectives

- Be able to handle any intermediate programming problem using C under Unix/Linux

- Have the skills to combine your knowledge to design and implement non-trivial software

    - program design and data structures (174/175)

    - useful algorithms and mathematics (272/204/304)

    - application-specific knowledge (291/379)

Goals

- Teach the C programming language
  - syntax, memory, debugging, execution
- Teach the basics of UNIX programming
  - using shell commands, standard in-out-error, pipes etc.
- Teach some practical aspects of programming methodology
  - common modularization techniques
  - testing
  - source code version control - using git

Official Class Information

- Course outline: https://eclass.srv.ualberta.ca/course/view.php?id=40979
- Overview of course schedule: https://eclass.srv.ualberta.ca/mod/page/view.php?id=2707174
- Textbook: K.N. King. C Programming: A Modern Approach, 2nd Edition, W.W. Norton & Company, 2008 .

Student Evaluation

- 3 assignments (8% each — 24% total)
- 12 lab exercises (best 10 of 12 — 15% total)
- 2 midterms (13% each — 26% total)
- 8 eClass quizzes (0.5% each — 4% total)
- Final (31%)

Assignments

- Three assignments almost 4 weeks apart
- Each assignment has explicit instructions on expectations. When in doubt, ask on the discussion forum!
- Each assignment counts for 8% of your grade
- Assignments will be automatically graded so **you must make sure to adhere to the requirements. If we cannot automatically grade your submission, you will get a 0.** We will provide scripts and test cases that will help you with that.

Quizzes

- 8 quizzes — check schedule on eClass
- 3-6 simple questions per quiz
- Quizzes become available on Monday 8am and close on Tuesday 8am
- Quizzes are timed (10-20min depending on quiz)
- **Do not post questions about quizzes during quiz time**
- Quizzes are meant to help you make sure you are on track

Labs

- Labs will start on Monday January 15th, 2018
- There are 12 lab exercises, each worth 1.5% of your final grade. Best 10 out of 12 labs will be used towards your final grade (which means you have 2 free excused absences).
- Read tutorials & start exercise before the lab
- You can only demo in your registered lab. No exceptions.
- Before next week, make sure that:
    - you get your UNIX (CS) accounts & test them before your lab
    - Have a CMPUT201 GitHub repository (more on that later)
    - contact HelpDesk if you cannot log in

Important Information

- Read the course policies on eClass. Pay specific attention to the collaboration policy.
- **General rule: No late submissions accepted for quizzes, assignments, or lab exercises**
- **All labs and assignments will be marked on the lab computers.**

Collaboration Policy

- **Write your own solution. All labs and assignments are individual!**
- You **should not** post your solutions on public code repositories — use the provided private repo & take a look at the CMPUT 201 License that applies to all solutions submitted in this course.
- What does collaborating mean ?
  - You can verbally talk about how to solve a particular problem in the assignment
  - You can draw pictures (e.g., of linked lists)
  - You can share test cases on the forums.
  - **You cannot share code. You cannot look at someone else's code. You cannot sit together side by side and code the solution step by step together. You cannot post any code snippets on the forum… you get the picture :-)**
- We will check for plagiarism both manually and through automated tools.

The “Ugly” Slide

- 10 plagiarism cases were reported in Winter 2017
- Sanctions included 0 on the assignment, letter grade reduction, and suspension. Some were in addition to an “8” note on your transcript.
- Getting an extra mark or two on an assignment is not worth jeopardizing your future!

How to Succeed in CMPUT 201

- Be fearless in “just trying it out”:
  - hard to do any permanent damage if you use backups and version control
  - write small programs to test out concepts
- Learn to find the answers for yourself
  - Your textbook is an excellent resources
  - `man` pages usually have the answer
  - Google your compiler error messages
- Learn how to use debugging tools (`gdb`, `valgrind`)

How to Succeed in CMPUT 201 *(Cont'd)*

- The difference between spending 10 hours vs. 100 hours on an assignment:
  - Follow “best practices” covered in class
  - Use the debugger, makefiles, and assertions
  - Think about your test cases, use incremental testing, and automate any repetitive tasks
  - start assignments on time
  - If you are completely stuck, ask your TAs and then the instructor

Additional Info

- Examples and additional details will be given in class & may not necessarily be documented in the slides

- Additional material related to the specified reading chapter may be covered in class/notes

- There will be group exercises during class. You can bring your laptop to code on it or you can try coding on a piece of paper (very useful for exam practice)

- We will be using metimeter.com for some questions during class. You can access the website on your phone or on your laptop

Introducing C & Unix

Why Unix?

- Dominant server operating system
- Uses open standards (e.g., POSIX threads)
- Free open-source versions available (FreeBSD, OpenBSD, Linux)
- Many free software development tools (e.g., gcc, emacs, gdb etc.)
- We will be using Linux in the labs

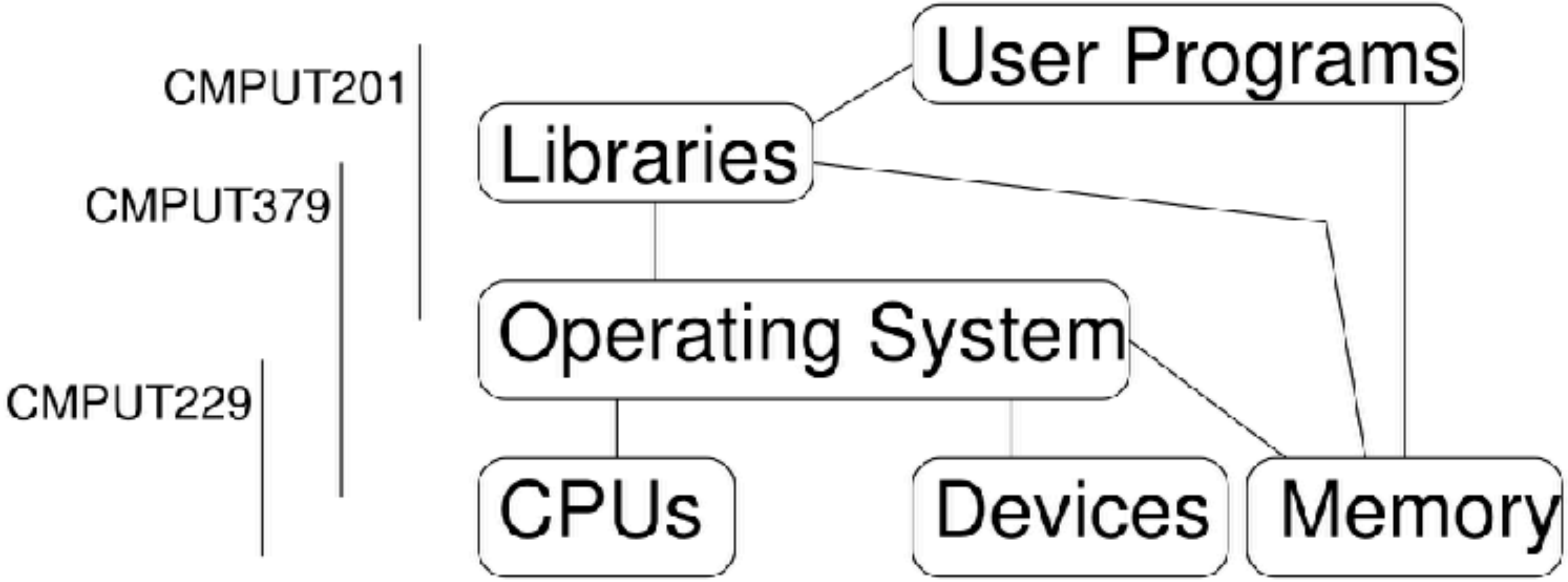
Why C & What is C?

- History of C
  - a byproduct of the UNIX operating system
  - a higher-level language than assembly
  - mostly done in the 1960s and 1970s
  - different language standards that evolved over time
- C influences many modern programming languages
  - C++, Java, C#, Perl
- Understanding how memory works helps you understand other programming languages as well

Strengths/Weaknesses of C

- Strengths:
  - efficiency, portability, power, flexibility, standard libraries
  - low-level, access to machine-level concepts
  - small, limited features
  - permissive; you need to know what you are doing
- Weaknesses:
  - error-prone, difficult to understand, difficult to modify

Relationship to Other Courses



In 201, we take a look at **using** the operating system UNIX & the high-level programming language C

Relationship to Other Software Engineering Courses

- 201: small-scale programming
- 301: team work, object-oriented design
- 401: large-scale programming

Relationship to the “Outside World”

Learning C also allows you to understand & appreciate higher-level programming languages, because you know what goes on “behind the scenes”

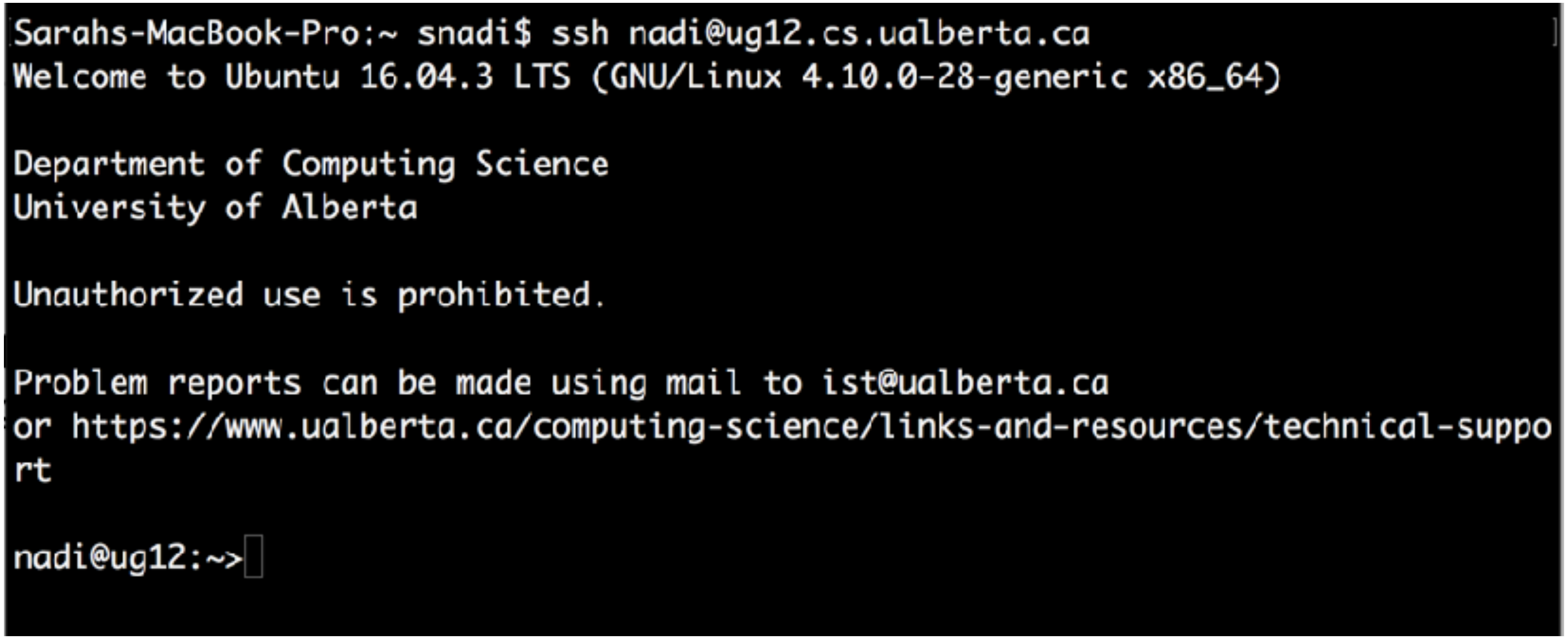
The UNIX/Linux Shell

- The machines in the CMPUT 201 lab are ugXX.cs.ualberta.ca, where XX ranges from 00 to 34

- If you are using a UNIX-based OS (e.g., Ubuntu or MacOS), just go to your terminal and use **`ssh`** to connect to the lab computer

- If you are using Windows, you will need an ssh client such as PuTTY

Example of Connecting to Lab Computer

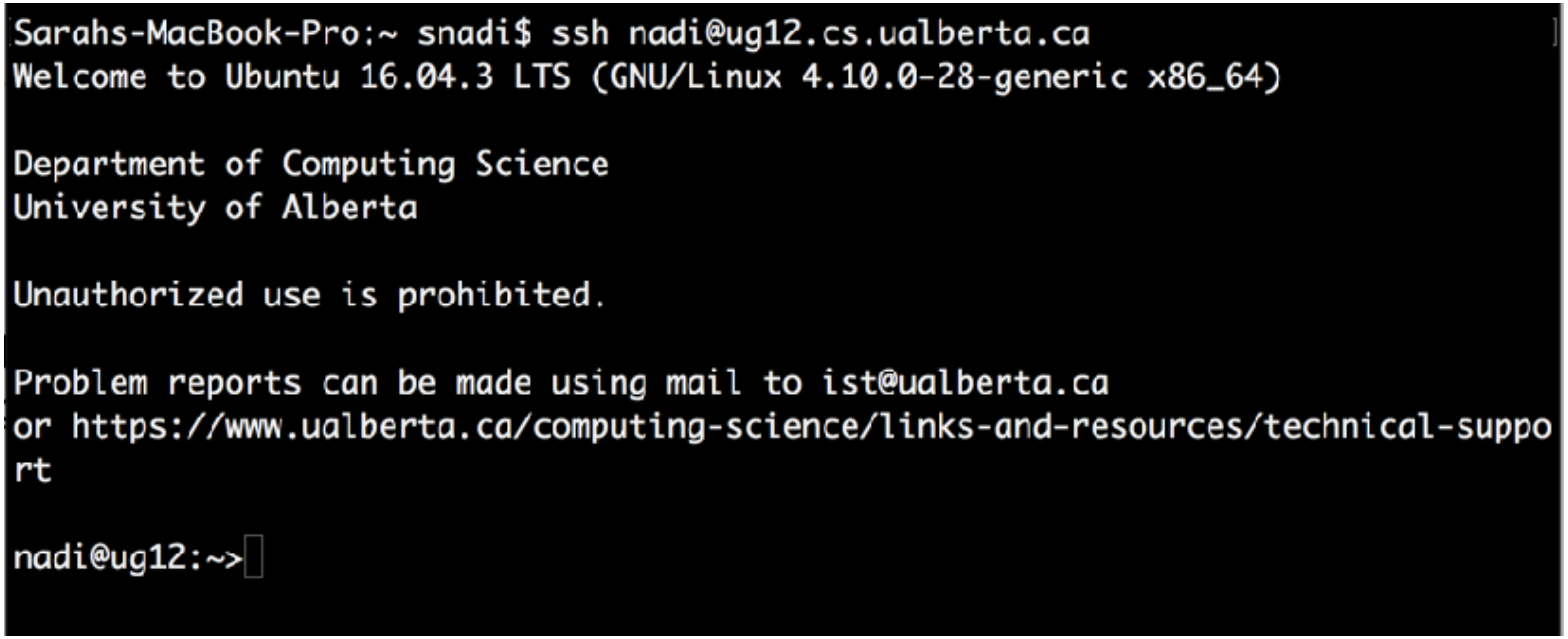


```
Sarahs-MacBook-Pro:~ snadi$ ssh nadi@ug12.cs.ualberta.ca
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64)

Department of Computing Science
University of Alberta

Unauthorized use is prohibited.

Problem reports can be made using mail to ist@ualberta.ca
or https://www.ualberta.ca/computing-science/links-and-resources/technical-suppo
rt

nadi@ug12:~>
```

demo

Example of Connecting to Lab Computer

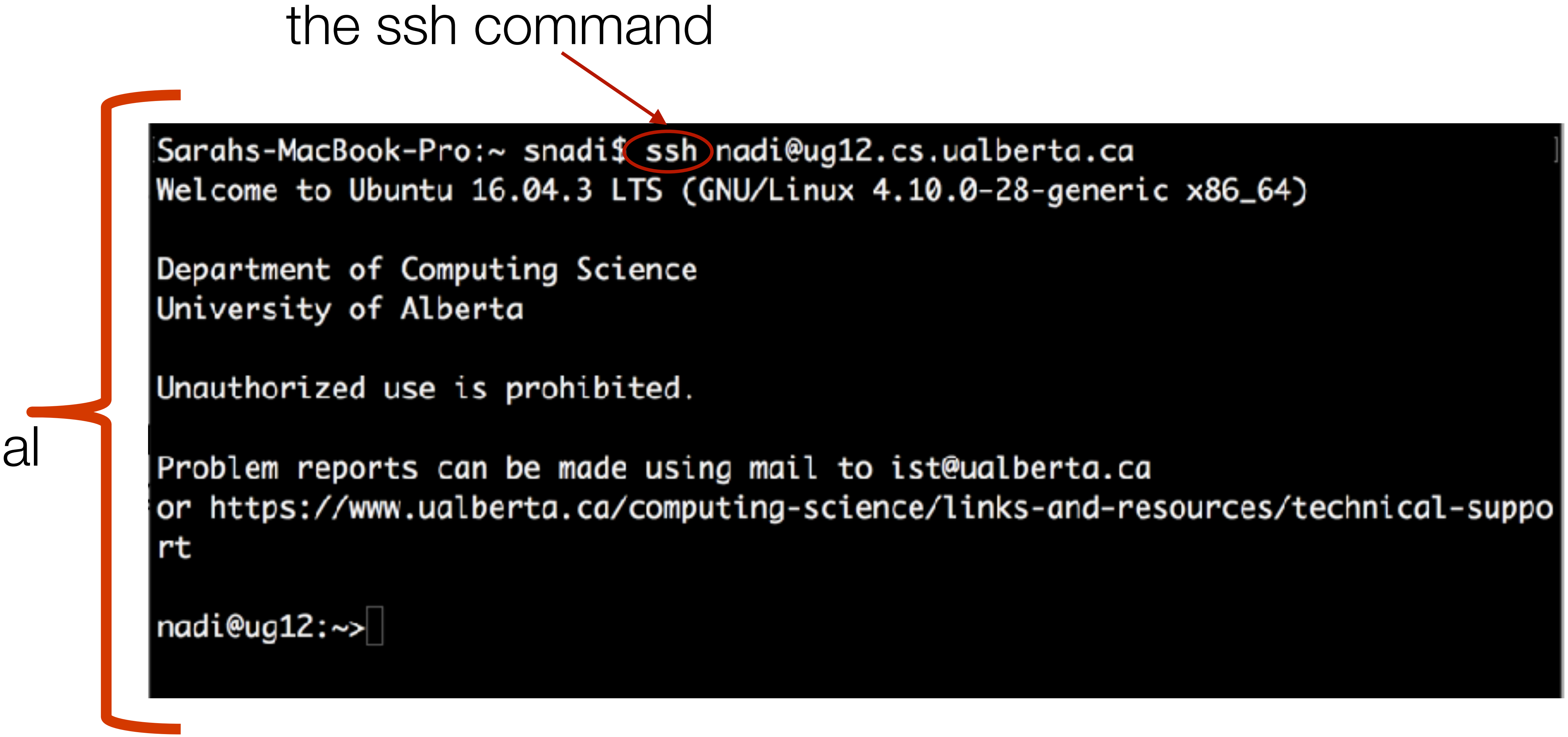
Shell/
terminal



```
Sarahs-MacBook-Pro:~ snadi$ ssh nadi@ug12.cs.ualberta.ca
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64)

Department of Computing Science
University of Alberta

Unauthorized use is prohibited.

Problem reports can be made using mail to ist@ualberta.ca
or https://www.ualberta.ca/computing-science/links-and-resources/technical-suppo
rt

nadi@ug12:~>
```

demo

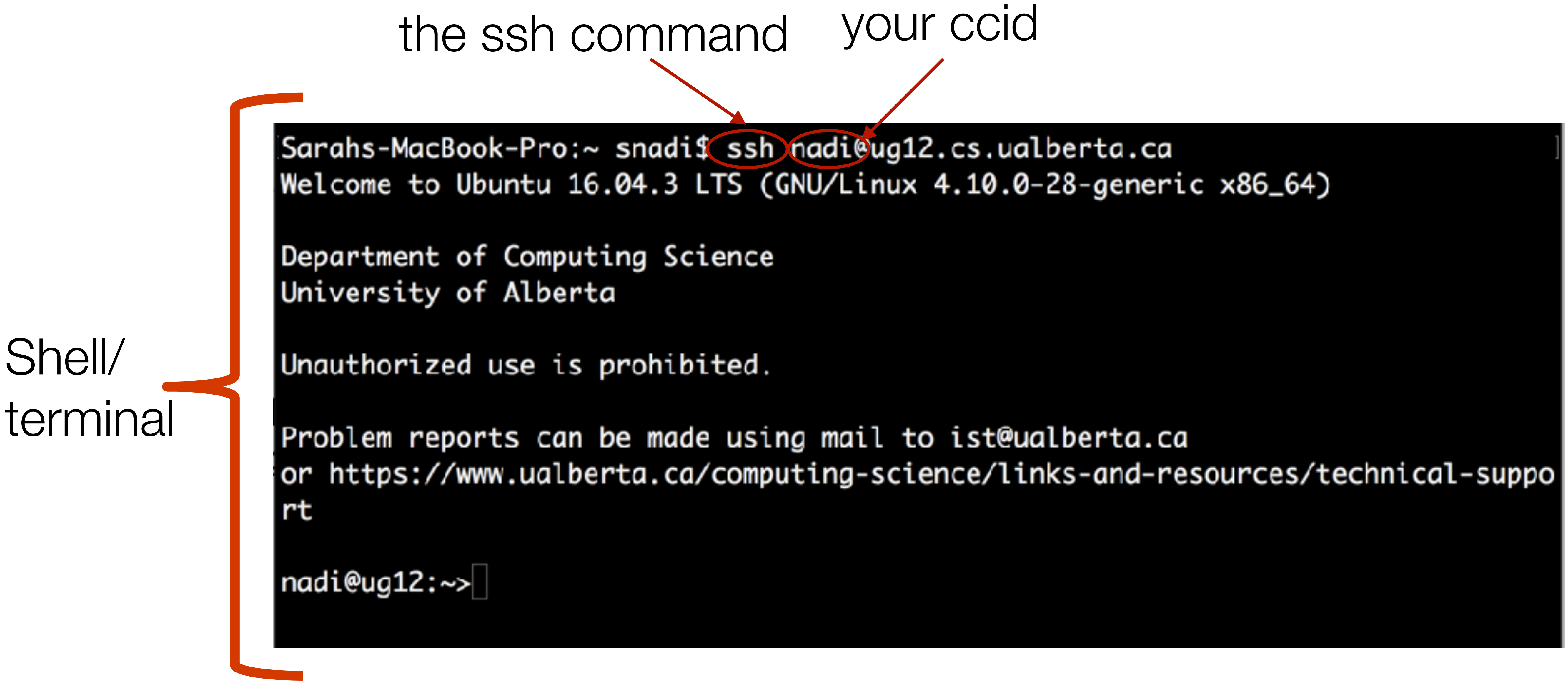
Example of Connecting to Lab Computer



```
Sarahs-MacBook-Pro:~ snadi$ ssh nadi@ug12.cs.ualberta.ca
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64)

Department of Computing Science
University of Alberta

Unauthorized use is prohibited.

Problem reports can be made using mail to ist@ualberta.ca
or https://www.ualberta.ca/computing-science/links-and-resources/technical-suppo
rt

nadi@ug12:~>
```

demo

Example of Connecting to Lab Computer



the ssh command

your ccid

Shell/ terminal

```
Sarahs-MacBook-Pro:~ snadi$ ssh nadi@ug12.cs.ualberta.ca
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64)

Department of Computing Science
University of Alberta

Unauthorized use is prohibited.

Problem reports can be made using mail to ist@ualberta.ca
or https://www.ualberta.ca/computing-science/links-and-resources/technical-suppo
rt

nadi@ug12:~>
```

demo

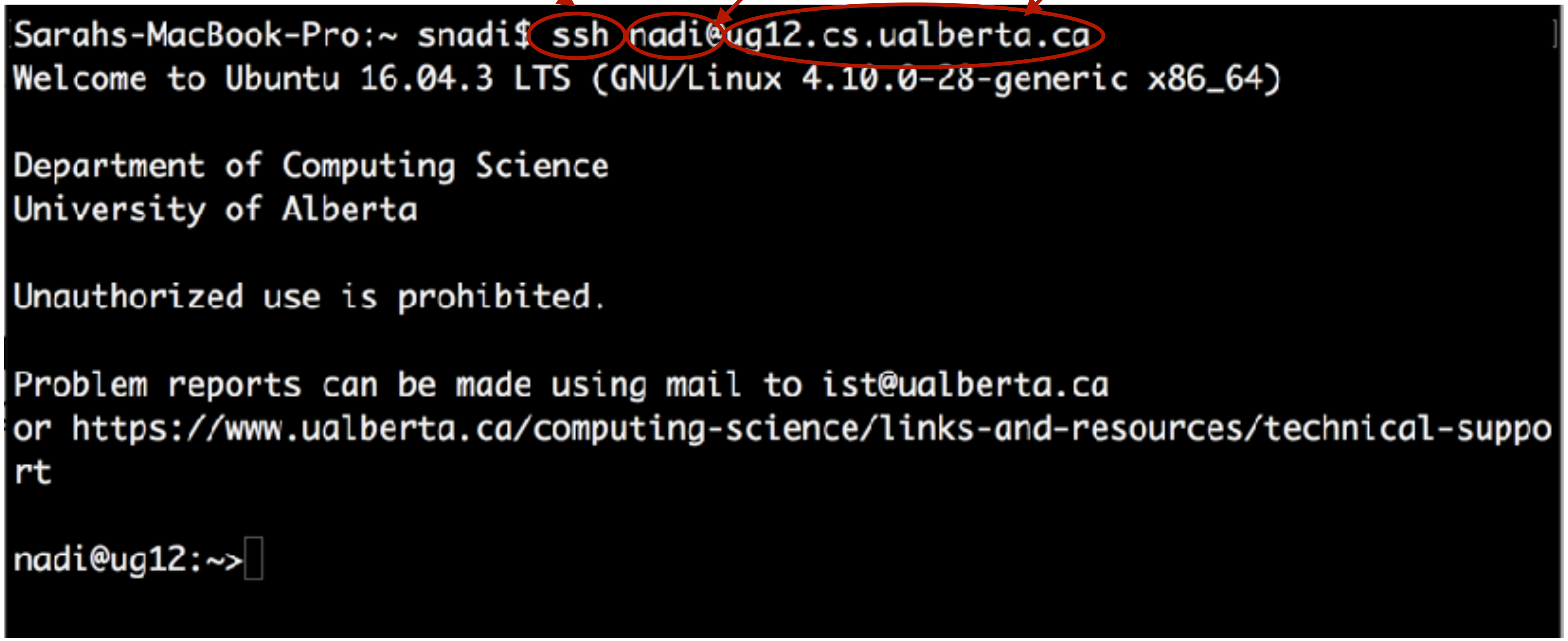
Example of Connecting to Lab Computer

the ssh command

your ccid

address of lab machine

Shell/ terminal



```
Sarahs-MacBook-Pro:~ snadi$ ssh nadi@ug12.cs.ualberta.ca
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64)

Department of Computing Science
University of Alberta

Unauthorized use is prohibited.

Problem reports can be made using mail to ist@ualberta.ca
or https://www.ualberta.ca/computing-science/links-and-resources/technical-suppo
rt

nadi@ug12:~>
```

demo

Shell Commands

- `ls, mkdir, cd, cat, vi, gcc, cp, rm, mv,` …
- `man` (manual pages are extremely useful)
- We will cover these commands throughout the course, especially in the Labs.
- You should also get used to searching for certain commands yourself.

demo

Sample C program:

```
#include <stdio.h>

int main(void) {
  int classNumber;

  printf("What is your class number? ");
  scanf("%d", &classNumber);


  printf("Hello CMPUT %d!\n”, classNumber);
  return 0;
}
```

Sample C program:

Only few things are included by default in C. You need to include the libraries you will use

```
#include <stdio.h>

int main(void) {
   int classNumber;

   printf("What is your class number? ");
   scanf("%d", &classNumber);


   printf("Hello CMPUT %d!\n”, classNumber);
   return 0;
}
```

Sample C program:

Only few things are included by default in C. You need to include the libraries you will use

```
#include <stdio.h>

int main(void) {
  int classNumber;

  printf("What is your class number? ");
  scanf("%d", &classNumber);


  printf("Hello CMPUT %d!\n”, classNumber);
  return 0;
}
```

the main function is the main entry point to your program

Sample C program:

Only few things are included by default in C. You need to include the libraries you will use

the main function is the main entry point to your program

this is a variable that can hold only integer values

```
#include <stdio.h>

int main(void) {
  int classNumber;

  printf("What is your class number? ");
  scanf("%d", &classNumber);


  printf("Hello CMPUT %d!\n”, classNumber);
  return 0;
}
```

Sample C program:

Only few things are included by default in C. You need to include the libraries you will use

```
#include <stdio.h>

int main(void) {
  int classNumber;

  printf("What is your class number? ");
  scanf("%d", &classNumber);


  printf("Hello CMPUT %d!\n”, classNumber);
  return 0;
}
```

the main function is the main entry point to your program

this is a variable that can hold only integer values

scanf is a library function that reads keyboard input

Sample C program:

```c
#include <stdio.h>

int main(void) {
   int classNumber;

   printf("What is your class number? ");
   scanf("%d", &classNumber);


   printf("Hello CMPUT %d!\n”, classNumber);
   return 0;
}
```

Only few things are included by default in C. You need to include the libraries you will use

the main function is the main entry point to your program

this is a variable that can hold only integer values

scanf is a library function that reads keyboard input

printf is a library function that prints output to the terminal

Sample C program:

Only few things are included by default in C. You need to include the libraries you will use

```
#include <stdio.h>

int main(void) {
   int classNumber;

   printf("What is your class number? ");
   scanf("%d", &classNumber);


   printf("Hello CMPUT %d!\n”, classNumber);
   return 0;
}
```

the main function is the main entry point to your program

this is a variable that can hold only integer values

scanf is a library function that reads keyboard input

printf is a library function that prints output to the terminal

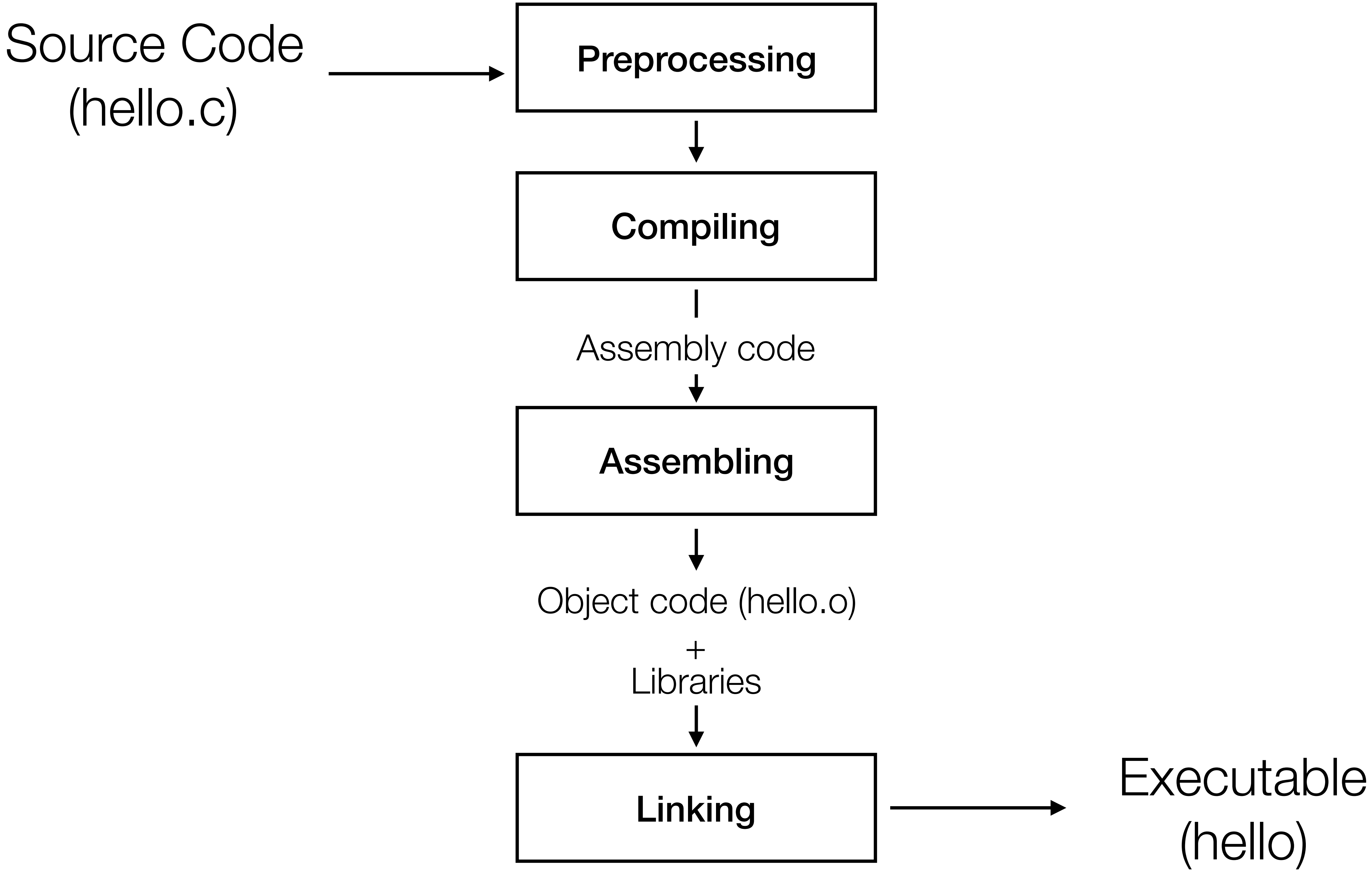
main must return 0 if it is successful

Compiling a C Program

- Wait, what does “compile” mean?
  - C is a compiled language, different from Python which is an interpreted language
  - With python, you used an interpreter that translates and executes one statement at a time. It would stop at the first error it meets.
  - With C, you use a compiler that translates the whole program into object code. After linking this object code, the program can be executed.
- We will use `gcc` with the following options to compile:
  - `gcc -Wall -std=c99 hello.c -o hello`
  - `-o` specifies the executable name, otherwise it’s a.out
  - `-Wall` enables all warnings and `-std=c99` specifies the C standard we will compile with

demo

Compiling a C Program (Behind the Scenes)



General Form of a C Program

```
/* directives */

/* global variables */

int main(void) {

 /* statements */

}
```

- Directives start with #
- Examples include:

```
// headers
#include <stdio.h>

// macros
#define PI 3.14
```

General Form of a C Program

```
/* directives */

/* global variables */

int main(void) {

 /* statements */

}
```

- Global variables are shared variables and can be accessed by any function in the file
- Example:
  `int counter = 1;`

General Form of a C Program

```
/* directives */

/* global variables */

int main(void) {

 /* statements */

}
```

- Your program must have a main function
- This is the entry point to your program, i.e., the main function starts your program
- Upon successful completion, your main function should return a value of 0. This is called the status code.
- For now, our main function will not take any parameters (hence the void between parentheses)

General Form of a C Program

```
/* directives */

/* global variables */

int main(void) {

 /* statements */

}
```

- A function consists of a list of statements
- Examples:

```
/* declarations */
int classNumber, numberOfStudents;

/* assignments */
numberOfStudents = 30;

/* function calls */
scanf(“%d”, &classNumber)

/* function terminates, and
returns a value */
return 0;
```

General Form of a C Program

```
/* directives */

/* global variables */

int main(void) {

 /* statements */

}
```

- A function consists of a list of statements
- Examples:

```
/* declarations */
int classNumber, numberOfStudents;

/* assignments */
numberOfStudents = 30;

/* function calls */
scanf(“%d”, &classNumber)

/* function terminates, and
returns a value */
return 0;
```

In C, every variable must have a type that indicates the kind of data it will hold

Documenting & Formatting

- Use indentation to construct blocks
- Use blank lines to separate logical blocks of code
- Use /* … */ or // to add comments that explain the semantics of your code

```
/*****************************************/
/* Converts a Fahrenheit temperature to Celsius  */
/* Name: celsius.c                                */
/* Author: K. N. King                             */
/* January 7, 2016 **************************/
```

Identifiers

- names for variables, functions, macros etc.
- must start with a letter or underscore
- case sensitive
- some keywords, such as `int` or `union` are reserved and cannot be used as identifiers (see Table 2.1 on page 26)
- try to use names that are self-explanatory and descriptive of the purpose of the variable, function, macro etc.

Version Control Systems

Version Control Systems

- record changes to a file over time
- allow you to keep track of the changes to your code and “go back” in time when you need to
- git is a widely used modern version control system
- Github (www.github.com) is an online project-hosting website that supports git
- a git cheat sheet is available at https://education.github.com/git-cheat-sheet-education.pdf
- there are LOTS of online resources on how to use git. Some of these are shared on eClass.

How will we use Git & Github in 201?

- You will have four **private** Github repositories: one for all lab exercises and then one for each assignment. All your lab and assignment solutions will be developed using Github.

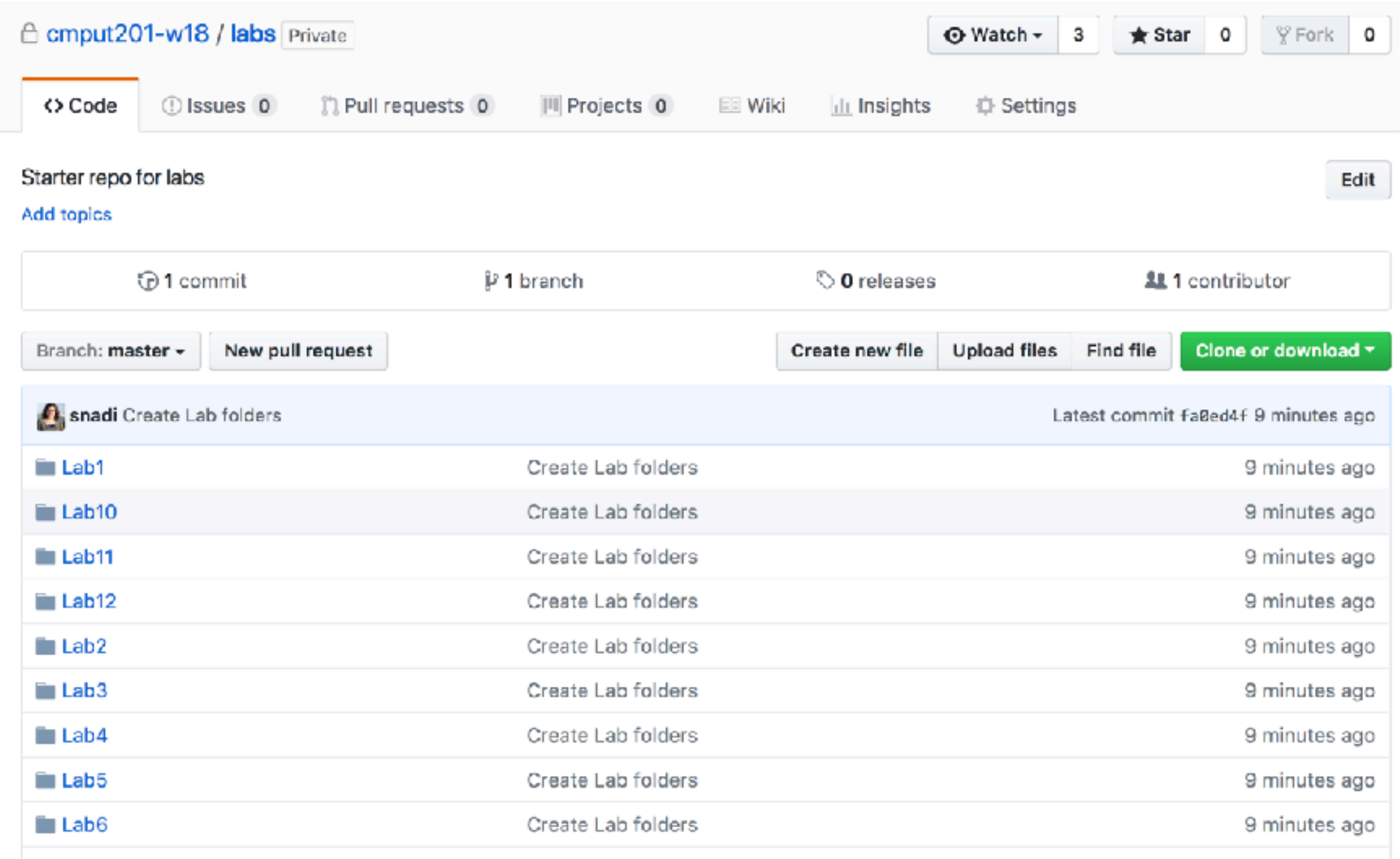
- You will receive a GitHub classroom invitation link for each of these four repositories.

- When you go to the link, GitHub will create the corresponding repository for you. You **must** use these repositories for labs and assignments. (Note that the first time you click on any of the links we provide you, you will be asked to select your CCID).

- When marking your assignments, we will simply pull the code on GitHub **at the time of deadline** and mark it. We will pull from the **master** branch.

- When marking lab exercises, the TA will pull the code from GitHub when you are ready to demo.

What Your Labs Repository Will Look Like



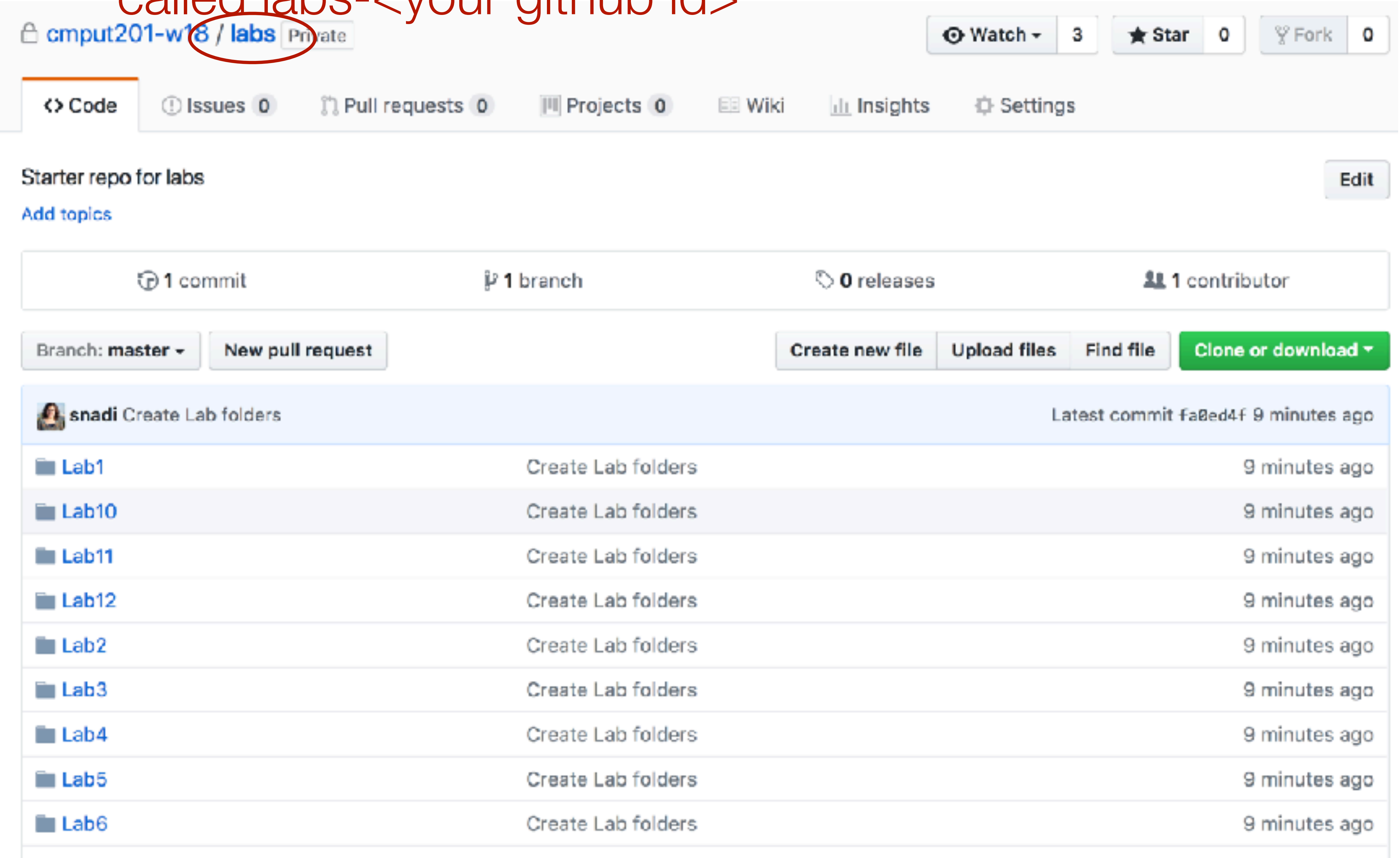
Starter repo for labs

snadi Create Lab folders — Latest commit fa8ed4f 9 minutes ago

Lab1	Create Lab folders	9 minutes ago
Lab10	Create Lab folders	9 minutes ago
Lab11	Create Lab folders	9 minutes ago
Lab12	Create Lab folders	9 minutes ago
Lab2	Create Lab folders	9 minutes ago
Lab3	Create Lab folders	9 minutes ago
Lab4	Create Lab folders	9 minutes ago
Lab5	Create Lab folders	9 minutes ago
Lab6	Create Lab folders	9 minutes ago

What Your Labs Repository Will Look Like

Your lab repo will be a copy of this and will be called labs-<your github id>



cmput201-w18 / labs Private

Watch 3 | Star 0 | Fork 0

<> Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Starter repo for labs

Add topics

Edit

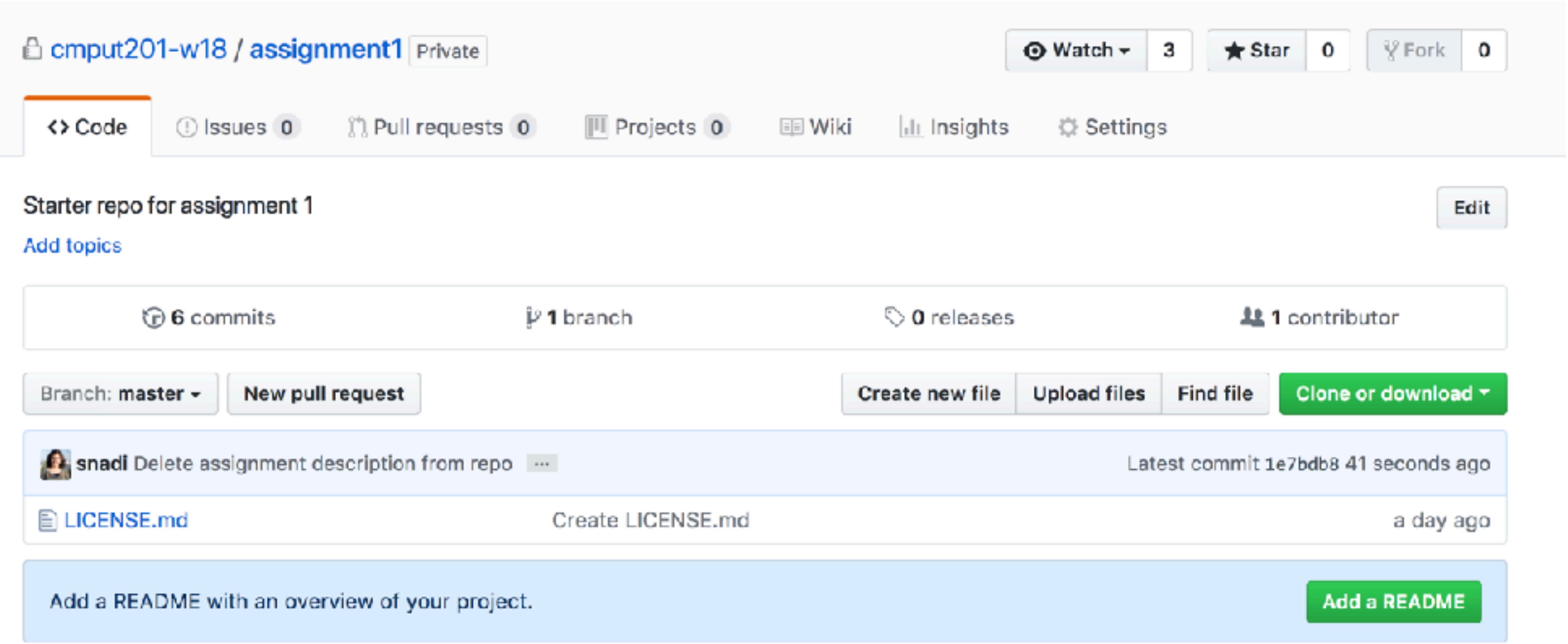
1 commit | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

snadi Create Lab folders — Latest commit fa8ed4f 9 minutes ago

Lab1	Create Lab folders	9 minutes ago
Lab10	Create Lab folders	9 minutes ago
Lab11	Create Lab folders	9 minutes ago
Lab12	Create Lab folders	9 minutes ago
Lab2	Create Lab folders	9 minutes ago
Lab3	Create Lab folders	9 minutes ago
Lab4	Create Lab folders	9 minutes ago
Lab5	Create Lab folders	9 minutes ago
Lab6	Create Lab folders	9 minutes ago

What an Assignment Repository Looks Like



cmput201-w18 / assignment1 Private

Starter repo for assignment 1

6 commits | 1 branch | 0 releases | 1 contributor

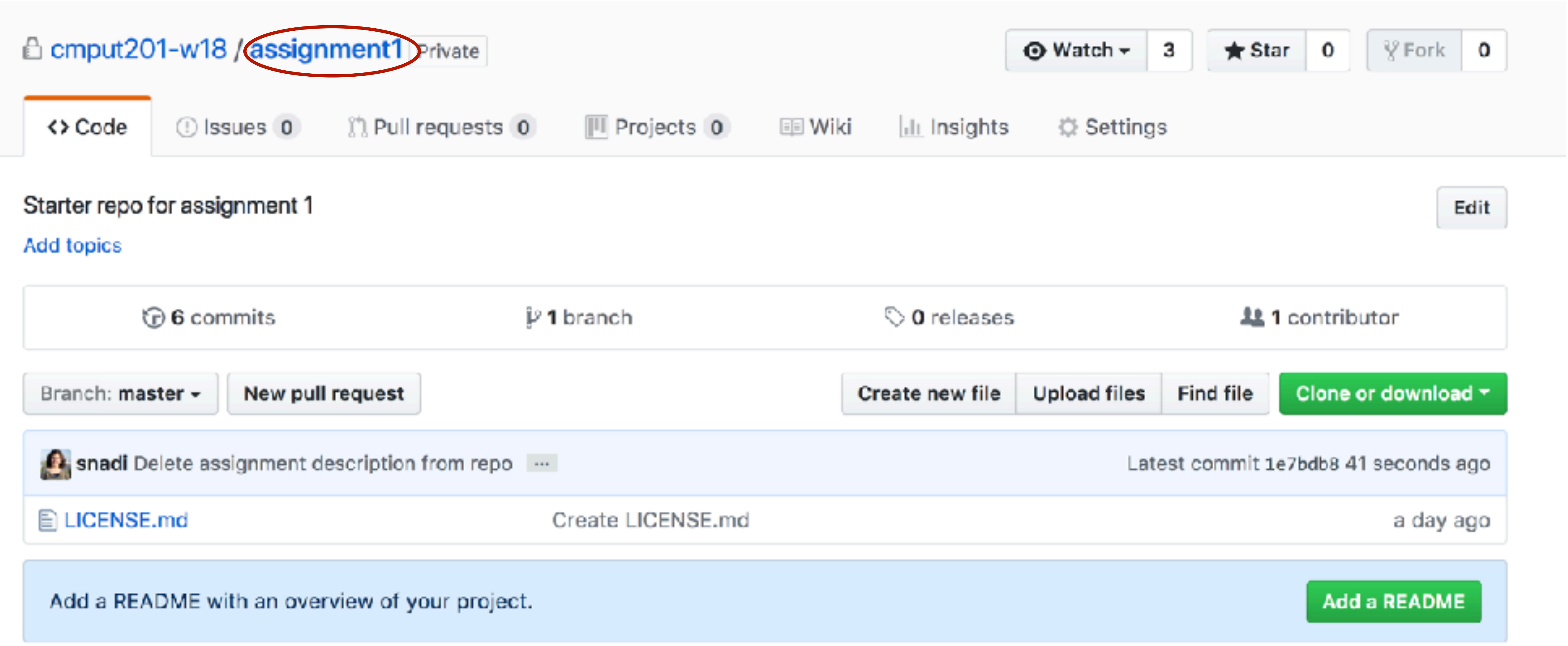
snadi Delete assignment description from repo ... Latest commit 1e7bdb8 41 seconds ago

LICENSE.md Create LICENSE.md a day ago

Add a README with an overview of your project.

What an Assignment Repository Looks Like

Your assignment1 repo will be a copy of this and will be called assignment1-<your github id>



cmput201-w18 / assignment1 Private

Starter repo for assignment 1

Add topics

6 commits 1 branch 0 releases 1 contributor

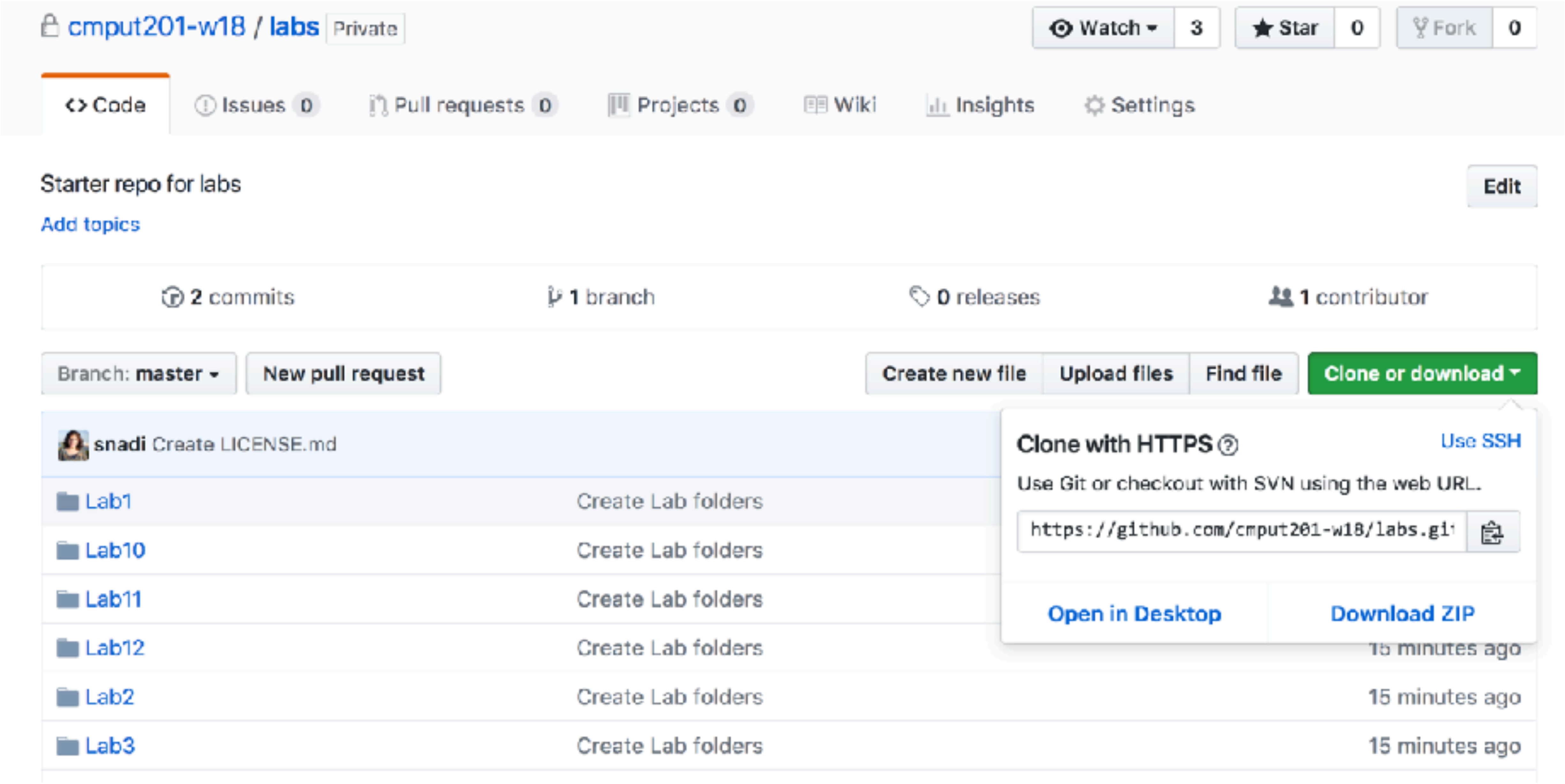
Branch: master

snadi Delete assignment description from repo ... Latest commit 1e7bdb8 41 seconds ago

LICENSE.md	Create LICENSE.md	a day ago

Add a README with an overview of your project.

Your Repository's Address



Starter repo for labs

snadi Create LICENSE.md

Lab1	Create Lab folders	
Lab10	Create Lab folders	
Lab11	Create Lab folders	
Lab12	Create Lab folders	15 minutes ago
Lab2	Create Lab folders	15 minutes ago
Lab3	Create Lab folders	15 minutes ago

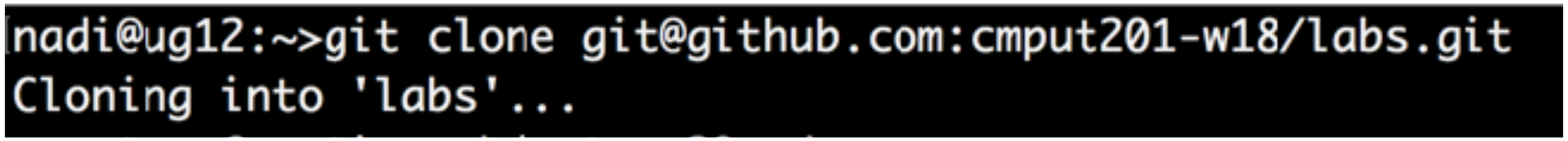
Clone with HTTPS

Use Git or checkout with SVN using the web URL.

```
https://github.com/cmput201-w18/labs.git
```

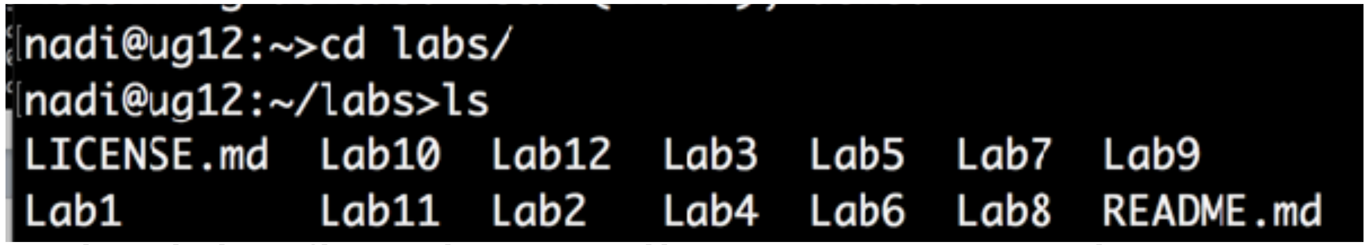
Example of Using git

- Clone repository (need to do only the first time you use the repo on a given computer):



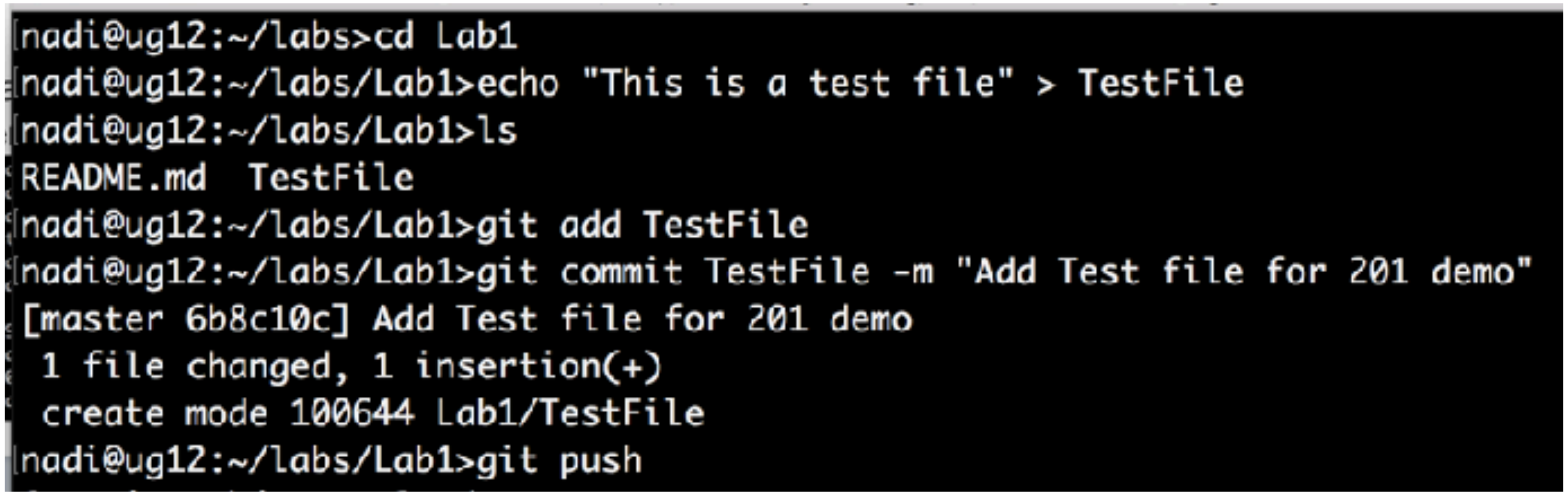
```
nadi@ug12:~>git clone git@github.com:cmput201-w18/labs.git
Cloning into 'labs'...
```

- Go to the directory that now contains your repository. You should see the same files you see on GitHub.



```
nadi@ug12:~>cd labs/
nadi@ug12:~/labs>ls
LICENSE.md  Lab10  Lab12  Lab3  Lab5  Lab7  Lab9
Lab1        Lab11  Lab2   Lab4  Lab6  Lab8  README.md
```

- Add/Modify a file and then commit this file, including a commit message. You then need to push this file to GitHub.



```
nadi@ug12:~/labs>cd Lab1
nadi@ug12:~/labs/Lab1>echo "This is a test file" > TestFile
nadi@ug12:~/labs/Lab1>ls
README.md  TestFile
nadi@ug12:~/labs/Lab1>git add TestFile
nadi@ug12:~/labs/Lab1>git commit TestFile -m "Add Test file for 201 demo"
[master 6b8c10c] Add Test file for 201 demo
 1 file changed, 1 insertion(+)
 create mode 100644 Lab1/TestFile
nadi@ug12:~/labs/Lab1>git push
```

demo

Your TODOs this week

- Before the labs start, get your UNIX (CS) accounts
- Try to log in before your first lab. Contact the HelpDesk if you can’t
- Create a Github account and submit it by midnight on Thursday through eClass following the instructions listed under the submission called Github user ids
- Go to the GitHub labs invitation link: https://classroom.github.com/a/UYMqsPms and link your GitHub account to your CCID
- Read Lab #1 tutorials and prepare for the lab
- Read all the general course information posted on eClass