COMP 2401 -- Tutorial #1

Welcome to Linux

Learning Objectives

After this tutorial, you will be able to:

- navigate a Linux file system
- consult man pages / use the help command
- create and edit a text file
- create and extract tar files

Tutorial

It is essential that you are able to navigate the file system in a Linux/Unix system. Some commands, such as `ls`, `cd`, `cp` and `rm` must be second nature to you.

1. Open Oracle VirtualBox and start the course's virtual machine. You can log in using the following credentials

   user : `student`
   pass : `student`

   More information about VirtualBox and the course's virtual machine are available in the Resources section for the course in *cuLearn*.

2. Once you have successfully logged on, open a shell (the traditional command line UI for Unix-like operating systems) by running the terminal program. You can do this by clicking on the “Terminal” icon in the task bar on the left-hand side of the screen. Once in the shell, you have access to many powerful programs and built-in shell commands.

   In the shell, type the following commands (hitting Enter after each command to execute it):

   ```
   whoami
   date
   pwd
   ```

   The first two commands are examples of programs provided for you to use (that are external to the particular shell). The first command (`whoami`) outputs who the current user is and the second (`date`) outputs the current date. The third command (`pwd`) is an example of a built-in shell command. Its name comes from print working directory and it outputs where you are currently located in the file system. You can identify which commands are external programs and which are built-in shell commands with the `type` command. It either shows where in the file system the executable program exists or identifies the command as built-in. Try the following:

```
type whoami
type date
type pwd
```

To learn more about a command, you can use the `man` command to read the manual page for an external command, or you can use the `help` command to get help for a built-in command. (Note: both `man` and `help` will work for some, but not all, built-in commands.) Try reading the manual pages for the three commands from above. That is, type:

```
man whoami
man date
man pwd
```

The up and down cursor keys will let you scroll through the `man` pages. The space bar will move you down a whole page. Now try getting `help` for `date` and `pwd`. That is, type:

```
help date
help pwd
```

3. Use `man` or `help` to read about each of the following commands: `ls`, `cd`, `cp`, `rm`, `mv`, and `mkdir`.

4. Starting from your home directory, let's see what files (and directories) are there. First, try simply typing `ls` to list the files in the current working directory:

   ```
   ls
   ```

   You can modify how `ls` displays information by giving it some options. In bash, a convenient way to use `ls` is with the `alF` options. Type `ls -alF` and notice the different output. This is such a useful way of using `ls`, that there is an alias to this already set up for you. Try the command `ll` (ell ell). Using the `type` command (`type ll`) you'll see that it is an alias and not a command. It is sort of like a shortcut. To see all the aliases already set up for you, type the command `alias`. Each line displayed is an alias that you can use. (You can ignore the first 4 for the time being.)

   You should have noticed that there are a lot more files (and directories) listed with `ls –alF` than with just `ls`. These extra files (directories) all that start with a "`.`" and are called hidden files (directories). Your home directory will always have a lot of these. Type `ls -lF` now and see all the non-hidden files and directories. Type `man ls` and see what the options (`a`, `l` and `F`) you have been using with `ls` mean.

   How do you know which are files and which are directories? (We'll talk about this more later.)

5. Let's create a new directory.

   ```
   mkdir testDirectory
   ```

   Type `ls` to confirm that the new directory was created and then move into this directory. You can change directory by using the `cd` command: `cd testDirectory`. Note that `cd` is a built-in shell command and there is no `man` page for it (there is a `help` page though). How do you know if you have actually moved into this directory? You can always print the current working directory to see where you are in the file system (`pwd`).

   What is in this directory? There should only be two hidden directories: "`./`" and "`../`". The first represents the current directory that we are in and the second is the parent directory of the current directory. This is extremely useful, since we can easily get back to the parent directory (which happens to be the home directory) with:

```
cd ..
```

We don't have to always remember the name of the parent directory. It is always accessible through "`..`". If we want to move up two levels, to the parent directory of the parent directory, we could use "`cd ../../`" (without the quotes).

6. Take some time to explore the file system. There isn't much in your home directory at the moment. However, your home directory is not the root directory of the file system. Type:

   ```
   cd /
   ```

   This changes directory to the root directory (`/`). Explore a bit of the file system starting from the root. In particular, notice that `/bin` and `/usr/bin` contain the programs for the external commands that you have been using so far (`mkdir`, `whoami`, etc). For more information about the file system structure in Linux, see here.

   Don't worry about trying to find your way back to your home directory. There are several ways that you can quickly do this. From anywhere in the file system, typing one of:

   ```
   cd
   cd ~
   cd $HOME
   ```

   will always bring you back to your home directory. The last example uses a shell variable, called `HOME`, which stores the name of your home directory. To access variables, you need to use `$`. The "`~`" is a built-in alias for `$HOME`. Try typing:

   ```
   echo $HOME
   ```

   This will print (`echo`) the value of the variable `HOME` to the screen. It should print "`/home/student`".

7. From your home directory (type `pwd` to make sure you are there), remove "`testDirectory`", create a new directory called "`Tutorial1`" and then move into this new directory. Do not delete any files (that you will be asked to create or use) unless directed to do so.

   Using a browser, log into *cuLearn* and go to the tutorial page. Download the `T01.tar` file and save it into your "Tutorial1" directory.

   A tar file is an archive file. You can think of a tar file like a zip file that is not compressed. It is a common way of collecting files together into a single file archive in Unix/Linux. In order to extract the files in the tar file (think unzip), use the `tar` command:

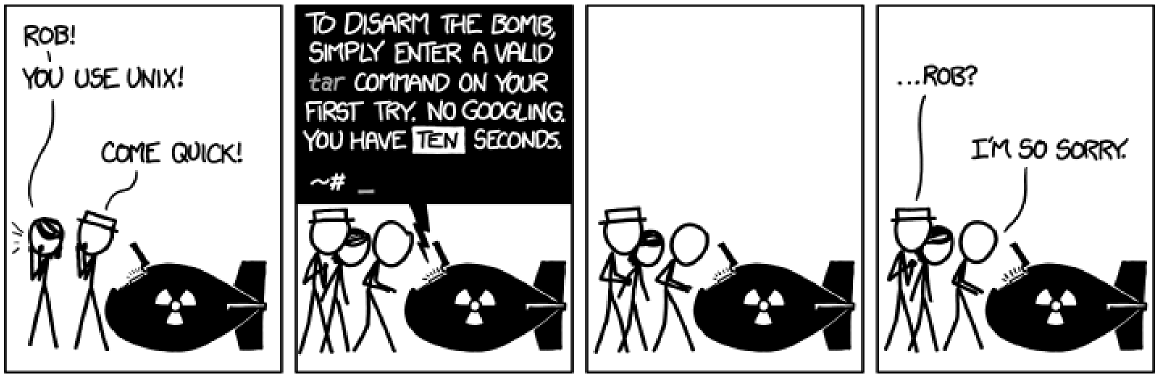
   ```
   tar -xvf T01.tar
   ```

   Here the options are "`x`" for extract, "`v`" for verbose, and "`f`" for file. This command will untar (or e<u>x</u>tract files from) the <u>f</u>ile called "`T01.tar`", while outputting some messages (<u>v</u>erbose mode).

   After you untar the "`T01.tar`" file, you will notice that there are several new files in your "`Tutorial1`" directory. Create a new tar file called "`Tutorial1.tar`" that contains all of these new files that have the

"`.addme`" extension (and no other files for now). Read the `man` page for `tar` to see how to create a tar file. [Hint: `x` is for extract and `c` is for create.]

Tar files are very common in Unix/Linux systems. You will be expected to tar and untar files for your assignments. Don't be discouraged if it takes you a while to remember how to do this without getting help (read the `man` page, or a quick google search, which will most likely bring you to the `man` page online). The following is from the XKCD online comic.


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

8. In your "`Tutorial1`" directory, create a new file called "`file1`". Open an editor (emacs, vim, gedit, pico, ...), add the line "I laughed, I cried, it was better than Cats!", and then save the file as "`file1`".

   Note: file name extensions are not needed in Linux. They are used to give the user/program a hint as to what format the file will be.

   You can quickly read the contents of your file without having to re-open an editor by using one of the following commands: `cat`, `more` or `less`. The `cat` command is great for reading short files, while `more` and `less` are quite useful for large (multi-page) files.

   Once you verify that your file does contain the line you added, add the "`file1`" to your "`Tutorial1.tar`" file. Do not simply create a new tar file with the desired files. Using the `tar` command with an appropriate option, you should add the file to the existing tar file.

9. In the "`Tutorial1`" directory, there should be a subdirectory called "`Sub1`". Change directory to "`Sub1`" and notice that there are two files in it: "`copyMe`" and "`IRun`". Using the `cp` command, copy this file into a new file called "`runMe`". You should now have three files in the directory "`Sub1`". All of these files are examples of simple `bash` script files. Try running the `runMe` script with:

   ```
   runMe
   ```

   Instead of executing the script file, the shell should have given you the error message "`runMe: command not found`". In the `bash` shell, the *default path* (where the shell looks for a program when you enter a command name) does not include the current directory. You either have to manually add the current directory ("`./`") to the `PATH` variable (try "`echo $PATH`" to see what the current search path is) or you need to explicitly tell the shell where the "`runMe`" program exists in the file system. We'll explicitly tell the shell that the file is in the current directory with:

   ```
   ./runMe
   ```

The shell did find the file this time, but now gives the error message: "`bash: ./runMe: Permission denied`". The problem here is that you do not have permission to execute this file. It is not an executable file. How do you know which files are executable and which are not?

The `ls` command provides us with a wealth of information about the files in the file system. It can be used to easily show us which files are executable and which are not. For example, the file "`IRun`" is an executable `bash` script. The `ls` command lists "`IRun`" in green and the `ll` command (which is just `ls` with some options) shows "`IRun`" in green and puts an asterisks (`*`) after the name. All of these indicate that "`IRun`" is executable. More importantly, however, the `ll` command displays all the file permissions for each file (and directory). The file permissions are shown in the first 10 columns of each line in the output. The output of `ll` in the "`Sub1`" directory should be (very similar to):

```
drwxrwxr-x  2 student student 4096 May  2 08:32 ./
drwxr-xr-x 26 student student 4096 May  2 08:28 ../
-rw-rw-r--  1 student student 1456 May  5 21:41 copyMe
-rwxrw-r--  1 student student 9654 May  5 21:56 IRun*
-rw-rw-r--  1 student student 1456 May  5 21:41 runMe
```

The first column (character) specifies if the file is a directory (`d`) or not (`-`). The next three columns give the read, write and execute permissions, in that order, for the user (you!). An "`r`" means it is readable, a "`w`" means it is writable, and an "`x`" means it is executable. A dash ("`-`") indicates that the given permission is not granted. The next three columns give the read, write and execute permissions for other users in your group, and the last three columns give the read, write and execute permissions for others (everyone in the world that has access to this file).

For example, the user (`student`, which is you) has read and write permissions for all three files. But, only the user has permission to execute the "`IRun`" program. The parent directory ("`../`"), on the other hand, is readable and executable by the user, the group and anyone that can access this file (the world). However, only the user has permission to write to it.

Permissions of a file (or directory) can be changed, if you own the file. In the output of `ll`, shown above, the first column with "`student`" in it, indicates who owns each file. In this case, you (`student`) owns everything. Using the `chmod` program, we change the access permission to a file system objects. For example, consider:

```
chmod u+x runMe
chmod a-rw copyMe
```

The first command will make the file "`runMe`" executable for the user (we have added, `+`, the e<u>x</u>ecute permission for the <u>u</u>ser). The second command removes read and write access for the file "`copyMe`" for <u>a</u>ll (user, group and other). For more information, see the `man` page for `chmod`.

The "`runMe`" script does some basic `bash` programming and calls some useful external commands. If you are interested, read through the file and execute it when you are finished the tutorial.

Save Your Work

10. You should save all of your work for all the tutorials since some of the tutorials are built on your solutions for the previous tutorials (and you can also refer to them later if needed). You can use dropbox, google drive or simply using the Z-drive. There is shortcut on the desktop of the virtual machine to SharedFolders, which includes your Z-drive.