COSC 2P95 – Lab 1 – Section 01 – Introduction

For this lab, you'll learn how to access your account under Red Hat within the lab, begin learning to use a Linux-based environment, and get your first introduction to Bash.
Refer to the final page to see what you actually need to do to submit.

Logging In

All of the labs for this course will be done through Linux. In this case, the labs have a version of Red Hat (geared towards enterprises and institutions). The *desktop environment* is Gnome. A desktop environment simply refers to the style of GUI (windows, notifications, tiling, menus, etc.) that you have on top of the GNU/Linux OS, as well as the selection of default applications that come preinstalled (e.g. which text editor, which calculator, etc.). Should you ever wish to install Linux yourself, every distribution comes with a default DE, but you can always change that yourself at any time. In any event, Gnome is a good choice.

But we don't care about that right now. For now, we just want to know how to get into it in the lab.

Booting into Linux

When you enter the lab, your computer will be in one of three states:

- In Windows
  - If so, either shutdown, or restart to get you to the next state
- Turned off
  - If so, turn it on (did anyone need this one explicitly stated?)
- Already in Linux
  - This is good, but still read the following steps in case it isn't next time

When a system has multiple operating systems installed, there's a *boot menu* that allows you to select which OS you wish to run when it first turns on.

The lab computers are set to only display a prompt, rather than the menu, so press any key when you see that prompt, and you should be presented with the list of installed operating systems.

By default, Windows will be the selected choice, but you'll notice that Red Hat Enterprise is at the very top of the list. Using the cursor keys, navigate up to highlight it, and press Enter.

Congratulations! The first step is done!

Logging In

Logging in is the next adventure.
You have a login that's shared across all COSC computer labs, that's independent from your normal login within the other computer labs at Brock.

Your username is the same, but your password is separate (unless, of course, you manually change it to be the same as your normal password).

Important: If you have never logged into a COSC lab (J301, J310, or D205) before, then your initial password will be your *student number*. Obviously, after logging in, you'll want to change that very very quickly (which we will fairly soon).

Assuming you logged in, you should now see the desktop environment.

Environment

Gnome uses a pretty classic desktop style. There are a couple things that might be immediately noticeable:

- You have a desktop
  - This is just like the Windows desktop
  - Please don't fill it with random files and folders. Or do. I'm not the boss of you
- At the top and bottom of the screen should be *panels*
  - A Gnome panel is like a taskbar, but more customizable
  - The bottom panel shows the application switcher and the workspaces
    - Both of these will be more interesting one you have some applications running
  - The top panel shows the main menu, including:
    - Applications: arranged in different categories
    - Places: shortcuts to open *nautilus* (the file browser) in different important folders, as well as showing any mounted drives (flash drives, etc.)
    - System: where administration and preferences handled

To start, let's start our text editor, *gedit*. Click on Applications → Accessories → gedit

Note that, when an application is running, you can see it in the bottom panel. This behaves pretty much the same as the classic Windows taskbar.

Let's also start nautilus. Click on Places → Home Folder.

- Your *home folder* is the top-level directory that holds all of the user-specific files for a user's profile
  - This includes your important files like Documents, Pictures, etc.
  - It also holds your personal configuration files, as well as cache files
  - If you open up a web browser (say, Firefox), then your browser profile will also be in here (albeit within a hidden folder)
- Depending on whether or not you've ever taken any COSC/APCO courses, this folder could either be very sparse, or *very* full. Either way, we'll soon be adding plenty to it
  - You'll also notice that you have a Desktop folder there (which mirrors what's on your Desktop), and those other folders you saw in Places

Finally, let's start a web browser (Firefox).
You're presumably used to this from Windows as well, but windows can be maximized, minimized, restored, etc. You can probably even find a place under Preferences to let you *roll up* windows by double-clicking their title bars (but you probably shouldn't. That's just weird).

However, as you get more and more windows on the screen, two problems tend to pop up:

- You might want to be able to see Window A, even when Window B is active and large
- You might want to have several windows open; some for one task, and some for another

We actually have solutions for both!
First, on gedit, click on the control box (upper-left corner). There'll be plenty of options in the drop-down, but we're looking for *Always on Top*. Click it.
Congratulations! Irrespective of which other windows you click on, that one will stay at the front.

As for needing more space to work, simply buy and attach a second monitor.

That's an option, right?

… no?

sigh

See those two grey boxes in the lower-right corner (in the panel)? That's your workspace switcher. The concept of workspaces has slowly been migrating into other operating systems as well, but in case you've never used them, the idea is that you can have multiple copies of your desktop view, with application windows assigned to different ones.

This is easier to just demonstrate.
See all those windows?
Press ctrl+alt+right.

Want to get back? Ctrl+alt+left.

You can also click on those workspace boxes in the panel. Alternatively, you can right-click on them to find the shortcut for changing the workspace preferences (including changing how many workspaces you have available). Personally, I think the initial setup is good, but feel free to change it as you see fit.

There are multiple ways to get a window to another workspace:

- Simply switch to the other workspace first, and then start the application
- Click on that control box, and tell it to move the window to the next workspace
- Ctrl+alt+shift+right (or ctrl+alt+shift+left)

Some DEs let you lay out your workspaces vertically, in addition to horizontally. For those, you simply also use up/down, as well as left/right.

There are two last things to know before we get to the terminal.

Whenever you need to leave your workstation for an extended period of time, just as when you're using windows, you should **lock your computer**. Of course, don't just lock it and then wander off for the day.
To lock your computer, press ctrl+alt+l (as in L, but you don't press shift).

Finally, though you can run most applications either from the main menu or from the terminal (which we'll be getting to next), you should also know about the handy-dandy run menu!
Press alt+F2 to bring it up.

When you start to type in the name of a command, the autocomplete can eventually take over (note, however, that this goes by the case-sensitive executable name; not what would appear in the title bar).
Since we need our terminal next, and the normal way to start it is mildly tedious (Applications → System Tools → Gnome Terminal? Sounds like work!), let's try using the run menu to start it:
gnome-terminal

This should bring up a *terminal program*, inside of which a *Bash shell* will be running.

Bash
Okay, so we're in the Gnome Terminal, using Bash. We'll be very slowly learning bash, a bit at a time. But it never hurts to get a taste of the basics, right?

If today was your first time ever logging in to a COSC computer, then you have an unacceptably terrible password, so… **change your password!**
How? Glad you asked!

Changing your password

There are two ways to change your password. I honestly don't care which you use.

- If you do happen to be logged into a COSC lab computer, under Windows, then just hit ctrl+alt+delete, and choose 'Change Password'
- Otherwise, we need the Bash terminal:
  - First, type: `kinit`
    - This should ask for your password, but won't *appear* to have done anything
  - Next, type: `kpasswd`
    - This will have you retype your current password, and then have you enter the new one twice
  - If this somehow doesn't work correctly, you'll need to either ask our sysadmin for help, or just try again once you're into sandcastle (which is coming up very soon below)

Before we get any further, I should point out that sooner or later you're going to need to look commands up. Thankfully, it's very easily googled (either by looking or *bash*, *linux command line*, or *CLI*). That said, let's see what we can start off with.

When you first see the terminal, you'll notice that there's a prompt with your username, possibly a host name, possibly a colon, likely a ~, and maybe a $.
You can actually customize that prompt, to include whatever fields you like, but let's discuss what you see now.

- Your name is you
- The host name is the name of the host
- : and $ are arbitrary filler
- ~ is your *current working directory*
  - It's shorthand for your *home folder*
  - In other words, it says that you're in the same folder as when you opened nautilus to your Home Folder above

But, how do we know where we are? For sure?
Try typing: `pwd` (for *print working directory*)
You'll see where you are, as a full path off the root folder (the anchor point for the entire file system).

To change directories, use `cd`. To see a listing of the current directory, use `ls`.
e.g.

```
cd Videos
ls
```

Of course, you won't have anything in the Videos folder (presumably), so let's return to the previous folder.
Either type:

```
cd ..
```

or

```
cd ~
```

or

```
cd /home/std/(username)
```

The last two should be pretty easy to understand. But .. might be new to you. A common convention is to say that whatever folder you're currently in is also know as a single dot. To return to the parent folder, .. is used.
So, for example:

```
cd .
```

does effectively nothing.

Let's try something a little more interesting. Try this:
`ls -lah`

Your home folder suddenly has a lot more in it now, eh?
This time, we added some *command-line arguments* (commonly known as flags in this context).
For most bash commands, we precede them with a -, and then it doesn't matter if we combine them, or use separate dashes for each term (i.e. `ls -l -a -h`, or even `ls -h -al` would have produced the same results). Question is, what does `hal` mean?

- `-l` says to use the long listing format (i.e. show more information about each file/folder)
  - In this case, it also showed us file sizes, timestamps, permissions, as well as ownership of files
    - Linux assigns both a *username* and a *usergroup* to every file
    - The permissions indicate who can read, edit, or execute that file, in the sequence of owner→group→all (so rwx------ means only you can read, change or execute a file, while r-xr-xr-x means *everyone* can read and execute it, but nobody is allowed to change its contents)
    - Note that the file sizes normally wouldn't be as readable (try `ls -la` to see what I mean)
- `-a` says to display all files
  - Notice that the new folders (that hadn't showed up before) all start with a period
  - Starting filenames/folder names with a period is how you mark something as *hidden*
- `-h` says to make the listing human readable. This means using shorthand like K, etc.

However, that said, we can do *so* much more with our listings!
You could type `ls --help` to get some help, but we have a better option.

Read the manual!
If you type `man` (command), that brings up the documentation for that tool.
So, try: `man ls`
Use the up/down cursors to scroll (or the space bar or PgUp/PgDn to go by page).
Press `q` to quit.
Press `/` if you want to search for a term.

Note that you have all sorts of options, including controlling what does or doesn't appear, as well as how to sort the listing.
For ha-ha's, try typing:
`ls -drl D*`

The `*` is a wildcard, telling it to match with any sequence of characters (a ? can match with any single character).
Can you tell what the `d` did? Try it again without the `d`.
By all means, feel free to refer back to the *man* page, but I wouldn't advise continuing until you can understand that statement.

Let's create a file:
`touch dealie`

Sure enough, we now have an empty (0 bytes) file, called dealie.
Let's edit the file:
`nano dealie`
nano is a *very* simple text editor, but it can be used without a GUI (so you can run it on sandcastle if you like).

You can read most of the keyboard shortcuts at the bottom of the screen, but the important ones are:

- ctrl+x to exit
- ctrl+o to save (or *output*)
  - Note that it'll have you verify the filename you want to save as
- ctrl+c to show your current position
- ctrl+w to find text within the document
- ctrl+k to cut the entire current line you're on
- ctrl+u to *un*cut (paste) that line back
  - You can uncut repeatedly, if you wish to create several duplicates of a line

Enter some text (gibberish is fine), save and exit.
If you do a listing again, you'll see that it now has a filesize greater than zero.
Let's make a copy of it.

```
cp dealie grobble
```

The man page has some tips on interesting flags (including `-r` to recursively copy entire folders), but the general usage is pretty simple: `cp` *`source target`*

Let's move the copy to our desktop:

```
mv grobble Desktop
```

If you're ever feeling lazy with paths, Bash supports tab completion.
For example, if you typed `mv g` and then pressed the tab key, it would finish *grobble* for you. On the other hand, if you pressed tab after the `D`, that wouldn't do much (because D could match with Desktop, Documents, or Downloads). Pressing tab a second time would display the matching options.

After the move, you should now see the file on your actual desktop.

Next, let's rename *grobble* to *wokka*.
The command to rename is… uh…
it's…

there isn't one (exactly).
Pop quiz: what's the difference between *renaming* a file and *moving* it? One changes the filename, while the other changes the path, right? But that means both are just changing part of the total path. The point: we rename via `mv`. So:

```
mv Desktop/grobble Desktop/wokka
```

Note that you can include full paths if you like. Heck, you could have typed:

```
mv ~/Desktop/grobble ./Desktop/../Desktop/wokka
```

(though I can't imagine why you would)

Feel free to play with ls, cd, mkdir, rmdir, and chmod.
What are those last three? I bet you they all have *man* entries!
One small note: never use `rm -r` (seriously. It's for recursively annihilating anything from the current level inwards. It's entirely safe and convenient when you're careful, but it can also wipe out your profile when you're not)

One last thing to note: what if you know that there's a command for what you need, but can't remember what it's called? You can hardly read the manual if you don't know the name.
Let's say I know I can estimate file space… somehow:

```
apropos estimate
```

Hey, apparently I was thinking of the `du` command! That actually looks like a very useful tool.
Personally, I'm partial to:
```
du --max-depth 1 -h
```
to see both how much space I'm using, and where.

Let's try displaying a bit of text:
```
echo hello
```
Alternatively, we could type:
```
echo 'hello'
```

Have you wondered why we can just type the names of commands to run them, but don't need to specify where they are? That's because they're in the *path* environment variable. Let's see what's in that:
```
echo $PATH
```
(again, remember that this is all case-sensitive)
Any program within any of those folders can be executed by simply typing its name. You can add more folders to that path if you like (but that's for another day).

Overall, echo is a very common command, but we probably won't need it for a while.

Let's look at the contents of dealie!
```
cat dealie
```
(don't forget the *man* pages for all of these commands! For example, cat can number the lines it prints)

The *cat* comand is used for a *lot* of things.

Though this isn't commonly necessary, let's address the need of having to access the same file from multiple locations. *Normally*, it's best to simply use the full path from both sides. However, if necessary, a file can be two places at once.

First, type this:
```
ln dealie stuff
```

This creates a *hard link* from *dealie* to *stuff*. What does that mean?
- Take a look at the directory listing. You'll see both files.
- Inspect the contents of both (either with cat or gedit or nano). They're the same
- Try editing one and saving. Now look at the other. It has the change!
- Basically: one place on the hard drive, but two entries in the file system

Now, try deleting *dealie*. The *stuff* file is still there (and it's just a normal file now, as it isn't sharing the target with any other links).

So, what's the complement to a hard link? A symlink, of course (I… didn't name these things).
Assuming you've been following along exactly, try this:
```
ln -s stuff dealie
ls -l
```

Huh. This time, it's clear that *stuff* is the 'real' file, and *dealie* is just a shortcut to it.
- Symlinks are a little odd
  - Some applications will treat them like the original files; others won't follow them
  - Some applications let you choose whether or not to follow them

- If you try to cat *dealie*, you'll see the contents of *stuff*

But let's try to delete *stuff*:

```
rm stuff
ls -l
cat dealie
```

Oh… it is not pleased, eh? Better just delete that, too:

```
rm dealie
```

Pipes and redirecting standard streams

Again, we can't cover everything, but we at least need to take the first baby steps for pipes and redirection.

There are three standard streams: *standard in*, *standard out*, and *standard error*.
Standard in, unless you say otherwise, will be what you type from the keyboard.
Standard out displays on the terminal, and is associated with normal output.
Standard error also displays on the terminal, but typically either when something goes wrong, or when feedback should be considered separate from normal data.

To be clear, until you say otherwise, standard out and error *look* the same (we'll be dealing with the latter later).

Try this:

```
ls --help > listing.txt
```

We just created a new file! How? By redirecting the output to that file, instead of the screen.
If we wanted to explicitly only redirect the standard output stream, we'd change `>` to `1>` (2 is error).

Well then, let's look at the contents:

```
cat listing.txt
```

Oh right. Still hard to read, right?
Well then, let's use less! (There's also more, but less > more)

```
less listing.txt
```

If it feels familiar, that's because it's virtually identical to how *man* works. Use the same keyboard shortcuts to search/scroll/exit.

Just for ha-ha's, what if we wanted to still use *cat*, but make it pause like that?
That's easy!

```
cat listing.txt | less
```

(if it isn't showing up well, that's the 'vertical bar' above the backslash key)

This says:

- Execute both the *cat* and *less* programs simultaneously
- Instead of *cat*'s output going to the screen, it's being piped into *less*
- Similarly, instead of receiving from the keyboard, *less* receives from *cat*

Let's try one more use of cat:

```
cat > something.txt
```

(and type several lines of text)

You can keep typing and typing, right? But… how do you stop?
Instead of a file, *cat* is using standard input. It'll keep going until it thinks it's hit the 'end of file'.
Simply press ctrl+d to send an EOF.

SSH
There's one last major thing to cover: Secure Shell.
The secure shell lets you connect to a remote (or local) computer over an SSL/TLS-encrypted socket connection. i.e. it gives you a secure means of logging into a terminal.

There's quite a bit you can do with ssh/ssl, including file transfers, but let's just worry about logging into a remote device. Like… sandcastle.

Try this:
`ssh sandcastle.cosc.brocku.ca`

(If you ever need to do this from another computer where you have a different username, just use `ssh username@sandcastle.cosc.brocku.ca`)

It should prompt you for your password.

Congratulations! You're still in a Bash shell, but now you're in a Bash shell running on a completely different computer! Sandcastle is the main COSC server. Try looking at a directory listing…

gasplesauce!

Yep. Your home folder on a lab computer (running Linux) is actually the same as your home folder when you're logged into sandcastle. This means that if you connect to sandcastle remotely, and upload a file, it'll be available to you in the lab (and vice versa).

If you're interested, take a quick look at *scp* and *sftp* in the *man* pages.

Exiting the shell
When you're done with a Bash shell, you can simply type `exit`.
However, four letters is a lot of work… ctrl+d (remember: EOF) tells the shell that you have nothing left to say, so that works the same as an `exit`.

Submission
To demonstrate that you've attained a basic working knowledge of the environment, and that you can access basic resources:

- Show your lab demonstrator some file you've created in nautilus (i.e. show that it exists; not its contents)
- Show your lab demonstrator that same file, from a Bash shell logged into sandcastle