Working with the Command Line

Useful Commands

At its heart, the command line is just another way to navigate the file system, like one might using the folder system (such as Finder on Macs). However, the command line is much more powerful than the folder navigator, and in CS 18, a good grasp of the command line will serve you well. First, we'll introduce a bit of vocabulary:

- Directory - the more official term for a folder.
- Executable - a file that can be executed, such as a compiled Java binary, or a Shell script.
- Permissions - who is allowed to open, modify, or execute files.

Wildcards

You can use wildcards with any of the commands we describe in this document. Basically, you can use a wildcard to refer to multiple files/directories at the same time. For example, suppose you're in a directory containing the following files:

```
X.java        y.java        z.out        a.txt        b.java      b.out
```

Then, if you type *.java, it refers to any file that ends in .java. Similarly, if you type b.*, it refers to any file that begins with b. Lastly, if I type *a*, it refers to any file containing an a. You'll find this invaluable when moving/copying/manipulating multiple similar files.

cd

cd stands for "change directory." You can use it to navigate between directories. For example, when you open the command line, you'll likely be located in your home directory, as indicated by the prompt:

```
<machine name>/gpfs/main/home/<your login> $
```

Suppose you want to navigate into your CS 18 workspace folder. You would cd into it, by naming the *path* to that folder, like this:

```
cd course/cs0180/workspace
```

This means "from the directory I'm in, enter the course directory, then cs0180 directory, then workspace, and stop there."

Now, you may be thinking, how do I go back to a parent directory? What if I'm in workspace, but I want to go back to the course directory?

Fear not! There are two methods to do this. The first is by using an *absolute path*. Earlier, the command indicated to start from the directory you were currently in. This is called a *relative path*, because the end location depends on which folder you started from! When I type this command from my own home directory, I end up in my workspace, but when you type it from your home directory, you end up in your workspace. When you navigate using an absolute path, everyone would end up in the same location, no matter what folder they typed the command from. Absolute paths begin with the / character. This stands for the root of all folders. For example, suppose I wanted to enter the CS 18 course directory:

```
cd /course/cs0180
```

This takes me to the course directory. Note that conversely, if you were in your home directory and typed

```
cd course/cs0180
```

You would be in your own personal folder, where your work lives. It's important to keep this in mind when navigating the filesystem.

So, one answer to "I'm in workspace, but want to go back to my cs0180 folder" is to type this:

```
cd /gpfs/main/home/<your login>/course/cs0180
```

Another is to use a shortcut. When you type “cd ..” it takes you up one level. For example:

```
/gpfs/main/home/afratila/course/cs0180/workspace $ cd ..
/gpfs/main/home/afratila/course/cs0180 $
```

The “..” refers to the parent directory of the directory that you are currently in!

One more trick you can use to navigate is the ~ shortcut. If you “cd ~” from any location in the file system, it will take you to your home directory:

```
/course/cs0180/src/lab01/src $ cd ~
/gpfs/main/home/<your login> $
```

Finally, cd also has a neat autocomplete feature. When you begin typing a filepath, you can use the tab key to autocomplete. If there is only one option for what you have typed so far, it will fill that in. If there are multiple options, press tab **twice** to display all of the options.

ls

ls is short for the word “list,” and what it does is list the contents of a directory. If you run it just by itself, it lists the contents of your current directory:

```
~/course/cs0180/workspace $ ls
javaproject/      scalaproject/
```

However, you can also give it a different directory by passing the filepath(s) of the directory/directories whose contents you’d like to list:

```
~/course $ ls cs0180/workspace
javaproject/      scalaproject/
~/course $ ls /course/cs0180/src
cs018codeformatter.xml  hw04/   lab03/  lab09/  scala_book@
finalexam/              hw05/   lab04/  lab10/  scalaformatter.properties
guizilla/               hw06/   lab05/  lab11/  search/
hw01/                   hw07/   lab06/  lab12/  showdown/
hw02/                   lab01/  lab07/  latex/  sparkzilla/
hw03/                   lab02/  lab08/  poems/
```

Notice that you can use either relative or absolute filepaths. In addition, you can also use multiple filepaths:

```
~/course/cs0180/workspace/javaproject $ ls sol src

sol:
show/  hw02/  hw04/  lab01/  lab03/  lab05/
hw01/  hw03/  hw05/  lab02/  lab04/

src:
show/  hw02/  hw04/  lab01/  lab03/  lab05/
hw01/  hw03/  hw05/  lab02/  lab04/
```

cp

cp is short for “copy,” and you can use it to copy files/directories to different locations while leaving the original unchanged. To copy a file:

```
cp <filename> <destination>
```

For example:

```
cp myFile ~/myWork
```

This would copy the file myFile and put it into the directory myWork, still named myFile. The original file myFile will remain undeleted in the original folder. As above, you can use either relative or absolute filepaths to specify both the file/directory to copy, and the new location to copy it to.

Sometimes, you’ll want to copy a file from somewhere to your current location. This is done like so:

```
cp <path to file> .
```

The period ‘.’ at the end indicates you want to copy myFile to exactly where you are. It is a shortcut meaning “the current directory,” similar to how .. is a shortcut meaning “one level up.”

However, you may want to copy a file to a new location and also rename it. To do so:

```
cp myFile ~/myWork/<newFilename>
```

This will copy myFile into the myWork directory, and rename it to newFilename.

Copying a directory requires you to pass in an extra argument, -r. This stands for recursive, and it tells cp to copy everything in the directory recursively:

`cp -r myDirectory ~`
This will copy myDirectory and all of its contents to your home directory.

You can copy it into your current directory with the same command from before:

```
cp -r myDirectory .
```

You can also rename it:

```
cp -r myDirectory ~/myNewDir
```

This will copy myDirectory into your home directory and rename it myNewDir.

A note-- when you want to rename files and directories like above, it is important for there to be no file or directory with the new name already existing in that directory. For example:

```
~ $ ls
course/     Desktop/    Documents/  Downloads/  Library/    misc/
~ $ ls misc
myFolder/
~ $ cp -r misc/myFolder Desktop     // this will copy myFolder inside of the
                                    // Desktop folder, NOT rename it
~ $ ls
course/     Desktop/    Documents/  Downloads/  Library/    misc/
~ $ ls Desktop
myFolder/
```

If the new name already exists, you are essentially performing a `cp <filename> <destination>` command.

mv

mv is short for “move,” and this allows you to move files/directories to a new location or rename them. Its usage is nearly identical to cp, except the *original copy* is moved (meaning that it is deleted in the original directory). For example:

```
~ $ ls
course/     Desktop/    Documents/  Downloads/  Library/    misc/
~ $ ls misc
myFolder/   myFile
~ $ mv misc/myFolder .
~ $ ls
```

```
course/     Desktop/    Documents/  Downloads/  Library/    misc/ myFolder/
~ $ ls misc
myFile
```
To rename a file, you simply mv it to the new name:

```
~ $ ls
Badfilename
~ $ mv Badfilename goodfilename
~ $ ls
goodfilename
```

Transferring Files Between A Local and Department Machine

The simplest way to do this is to first set up SSH, and then use SFTP. SSH stands for Secure SHell, and it's just a way to connect your computer to a computer in the CIT. SFTP stands for Secure File Transfer Protocol, and is a way to transfer your files back and forth between machines, using the connection established with SSH!

Here is the Brown CS guide to setting up SSH:

https://cs.brown.edu/about/system/connecting/ssh/

Once you have SSH set up, you can "SSH in" which means you enter this command in your local machine's terminal:

```
ssh <cs login>@ssh.cs.brown.edu
```

Now, you'll be in the department filesystem and can move stuff around and edit files! To exit SSH, you can simply type `exit`.

In order to transfer files between your machine and the department machine, you need to use SFTP instead of SSH. Luckily, SSH setup also sets up sftp for you! You use SFTP similarly to SSH (note that if you are SSH'd in, you should exit before trying to SFTP in):

```
sftp <cs login>@ssh.cs.brown.edu
```

Now, you'll be SSH'd into the computer, but with two helpful commands now at your disposal: `put` and `get`. If you want to transfer a file from your machine to the department, use `put`:

```
put <path to my file> <path to destination on dept machine>
```

If instead you'd like to copy a file from the department machine to yours, you use `get`:

```
get <path to file to copy> <path to copy it to on your machine>
```

You can easily substitute the second path for just a period "." and then use your computer's file explorer (like Finder) to move it to your desired location.

Just like with other commands, you can use shortcuts with sftp. For example, suppose I would like to copy over all my Java files from my hw01 sol folder:

```
> sftp afratila@ssh.cs.brown.edu
> ls ~/course/cs0180/workspace/javaproject/sol/hw01/sol
Animal.java        readme.txt       Giraffe.java      Armadillo.java
> get ~/course/cs0180/workspace/javaproject/sol/hw01/sol/*.java .
```

This would copy all my .java files to the directory I am in on my local machine, since * stands for anything and "." means the current directory. SFTP is a very helpful tool for working from home!

Running a Java Program

As you know from CS 17, a great way to get started developing code is to use the command line and an accompanying text editor. You will need this in CS 18 to test some of your projects, so here is an overview of the steps involved:

Writing Your Code

Either in Eclipse, or in a text editor, write your program. Make sure to save each file with the name of the class and the .java extension in the correct folder. Any code you write for an assignment should be located in `workspace/javaproject/sol/<assignment name>/sol`. For example, the class Squidward could be saved in the file Squidward.java, in the `workspace/javaproject/sol/lab01/sol` directory.

Compiling Your Program

Next, compile your program by navigating to the project directory, javaproject. Enter the following command:

```
> javac -d bin <package>/<filename>.java
```

Note: The > is our representation of the prompt; for all of these commands, you should type what comes after the > (i.e. `javac -d bin <package>/<filename>.java`).

The <package>/<filename>.java part refers to the path to the file you want to compile. For example, to compile Squidward.java, which resides in the `javaproject/sol/lab01/sol` subdirectory, you would enter '`javac -d bin sol/lab01/sol/Squidward.java`'.

Assuming your compilation succeeds, the Java compiler will create a '.class' file with the same name as the .java (source code) file it compiled, and will save that file in the corresponding location within the bin directory. In this example, Squidward.class would be saved in the `bin/lab01/sol` directory.

To simultaneously compile all files in a package, enter the following command:

```
> javac -d bin <package>/*.java
```

For example, to compile all of your lab01 files, enter '`javac -d bin sol/lab01/sol/*.java`'. As in the case of just one file, assuming your compilation succeeds, the Java compiler will create one .class file with the same name as each .java file it compiled, and it will save those files in the corresponding locations within the bin directory. You will need to use this functionality particularly when you want to run projects, which have multiple classes, in the command line.

If you want to compile files that reside in *both* the src and the sol directories, you can use multiple wildcards -- * -- to compile both at once. For example, '`javac -d bin */lab01/*/*.java`' will compile all java files in the src and sol folders for lab01. This will be handy when you need to run programs that have source code.

Note on Testing

If you want to compile and run code that uses our testing system (i.e., tester.Tester), the command line compilation will look a bit different. You will also need to include the tester.jar file while compiling, otherwise you will fail to compile your code. That looks like the following:

```
> javac -cp .:<path to jar 1>:<path to jar 2> -d ...
```

where the '...' represents the code you'd like to compile, and you do so in the same way as before by listing filenames or using wildcards.

Running Your Program

Finally, to run your program, navigate to the bin directory of your project and enter the following command:

```
> java <package>.<classname>
```

It is very important that the <classname> contains a main method. When you are running a project in the command line, make sure that you are running the file that contains your *main* method. (i.e. don't try to run one of your User classes in Pokerealties).

For example, to run your Squidward.java program (assuming it has a main method), you

would cd into the javaproject/bin directory, and type '`java lab01.sol.Squidward`'.

If you want to run your program with arguments passed in, simply add a space after the class you are running and enter the arguments you would like to pass in, with space separation. For example, if you are trying to run MyProject in the command line use the following commands from your MyProject's project directory:

```
> javac -d bin <path to file with main class> //This first compiles the
project
> cd bin
> java <Path to file with main class> <possible arg 1> <possible arg 2>
```

Make sure that you give a full path to any argument files you would like to use!

Like before, if you want to include the tester, the running is a bit more complex:

```
> java -cp .:<path to the tester.jar> org.junit.runner.JUnitCore ...
```

where the '...' is the code you want to run, exactly as done above.

Running a Scala Program

Running Scala programs in the command line is very similar to running Java programs in the command line.

In order to compile and run your code from the command line, you should follow the exact same steps you used for compiling and running Java code from the command line. The only difference is that you should use the scalac and scala commands, instead of javac and java.

For example, suppose you want to compile and run a Scala file named Class.scala, located in your `sol/lab07/sol` directory. First, navigate to your scalaproject directory and type '`scalac -d bin sol/lab07/sol/Class.scala`'. To then run your compiled code, cd into your scalaproject/bin directory and run '`scala lab07.sol.Class`'.

You will sometimes want to compile multiple files that reside in multiple directories, e.g., in src and in sol. The most concise way to do this is using wildcards, like we did in Java. For example, you can compile all the Scala files in the src/lab07/src and sol/lab07/sol directories like this:

```
> scalac -d bin */lab07/*/*.scala
```

Now, you should have the basic tools needed to navigate and run programs from the command line!