Lab 0: Introduction to UNIX

Table of Contents

Introduction

Hello! The first thing you probably noticed about these computers is that they don't have Windows or MacOS installed. And you're right — they're running UNIX (Ubuntu to be exact). But fear not! We'll get you familiar with this new system in no time!

This lab explains how to use the school computers for CS 61A labs and also teaches the basics of using a UNIX terminal. Lab 1 discusses how to use your own computer for CS 61A.

UNIX, Python, and text editors

In CS 61A, you will be interacting with a variety of programs:

- UNIX is an operating system, just like Windows and MacOS. UNIX supports a filesystem with which you can manage files and folders.
- Python is a language, just like English. Python is used to describe programs. (This lab will not talk about Python — you'll see plenty of it in the coming weeks.)
- Text editors are like Microsoft Word — these are the programs in which you will be writing Python, just like how you write English in Microsoft Word. We'll talk more about this in Lab 1.

Registering your account

Before we continue, make sure you have set up your class account by doing the following:

1. Logged in to your class account (`cs61a-??`, with `??` replaced by your unique two or three letters).

2. Opened a terminal (by typing `ctrl-alt-t`) and entered registration information. Make sure you enter your *berkeley.edu* email address.

   > If you made a typo (e.g. misspelled your name), don't worry. After completing the registration process, you can type `re-register` into your terminal and hit enter. This will restart the registration process.

3. Changed your password (by typing `ssh update` into the terminal).

   > When you are typing in your old and new passwords, no characters will show up in the terminal — this is a security feature, not a problem.

Please do not forget your login information. In particular, memorize

- Your login (`cs61a-??`, with `??` replaced by your unique two or three letters)
- Your password

If you forget your login at anytime during the class, you can ask your TA to find it for you. However, if you forget your password, you will have to email inst@eecs.berkeley.edu or go to 333 Soda.

Logging in from Home

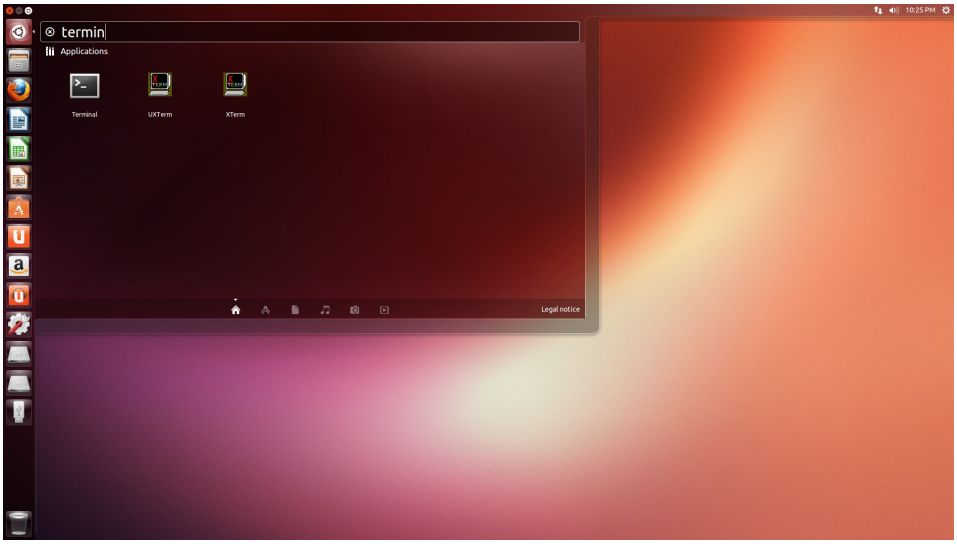
If you don't have access to a school computer for this lab, you can still try it out: refer to lab 1 for more details about setting up your home computer.

Meet the Terminal

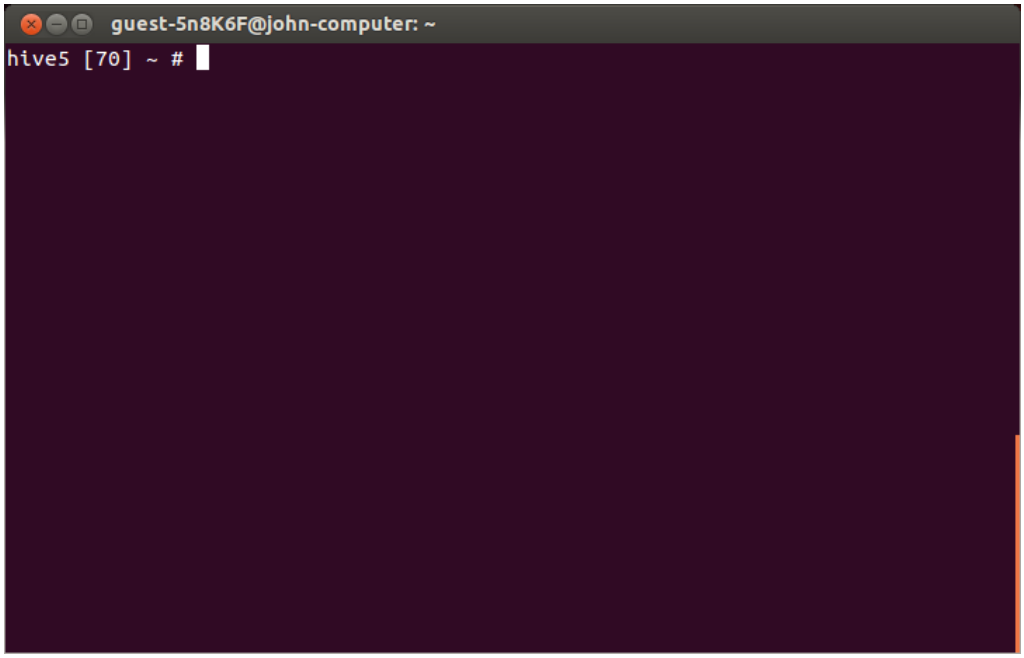
The terminal is a program that allows you to talk to your computer by entering commands. No matter what operating system you use (Windows, MacOS, Linux), the terminal will be an essential tool for CS 61A.

On a school computer, you can start the terminal in various ways:

- Type `ctrl-alt-t`, or
- Click on the launcher in the top left corner and type "terminal":



Both methods will open the terminal, which looks like this:



In the image above, the *prompt* says `hive5`. Depending on which school computer you are using, your prompt might say something different (like `s271-01`).

The terminal lets you give commands to your computer. Try this command:

```
echo "Hello world"
```

Your terminal will repeat "Hello world" back to the screen. The `echo` command just tells your terminal to repeat the words you typed. Not too impressive just yet, but it turns out the terminal can do a lot more!

The Filesystem

On Windows or MacOS, you are probably familiar with folders and files, with which you can interact by dragging and dropping icons. Today, we are getting rid of the icons and using just the terminal to manage our files and folders!

> Note: most school computers support graphical interfaces for file management, much like Windows and MacOS. If you really, really, *really* don't want to use the terminal for file management, that's okay. However, learning to use the terminal pays off!

With the terminal, you can do everything that a graphical filesystem can do — and more!

Directories

The first command we'll use is `ls` (the letter `l` and the letter `s`). Try typing it into the terminal!

```
hive5 [70] ~ # ls
Desktop/  Documents/  Downloads/   ...
```

> Depending on what computer you are using, the output that you see after typing `ls` might be different.

The `ls` command lists all the files and folders in the current directory. A directory is another name for a folder (such as the `Documents` folder).

When you open a terminal, you will start from the home directory. Notice that your prompt (`hive5 [70] ~ #`) has a tilde `~` in it. Your prompt helpfully tells you your current directory — in this case, your current directory is `~`, which stands for the home directory.

Making New Directories

Our next command is called `mkdir`. Try typing the following command into your terminal:

```
mkdir lab0
```

The `mkdir` command makes a new directory (i.e. makes a new folder). Notice that, unlike `ls`, we don't just type `mkdir` and press enter. We also need to specify an argument to the command (the argument is `lab0` in this example). For `mkdir`, the argument is the name of the directory we want to create.

> Some commands always require arguments to work, like `mkdir`. Other commands can work just fine without supplying any arguments, like `ls`.

Now that we've made our `lab0` directory, let's make sure it is actually there. Use the `ls` command to verify that `lab0` shows up in our list of directories.

Moving to other directories

To move into another directory, we use the `cd` command. Try typing the following command into your terminal:

```
cd lab0
```

The `cd` command will change directories — in other words, it moves you into the specified folder. In the example above, we chose to move into the `lab0` directory.

Notice that the `~` in your prompt turned into `~/lab0`. Again, the prompt will tell you what your current directory is. In this case, we are in the `lab0` directory, which is located within the home directory (the `~`).

If you use the `ls` command now, you'll notice that no output shows up. This makes sense, since we just created the `lab0` directory and we haven't added any files to it. We'll come back to this later.

For now, let's get back to our home directory. There are a few ways to do this:

- Type `cd ..` (two dots). The `..` means "the parent directory". In this case, the parent directory of `lab0` happens to be our home directory, so we can use `cd ..` to go up one directory.
- Type `cd ~` (the tilde). Remember that `~` means home directory, so this command tells your terminal to change to the home directory, no matter where you currently are.
- Type `cd` (that is, the `cd` command with no arguments). In UNIX, typing just `cd` is a shortcut for typing `cd ~`.

Removing Directories

We now know how to see, create, and move to directories. Our last command involving directories will be to delete them using the `rm` command.

First, let's create a temporary directory:

```
mkdir tmp
```

If you use the `ls` command, you should now see `tmp` listed as a directory.

Next, let's delete the `tmp` directory:

```
rm -r tmp
```

The `rm` command will remove files and directories from your filesystem. By itself (that is, without the `-r`) the `rm` command only removes files. However, since we are removing a directory, we need to specify `-r` to recursively remove the `tmp` directory and any files that `tmp` might contain (the process is called "recursive" because, in order to remove `tmp`, we have to remove everything inside of `tmp`).

> As you've seen, some commands require arguments, like `mkdir`. Other commands do not require any arguments in order to work, like `ls`. In addition, most commands can also be given flags, like the `-r` for `rm`. Flags are ways to specify modified behavior for commands — for example, `rm` by itself only removes files; using `-r` tells `rm` to remove directories.

Summary of directories

So far, we have learned how to do the following with directories (folders):

- `ls`: list the files and folders inside of the current directory
- `mkdir`: make a new directory
- `cd`: *c*hange *d*irectories
- `rm -r`: remove a specified directory

Files

Directories are not very useful if they don't contain any files. In this section, we walk through some more commands that allow you to interact with files.

For this section, let's start back in our home directory. Recall that we can do this by simply typing

```
cd
```

into our terminal. Your prompt should now say `~`.

Creating files

There are many different ways to create files. For this class, you will usually be using a text editor to directly write the file, much like how you would edit a Word document in Microsoft Word. We'll talk more about text editors in Lab 1.

For now, we'll just download a file called `unix.txt`, which can be found here. In this class, you will start most homeworks and projects by downloading a file.

The default location for downloads on the school computers is in the `Downloads` directory. Let's change into that directory using our `cd` command.

```
cd ~/Downloads
```

You can use the `ls` command to verify your `unix.txt` is in this directory.

Moving files

On Windows and Mac, much of your interaction with files is likely spent dragging them from folder to folder. UNIX provides a way to move files with the `mv` command.

Remember that we created a directory called `lab0`. Let's move `unix.txt` into `lab0`:

```
mv unix.txt ~/lab0
```

The `mv` command moves one file/directory into another file/directory. Here, we are moving the `unix.txt` file into the `lab0` directory, which is inside the home directory.

To verify that the `mv` command work, do the following:

1. Use `ls` to check that `unix.txt` is no longer in our current directory (which is the Downloads directory).
2. Change into the `lab0` directory. Your prompt should now show `~/lab0`.
3. Use `ls` to verify that `unix.txt` shows up in `lab0`.

Reading a file: the quick and easy way

Files are useful because they contain information. Let's see what `unix.txt` contains. Type in the following command:

```
cat unix.txt
```

This prints out a list of all the useful UNIX commands we've seen so far. The `cat` command prints the contents of a file to the screen. This is a fast way to verify that a file is correct or to read what a file contains. For example, if you forget any UNIX commands in today's lab, you can quickly `cat unix.txt` to read about them.

Renaming files

To rename files on Windows or MacOS, you would click on the name of the file and type in the new name.

Renaming files with the terminal can be a little confusing at first. Try the following command in the terminal (from the `lab0` directory)

```
mv unix.txt unix_commands.txt
```

Using `ls`, you'll see that `unix.txt` is gone — in its place is a file called `unix_commands.txt`. Furthermore, typing `cat unix_commands.txt` will print out the same list of UNIX commands.

It appears that we renamed `unix.txt` to `unix_commands.txt` by using the `mv` command! Here's how to think about it:

- `mv` will move the contents of a file/directory into another file/directory. In the previous section, we moved a file into a directory.
- This time, we are moving the contents of a file (`unix.txt`) into another file (`unix_commands.txt`). While we are *technically* moving file contents, this is *effectively* the same thing as renaming a file!

This can be a bit confusing if you're seeing it for the first time, so make sure you understand it before you move onto the next section.

> Note: Suppose you already have two files, `alice.txt` and `bob.txt`, and you issue the command:
>
> ```
> mv alice.txt bob.txt
> ```
>
> This will overwrite the old contents of `bob.txt` with the contents of `alice.txt`! UNIX won't warn you about overwriting, so be careful when using the `mv` command!

Copying files

Sometimes, it is useful to have multiple copies of a file. Try the following command:

```
cp unix_commands.txt lab0.txt
```

The `cp` command copies the contents of one file into another file. Using `ls`, you will see that the `lab0` directory now contains two files, `unix_commands.txt` and `lab0.txt`. Using `cat` will verify that both files have the same contents.

Suppose we also wanted to copy the `unix_commands.txt` file to our home directory. Here's one way to do it:

1. Change back to the home directory. (challenge: try doing this without looking up the command!)
2. Next, use the following command:

   ```
   cp lab0/unix_commands.txt .
   ```

   Don't forget the dot at the end!

The first argument (`lab0/unix_commands.txt`) tells the terminal to look in the `lab0` directory to find `unix_commands.txt`.

The second argument `.` tells the terminal to copy `unix_commands.txt` to the directory `.`. Just as two dots (`..`) represents the parent directory, a single dot (`.`) represents the *current* directory (the directory we're in right now).

Now that we're in the home directory, we can use `ls` to verify that there is a copy of `unix_commands.txt`:

```
hive5 [70] ~ # ls
Desktop/ ... unix_commands.txt ...
```

Using `cat unix_commands.txt` will show the same output of UNIX commands.

> Recap: we've seen two special directories: two dots `..` represents the parent directory (one directory up), while a single dot `.` represents the current directory. You can use these special expressions with any command that deals with directories. For example, you can `mv` a file to the current directory with the command
>
> ```
> mv some_file .
> ```

Removing files

Recall the `rm` command we introduced earlier. We originally used the `-r` flag to remove directories. Now we will use `rm` without the `-r` to remove a file. Type this into your terminal:

```
rm unix_commands.txt
```

This will delete the copy of `unix_commands.txt` that is in our current directory (which is the home directory). A quick `ls` will show you that `unix_commands.txt` is gone.

> Warning: Unlike on Windows and MacOS, there is no friendly Recycle Bin or Trash from which you can restore deleted files. In UNIX, when you `rm` a file, it's gone. You can't "undo" `rm`, so think twice (and thrice!) before using the `rm` command!.

Summary of files

In this section, we learned the basics of manipulating files:

- `cat`: displays the contents of a file on the screen
- `mv`: moves a file/directory to another file/directory. When moving one file to another, we are effectively renaming the file!
- `cp`: copies a file to another file/directory.
- `rm`: removes a file. When using the `-r` flag, `rm` will delete directories.

In addition, we learned about two special directories: `..` (the parent directory) and `.` (the current directory).

Getting Help

If you ever come across a terminal command with which you are unfamiliar, you can use a command called `man`:

```
man ls
```

The `man` command will show the manual pages (reference pages) for another command. In the example above, we ask the terminal to show the manual pages for the `ls` command. As you skim through the manual pages, you'll notice that `ls` can do a lot more than just list the contents of a directory! `man` is a great way to learn more about new commands and even commands that you think you already know.

> Note: Some school computers do not have the `man` command installed, so you might get an error. That's okay — if `man` ever fails, Google is your friend!

Conclusion

While the primary programming language in CS 61A is Python, it is important to know how to navigate through the UNIX filesystem to manage your class assignments. You will also be interacting with the Python interpreter from your terminal, whether you are using a school computer or your home computer.

In addition, if you continue with computer science after 61A, you will definitely interact more with UNIX and the terminal. For that reason, be sure to review today's lab if you have any questions!

> At this point, if you have finished the lab early, you can begin on Lab 1. Lab 1 walks you through how to pick a text editor, and how to write, test, and submit your first homework assignment for this class.

Appendix

Terminology

- Terminal: a program that allows users to enter commands to control the computer
- Prompt: displays certain information every time the terminal is ready to receive new commands. For example, your prompt might look something like this:

  ```
  hive5 [70] ~ #
  ```

  Usually, prompts will tell you your current directory (in the example above, the current directory is `~`)

- Directory: the same thing as a folder. Directories can contain files as well as other directories
- Parent directory: the directory that is immediately above the current directory (i.e. one directory up). This is represented in UNIX as two dots, `..`
- Current directory: the directory that we are currently looking at. This is represented in UNIX as a single dot, `.`
- Home directory: the top-level directory that contains all of your files and sub-directories. This is represented in UNIX as a tilde, `~`.

UNIX commands

- Directories:
  - `ls`: list the files and folders inside of the current directory
  - `mkdir`: make a new directory. For example, `mkdir lab0` creates a directory called `lab0`
  - `cd`: *c*hange *d*irectories. For example, `cd lab0` changes directories to `lab0`
  - `rm -r`: recursively remove a specified directory. For example, `rm -r lab0` removes the `lab0` directory and all files and subdirectories inside it.
- Files:
  - `cat`: displays the contents of a file on the screen. For example, `cat unix.txt` shows the contents of the file `unix.txt`
  - `mv`: moves a file/directory to another file/directory. For example, `mv file1 file2` moves the contents of `file1` into a (possibly new) file called `file2`. When moving one file to another, we are effectively renaming the file!
  - `cp`: copies a file to another file/directory. For example, `cp file1 file2` copies the contents of `file1` into a file named `file2`.
  - `rm`: removes a file. For example, `rm file1` deletes the file called `file1`.

- Miscellaneous

- Miscellaneous:
    - `echo`: displays words on the screen
    - `man`: displays manual pages for a specified command