

# Lab 0: Introduction

Welcome to your first lab! Labs (like this one) are opportunities for you to get hands-on experience with lecture material. Labs are graded on completion and you have a week to finish them. If you've finished the lab during this section, your TAs will check you off in the gradebook. If you're still working, you can finish the lab on your own time and get checked off during next week's section/lab.

## Welcome to The Sunlab

You will likely be doing some of the coursework for CS15 in this space, so we hope this first lab serves as a friendly introduction. The computers here run an operating system called [Linux](#) rather than Windows or Mac OS. It may feel a little unfamiliar at first, but Linux has many powerful features and you'll quickly get used to the differences.

This first lab will cover:

- Setting up your CS department account
- Learning some basic information about using Linux
- Learning how to open and use commonly-used programs

Whew! That's a lot to do in one lab. It's okay if you don't understand everything covered here—you'll have all semester (and beyond!) to perfect your Linux skills, so there's no need to worry about memorizing every command.

## Getting Started

- Bulleted lists have instructions for you to follow.
- For labs, code will be represented with **this font**.
- Don't worry about clicking on the "wrong" thing: you don't have access privileges to break anything. :)

## What We Assume

This lab assumes that you have basic familiarity with using computers — how to use a keyboard and mouse, a web browser, and navigation menus. If you have experience using Windows or Mac OS, this will all likely be familiar.

Please speak to a TA if you feel you do not have this level of background knowledge.

**If you have any questions during the lab, please raise your hand and a TA will stop by to help as soon as possible.**

One of the most powerful features of Linux is that it gives you extensive access to go “behind” the graphics and work more closely with what the computer is actually doing. One way of doing this is through the use of a “terminal.” A terminal is a window that allows you to interact with your computer directly through written commands. The terminal itself is actually a GUI for a “shell,” the program that executes those commands.

## Opening Terminal

- To open a terminal, click the Applications menu in the top left corner of Xfce, click System, then click Xfce Terminal. You can also open a terminal by right clicking anywhere and clicking “Open Terminal Here”.

Welcome to the Linux terminal! You’ll see a new window with something like:

```
cs1ab5f ~ $
```

displayed on the first line. The first part is the name of the computer, the “~” is an abbreviated file path (more on this soon), and the “\$” is called the “command prompt” - which *prompts* you to write *commands*.

## Shell Commands

Let’s get started! A command typed into a shell may have up to three parts, depending on the command and how it's used:

- The command name
- Modifications to the command (called "flags")
- Things to perform the command on or with (called "arguments")

Not all commands require all three parts, but they all must contain a name.

## Group Check

Before we get started using the terminal, let’s check to make sure that you are in the CS15 group.

- At the command prompt, type:

```
groups
```

And press “enter.” You should be able to see “cs-0150student” listed as one of your groups.

If not, call over a TA, who will get a consultant with systems privileges (SPOC) to add you to the student group.

## pwd & A Quick Introduction to the Linux File System

Let's take a look at a sample command: **pwd**, which stands for “print working directory.” Like the Windows Explorer and Mac Finder, Linux terminals have the concept of a “current location” when browsing the file system. Like the file system on Windows and Mac, files on Linux are organized hierarchically, with directories (also known as folders) located one within another.

On Linux, all files and folders are contained within a "root" folder. This "root" directory is just called `/`. You'll be able to access all these files in the root folder no matter which computer you're logged into (as long as it's a department machine).

The "path" to a file or directory (i.e. its location in the file system) can be specified as either absolute or relative. An absolute path leaves no ambiguity about its location, and is defined with the root directory `/` as the reference point. A relative path, however, is defined from the current directory as a reference point. **It's easiest to remember that an absolute path starts with a `/`, while a relative path does not.**

All subdirectories are also separated by a `/`, so the absolute path `/stuff/things/my_thing.txt` would represent a text file called `my_thing.txt`, which is contained in the directory `things`, contained in a directory `stuff`, contained within the root directory `/`. If your shell was already located within `stuff`, the relative path `things/my_thing.txt` would have the same meaning.

Two directories on the department system that you will use often are:

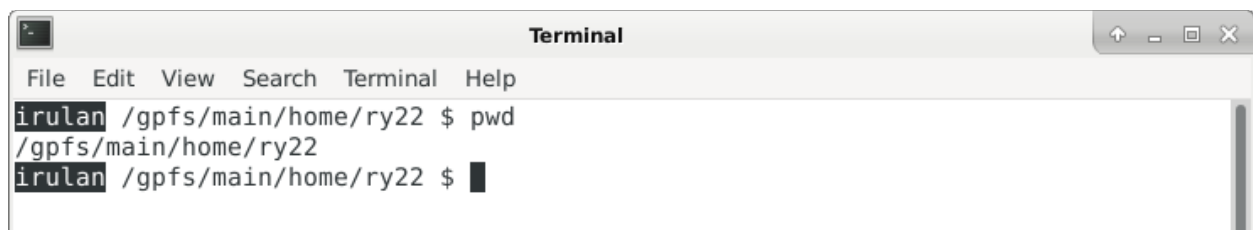
- `/home/your_login/` (for example, `/home/avd`): Everyone with a department account owns a "home" directory within `/home/`. This and its subdirectories are where you should store your own coursework and files. Only you should have access to your home directory.
- `home/your_login/course/cs0150/`: This is a directory where your CS15 code is stored. (If you don't see it yet, our install scripts later on will take care of that for you!)

The location of the current working directory can be found by using the command `pwd`, which stands for "print working directory".

- At the command prompt, type:

`pwd`

And press "enter."

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows a user named "irulan" at a prompt in the directory "/gpfs/main/home/ry22". The user enters the command "pwd", and the terminal outputs the full path "/gpfs/main/home/ry22". The prompt then returns to the user in the same directory.

```
irulan /gpfs/main/home/ry22 $ pwd
/gpfs/main/home/ry22
irulan /gpfs/main/home/ry22 $
```

By default, new shells will start in your home directory, which is what the results of the command show. (Note that you can also see the working directory to the left of the command prompt. The home directory is often abbreviated with the tilde character, `~`.)

# An Exploration of Shell Commands

Let's try out some of the most commonly used terminal commands. Note: press enter after each command to enter it.

## Navigating Directories

In this section you will navigate through directories on your file system, create files and folders, and delete them again.

- At the command prompt, type:

```
ls
```

The **ls** command *lists* the contents of a directory. If you specify a directory as an argument (e.g. **ls Desktop**), it will list the contents of that one, but otherwise it will show the contents of the current (or working) directory.

The list includes both files and directories (which you can distinguish by the “/” at the end of the name, for example, **Desktop/**).

- Type:

```
cd Desktop
```

The **cd** command stands for *change directory*. Your shell's working directory is now **~/Desktop/** (which is the same as **/gpfs/main/home/<login>/Desktop/**). You should see this change reflected to the left of the command prompt.

**Note:** Linux is a case-sensitive operating system. When you run a terminal command, make sure that you are using the correct case for your filename. Linux will believe that “Desktop” and “desktop” are two different folders, and if “desktop” doesn't exist, you'll encounter errors trying to access it.

## Shortcuts for Directories: “.” and “..”

Linux also provides a shortcut for you to refer to the current directory and its parent directory when working with relative paths. The symbol **.** refers to whatever the current directory you're working in is, and **..** refers to its parent.

- Type:

```
cd ..
```

This will bring you one level up, so you should be back at `/home/<login>/`.

- Now, type:

```
cd Desktop
```

to go back to the Desktop (where we'll make some files in the next section!)

## Making Directories & Files

- Type:

```
mkdir NewDirectory
```

The `mkdir` command is short for *make directory*. It takes the desired name of the directory as an argument.

Minimize any screens you have open to take a look at your Desktop. You should see an icon for your **NewDirectory** there.

- Type:

```
cd NewDirectory
```

which navigates into your new directory. Then type:

```
ls
```

to view the contents of the directory. Soon this combination of entering a directory and listing its contents will become almost second nature! In this case, nothing should show up, since the directory you just made is empty.

- Type:

```
touch MyNewFile.java
```

which creates a new file (among other things). Then type:

```
ls
```

to see that the new file is now listed.

- Type:

```
rm MyNewFile.java
```

to delete your new file. Type `y` to confirm the removal of the file.

- Type:

```
cd ..
```

to change out of “**NewDirectory**” back to your **home** directory.

- Type:

```
rmdir NewDirectory
```

To remove the now-empty directory you’ve made.

- Type:

```
ls
```

- Your files and shell are now in the same state that we started off in.

## Running Programs

Let’s move on to running programs. From the Linux shell, you can launch familiar applications like web browsers and word processors. Many of these can be launched through the Applications menu, but it’s often more convenient to use a terminal.

## The World Wide Web

- At the command prompt, type:

```
chrome &
```

And press “enter.” Notice how the web browser opens.

## Foreground/Background Processes

By default, when you run a program from the command line — either a simple UNIX utility like **ls** or a full application like Chrome — the program runs in the *foreground*, meaning the Terminal will wait for the *process* to finish running before a new process can begin. For a very short process, waiting could be ok, but when you open an application like a web browser, you’re probably going to want to use it for a substantial amount of time.

In order to continue to use the terminal while you keep your program open, you can instruct the shell to open the program in the *background* by adding an **&** symbol to the end of the command,

like you did above. Having the program run in the background means that the Terminal will free itself up to accept new commands.

In general, if you're opening an application, you'll probably want it might be advantageous to run it in the background with `&`.

## Closing the Terminal

**Important:** When you close a terminal window, all the applications you opened from that terminal window will close without prompting you to save your work. Closing the terminal *kills* the Terminal and everything running out of it.

## Writing Java Programs

When you write Java programs for CS15 (or any other program for any language you may encounter in the future), you can use any text editor you want—you don't need any special software (although software does exist that will make the task of writing large programs significantly easier, and these programs will be introduced later in the course). You just need working Java code (which you will learn to write) and a way to *compile* your code. There is nothing “magical” about the text editors you will use in CS15, **except** that there are a lot of helpful programming tools built on top the text editors you will be using.

The text editor that we will use for now is Atom. Atom is just one of many editors that is available to you.

## Opening Atom

- Type `atom &` to open the text editor (it may take a couple seconds).

**Note:** If you want to open a specific folder or file in Atom, you can go to menu bar and click File > Open Folder, or File > Open File, and navigate to the folder or file that you want to access.

## File Extensions

In Linux, file extensions (the part of the filename after the “.”, e.g. `.java`, `.txt`, `.docx`) are completely optional, and serve no purpose other than helpfulness for the user. For example, by convention text files that contain Java code end in `.java` (e.g. `MyJavaFile.java`). While this extension makes it easier for a user to identify the files, it makes no difference to a Linux computer—you can still compile and run Java code stored in files with no extensions (although this is highly discouraged).

One nice thing about using extensions, though, is that if you want to refer to all the files of a particular type, you can use the terminal command `*`, which is another Linux shorthand called a “wildcard” and stands for any number of characters. For example, to open all of the Java files in a particular directory, you can refer to them all using the command `*.java`.



So if you typed:

```
atom *.txt &
```

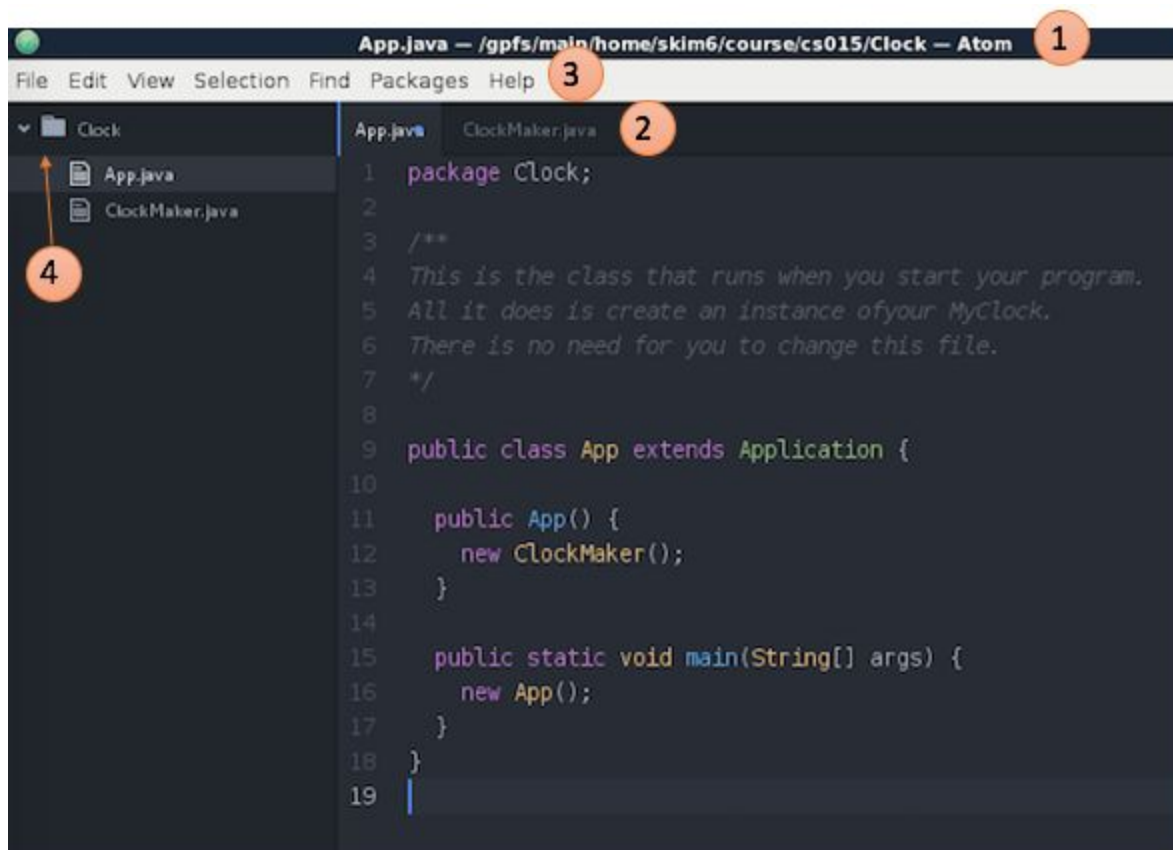
Atom would open all the files in the current directory that ended in ".txt". Likewise,

```
atom My* &
```

 would open all files in the current directory that started with "My".


## Using Atom

Here is what Atom looks like with two open Java files (don't worry about understanding the actual code):



The parts of the interface are as follows:

- 1) The working directory. Looks like `pwd`, right?

- 2) The open file tabs. Much like in a web browser, you can use these tabs to switch between files you are actively editing. The  in the first tab shows that the document has unsaved changes, whereas the other tab has no unsaved changes.
- 3) The menus across the top of the window have basic commands like "Open," "Save," "New File," etc.
- 4) This side menu shows the working directory you have open as well as the files in that folder.

Atom also has *syntax highlighting*, which means that different keywords are automatically displayed in different colors, depending on their meaning. This makes it easier to read code.

An example of syntax highlighting:

In a normal text editor:	In Atom:
<code>public static void main(String[] args){</code>	<code>public static void main(<i>String</i>[] args){</code>

Unfortunately, you can't really play with these features until you have some code to write. In the meantime, rest assured that Atom is tons o' fun.

## Writing, Compiling and Running Java Programs

Whenever you write a Java application, you start by editing Java files, which are saved with the extension `.java`. You compile your code by running those files through the *javac compiler*, which translates your code into a format the computer can execute. Then, your program is ready to run. You'll learn more about the structure of a Java application as the semester progresses. For now, let's dive into an introductory example:

## Hello World!

- Type `cd` to return to your home directory, if you're not already there.
- Run `cs0150_install lab0`. This will create a directory for lab0 in your cs0150 directory.
- Go to your `lab0` directory using `cd course/cs0150/lab0`.
- Type `touch HelloWorld.java` to create a new file.
- Run `atom HelloWorld.java &` to open the newly created file in Atom. Type out the following code into this new file:

```
package lab0;

public class HelloWorld {
```

```

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}

```

(Note that we recommend you re-type this code rather than copy/pasting it—oftentimes, text is encoded differently in the lab PDFs than it should be in a text editor, and this can wreak havoc with your Java compiler.)

The **System.out.println()** command prints out text into your terminal. This is a great way of confirming your program has reached a certain point in the code, and will become one of your best friends throughout the course.

- Save the file, then compile your code by typing **javac \*.java** in your terminal
  - If your code compiled successfully, nothing will print in your terminal. A new file called **HelloWorld.class** will appear in your **lab0** directory.
  - If error messages did print out, make sure your document looks like this in atom:



- Then recompile your code by typing **javac \*.java** in your terminal
- After you compile your code successfully, run your code by typing **java lab0.HelloWorld** in your terminal
- You should now see the text “Hello World!” printed out in your terminal.

Congratulations! You just compiled and ran your first Java application!

In addition, please take the time to fill out the **CS15 initial questionnaire**, which will provide the course with valuable insight into the demographics of the students taking it. The questionnaire can be found here: <https://forms.gle/dF5vwCfPsyWyEt9Z9>

**Show a TA your code and the completed questionnaire, and they'll check you off! Remember to logout before you leave.**

To logout, click on the top right corner of the screen, then your name, then logout. You can also use the shortcut "Ctrl+Alt+Backspace" to log out quickly.