UNIX 1

In this lab you will be doing 3 exercises to help get you started on:

1. The basics of working with files and directories
2. The basics of redirection and ASCII files
3. The basics of how to use the exit status of commands

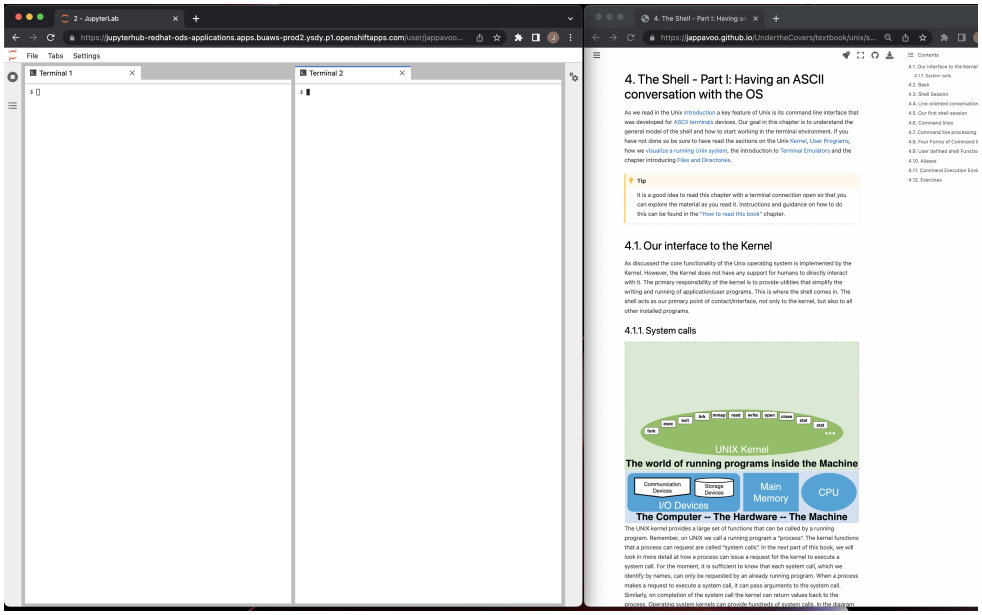
Setup

Form partners and do a quick scan of the three exercises. Then get setup and start on Exercise 1. The TF and CA will be walking around and available to help. Note you may not get everything done in the lab session. That's ok you are encouraged to take the handouts home and continue to play with exercises.

Using the chrome web browser:

1. login to the UNIX environment
2. start your server and create two terminals arranged side by side
3. open another web browser window with "Under the Covers: The Secret Life of Software" open to chapter "4. The Shell - Part I: Having an ASCII conversation with the OS"

Things should look something like this:



The two terminal windows we have opened each have a running independent bash shell process connected to them. The command lines we will enter into the terminal windows will be sent to the connected shells and any responses from the commands are sent back to the terminal window that the shell is connected to. In the rest of the lab we will refer to the left terminal as terminal 1 and the right terminal as terminal 2. To switch between the two terminals you need to use your mouse and click in the window of the terminal you want to be active. Once active, the things you type at the keyboard will go to that terminal window.

Ok lets get going and have fun.

Exercise 1: Exploring Files and Directories

In this exercise, we will have hands on experience working with files and directories in a Unix Operating system. We will use a terminal to interactively send commands to a bash shell process. Each command line will be processed by the shell. After this tutorial, you should be familiar with:

- Creating new directories and files (`mkdir` and `touch`).
- Navigating through Unix file system (`pwd`, `cd`, and `ls`).
- Managing directories and files (`rmdir`, `rm` and `mv`).
- Using terminal efficiently (`man`, and `info` - tabbing - arrow up and down).

For more details about the commands mentioned in this exercise, check chapters 3 and 4 from our textbook `under the covers`.

This exercise has two parts:

- Replicating the directory tree starting from `classes` in the example under 3.3.1.1 in our textbook.
- Dumping the tree for the created directory into a text file.

Creating the directory tree example in section `3.3.1.1` in Under the Covers.

Follow the below steps in order to replicate a portion of the directory tree in the figure. Specifically, we will be focused on the sub-tree that is below the `classes` sub-directory. After you have completed the exercise you can try and create the rest of the tree.

Step 1.

Navigate the “working directory” to the “home” directory and list its current contents. This is where we will build the directory tree in the exercise.

```
cd ~
ls
```

> **Note**
>
> The second command lists the contents of the current working directory. But what did the first command do and why? Before we go on lets confirm what the working directory is. Every process has a working directory. The working directory is a directory that acts as a default location for the commands and operations you do within a particular shell. You can see what directory, path, the current working directory for a shell is by using the `pwd` command (print working directory). Try it out. What do you see? Now in the other terminal window, terminal 2, which is connected to its own, separate, shell process try
>
> ```
> cd /var/log
> pwd
> ```
>
> This sets the working directory of this shell to the directory `/var/log`. Now try running `ls` in terminal 2 and then run `pwd` and `ls` back in terminal 1. **Notice that changing the working directory in one shell has no effect on other shell.**

Step 2.

Back in Terminal 1, where our working directory is `/home/jovyan` let’s create the starting directory in our example tree called `classes`.

```
mkdir classes
ls
```

Step 3.

Now, lets navigate (change our working directory) into `classes` and create three sub-directories called `cs`, `Physics`, and `Psychology`. (Note that writing just `cd cl` then pressing tab will complete the command for you. If the directory has multiple directories with the same name, double pressing tab will show all options.)

```
cd classes
mkdir cs Physics Psychology
```

Step 4.

Let's make sure that everything is going as expected now. Again remember `pwd` command shows the current directory and `ls` command shows the list of directories/files in the current directory. It should show the three directories we have just created.

```
pwd
ls
```

Step 5.

Now let's go on to create the three sub-directories `111`, `112`, and `210` inside the directory `cs`. However, instead of navigating inside `cs`, we will create it while we are inside `classes`. To count for the different parent directory than the current one, we add the name of the new directories appended to its relative path and the command becomes as follows:

```
mkdir cs/111 cs/112 cs/210
```

Step 6.

Now we need to create inside `cs/210` three sub-directories and make a new file inside each one of them. We will use the `touch` command to create empty files. However, since we need to create a new directory first, we will need to use the `mkdir` command. (We could have appended `-p` flag to the `mkdir` command. This flag will not only create the target directory but will also create all needed parents that do not exist. This would have saved us step 5 above).

```
cd cs/210/
mkdir Asisgnment1 Assignment2 Assignment3
touch Asisgnment1/Problem1 Assignment2/Problem1 Assignment3/Problem3
```

Step 7.

Let's make sure that files were created successfully. Again Instead of navigating, we can give the `ls` command the target directory we need to list its content.

```
ls
ls Asisgnment1 Assignment2 Assignment3
```

> **Note**
>
> Again to confirm that what we have been doing has had no effect on the other shell connected to the second terminal let's re-run `pwd` and `ls` in terminal 2. Note however, if we do want to see the changes to the file tree, without changing the second shells working directory , we can pass the path of where we made our change to the `ls` command. Eg. in terminal 2 try
>
> ```
> ls /home/jovyan
> ls /home/jovyan/classes
> ```

Step 8.

Oops, we have a typo in the first directory we wrote `Asignment1` instead of `Assignment1`. Let's fix this by deleting the wrong one and then create a new one. (note that we could use the command `rm` with flag `-r` instead to remove `Assignment1` and all its content recursively.)

```
rm Asisgnment1/Problem1
rmdir Asisgnment1
mkdir Assignment1
touch Assignment1/Problem1
```

Step 9.

You might have noticed that `cs` is lower case, let's see how we can rename it to `CS`. Of course we could delete the entire sub-tree and recreate it but that is more work than we need to do. The UNIX `mv` command lets us "move" paths from one name to another. To do this let's move up the directory tree and then run `mv cs CS` and see what happens.

```
cd ../..
pwd
mv cs CS
ls
cd CS/210
```

Step 10

Now, let's make sure that the whole tree directory is correct. First, we navigate to the parent directory called `classes` and then let's use the command `ls` to see what we have. Since we are interested in seeing the content of `classes` and its children and their children etc., we add the flag `-R` which means recursive to the command `ls`.

```
pwd
cd ../..
pwd
ls -R
```

> **Note**
>
> As a teaser to the next exercise lets save the output of the `ls -R` to a file. To do this we can *redirect* the output as follows
>
> ```
> ls -R > tree.txt
> ```

Now use `ls` to confirm that the file is there. To see the contents of the new `tree.txt` file we can use `cat text.txt`.

Step 11.

Note, to know more info about such cool flags we can use the `man` command and give it the name of the command we want to know more about (the command `ls` for example).

```
man ls
```

Now go back and use `ls -l` to examine the `meta-data` associated with all the directories and files we created.

Step 12 – if you have time

Now can you create the other directories and files so that the output of `ls -R /home/jovyan` matches the figure in the book? But note your tree will be below `/home/jovyan` rather than `/home/jonathan`.

Exercise 2: Exploring Redirection and ASCII files

A hallmark of UNIX is how easy it is to create and work with text based data. Text based files in UNIX are those in which we store information encoded in ASCII (see the section in the text book on files and ASCII Chapter 2.1.1).

In this exercise we will look at some of the basic features of the shell syntax and commands for creating and working with ASCII text files.

This exercise is structured around the task of creating a file that contains a poem. Once we have created the file you will then need to use UNIX commands to work with the file.

“Write a song”

Your goal is to create a file in the home directory called `song`. In this file we want the following text.

```
Type, type, type your Byte
Gently down the standard IO
Bash bash, bash, bash
Life is but a command-line to type

Type, type, type your Byte
Gently down the standard IO
Bash bash, bash, bash
Life is but a command-line to type

Type, type, type your Byte
Gently down the standard IO
Bash bash, bash, bash
Life is but a command-line to type

Type, type, type your Byte
Gently down the standard IO
Bash bash, bash, bash
Life is but a command-line to type
```

Let’s start by creating a file that has one stanza in it

Step 1 create the first line

One of the most powerful aspects of the shell is the ability to redirect the output of a command to arbitrary files.

Perhaps the most basic of built in shell commands is `echo`. Use `echo` and redirection to create the `stanza` file with the first line of our poem.

Lets combine the `echo` command with **standard output redirection** (4.7.5.9.1. Standard output and redirection).

```
echo "Type, type, type your Byte"
```

The above passes `echo` a single argument. This argument is the string `Type, type, type your Byte`. `echo` simply prints its arguments to the standard output stream. So by default we see the line “echoed” back to your terminal. Let’s now use the `>` to redirect it to the file.

```
echo "Type, type, type your Byte"  > stanza
```

> **Note**
>
> Use `ls` and and `ls -l` to confirm that the new `stanza` file was created and that it has come bytes in it.

Step 2 : copy a files contents to your terminal

`cat` is another bread and butter UNIX command that lets us take the contents of a file and copy it. Use `cat` to copy the contents of the song file to your terminal. Lets use it to view the current contents of the stanza `file`.

```
cat stanza
```

Look good? Now run the following and use `cat` again to examine the contents of the file

```
echo "hello" > stanza
```

Oops. What did this do? Let's repeat the right command from step 1 to fix the file. Remember you can use your arrow keys to go back in your history.

Step 3 : append lines

The shell has special syntax that lets us use redirection to append to a file. Let's use it to get the rest of the lines of the first stanza into the file. To append you use the `>>` syntax rather than the single `>`.

```
echo "Gently down the standard IO" >> stanza
```

Now `cat` the file. Look good?

Use the same syntax to add the rest of the lines of the first stanza.

Step 4: creating the song file from from the stanza file

Now that we have the correct contents in the stanza file lets use it to create the song file that has three copies of the stanza file.

Can you figure out how to use `cat` and redirection to do this?

Remember you will also want a blank line in between you stanzas. Hint: what does `echo` do without any arguments.

Step 5: using pipe lines to process ASCII data

Now that we have our song let's learn the model of combining commands to process ASCII files. To do this we are going to use two very powerful UNIX commands `wc` (word count) and `grep` along with shell pipelines.

See the book and man pages for more details.

Try these command lines. What do the do?

```
wc song
wc -l song
wc -w song
wc -c song
```

How about these?

```
grep Type song
grep type song
grep -i type song
grep Gen song
```

Use shell pipelines to do combined processing

We see there are individual commands that we can use for counting lines (`wc -l song`) and for filtering lines (`grep Gen song`). One of the most powerful features of the UNIX model is to combine them using a special form of redirection called `pipelines` (see …)

By default most UNIX commands that can read data from a file will read their input from the standard input stream (you can read the details about this in the book). The shell lets us use features of the UNIX kernel to redirect the standard output of one command to the standard input of another command.

For example

```
cat song | wc -l
cat song | grep Gen
```

Notice that by skipping the name of the input file for the second commands (`wc -l` and `grep Gen`) the commands read their input from the output of the `cat song` command. The `|` symbol tells the shell to setup redirection directly between the two processes started for the command on the left and right of the `|`.

Ok with this in mind what are some command lines that will count the number of lines in the song that have the word Gently.

Exercise 3: Exploring Exit Status

Please read through the following sections in the book and try the examples

- 4.7.7.2 exit status
- 4.8.3.3 AND list
- 4.8.3.4 OR list

The following exercises assume you are in home directory. The absolute path is **/home/jovyan**

Q1

Armed with the knowledge from the book lets come up with “template” command line for running a simple command such that:

1. if the simple command succeeds it will not print anything extra out
2. if the simple command fails it will print out the following error message after any output of the command.

```
Failure: command failed with the following exit status n
```

Where n is the value of the exit status. For example if we ran `ls foo` and foo does not exist then our command line would behave as follows

```
$ ls foo --MAGIC CODE--
ls: cannot access 'foo': No such file or directory
Failure: command failed with the following exit status 2
```

On the other hand if foo does exist then nothing extra would appear:

```
$ ls foo --MAGIC CODE--
foo
```

What MAGIC CODE will achieve the above.

Q2 : Exit status logic

Flip things around from Q1. Instead of printing an error message when the command fails print “YAHOO it worked” every time the simple command succeeds.

Q3 : Exit status logic

Will what you did for Q2 work with a pipeline rather than just a simple command. Eg. We want to print “YAHOO it worked” if the stanza file of Ex2 has lines with the word “Byte”

```
cat stanza | grep Byte -- MAGIC CODE --
```

Q4 : This can get subtle here’s a puzzle

If you want a puzzle what are these two command lines doing?

```
(cat /etc/passwd | grep "jovyan" && echo "found user1") || (cat /etc/passwd | grep
"root" && echo "found user2")
```

and

```
(cat /etc/passwd | grep "foo" && echo "found user1") || (cat /etc/passwd | grep
"root" && echo "found user2")
```

Why do we need the parenthesis.

OPTIONAL: Take home and extra things to try

Explore how to view the hex values of the data of any file

Reading section “The Byte” we are told that all information is represented as byte values and we are introduced to hex notation where byte values are expressed as base 16 numbers. There are several commands that allow you to view the data of a file in various notation.

1. use `man` to learn about `hexdump`
2. use `hexdump` to examine the file you created in Exercise 2. Can you figure what “option” to pass to `hexdump` so that you can see both the hext and ASCII interpretation?
3. try exploring some other files like: `/etc/passwd` and `/opt/conda/bin/python`
4. now use the `file` command (see `man file`) on those same files (eg. `file /etc/passwd`). What do you think is going on?

Explore quoting

Can you figure out and explain what the difference is between the following four command lines?

1. `echo hello there $USER`
2. `echo hello there \$USER`
3. `echo 'hello there $USER'`
4. `echo "hello there $USER"`

The real trick is to understand what is going on not just what the output is. There are sections in Chapter 4 that can help you. Play around with the commands and try variations to test your understanding.

Hint: For each command can figure out how many arguments are passed to `echo`?

Explore Redirection and compound command lines

Can you figure out a way to create the song file from the stanza file in one line?

Why is this not good enough

```
cat stanza; echo; cat stanza; echo; cat stanza > song
```