INFO1910 Week 1 Tutorial

Introduction to Linux

Introduction to the Course

Ed
We will be using Ed this semester for discussions, answering questions, and some assessments. Check Ed at least once a day, feel free to try to answer questions posted there, it's a good test of whether you understand the content.

On the topic of asking questions, do this. When you're stuck, search things up, and when you're still stuck ask.

Canvas
You can find your lecture recordings here, in particular the Advanced lecture recordings. Your current marks will also be recorded on Canvas under the marks tab.

Git
We will be learning and using git in this course. Git is a version control system, it trackschanges to collections of files and allows you to share your modifications between different computers.

Linux
You will **need** your own local installation of Linux, either on bare metal, dual booted or in a VM, OSX is a stopgap measure if you don't want to dual boot and your hardware is insufficient to run a VM, but will lead to problems in future subjects. WSL and Git Bash will not be sufficient. If you have difficulty doing this please ask sooner rather than later. For installation instructions, see the 'Week 0' tutorial sheet.

Text Editors
You will need to be familiar with a text editor (not an IDE). The recommendation is for a combination of vim or emacs, and sublime text or atom, but other options are out there. Again see the 'Week 0' tutorial sheet for more details.

Introduction to Linux

What is Linux

Linux is a family of operating systems that derive from the original AT&T Unix developed in the 1970's at Bell Labs. As an operating system it manages the computer's hardware and software and provides a common interface for applications to use as they manipulate elements of the system.

Unix follows a philosophy that revolves around designing small, well defined programs that integrate well together. This can be summarised as follows:

- Write programs that do one thing and do it well
- Write programs to work together
- Write programs to handle text streams, because that is the universal interface

Everything in Unix is either a file or a process. A file is any collection of data, such as a series of configuration strings, an image, or compiled source code, while a process is a program that is currently being executed. Unix then revolves mostly around passing and parsing streams of data between different processes and files.

The Kernel and the Shell

The Unix kernel is a master control program that handles low level tasks such as starting and stopping programs, managing the file system and peripheral devices and passing instructions to processors or graphics cards. The kernel is complimented by a series of small programs that perform common tasks such as listing the files in a directory, changing directories and executing compiled code. Other tools can then build off these functions to provide search functions, document readers and image processors such as desktop environments.

Optional Challenge

If you already have some prior exposure to Linux, or the command line, consider the following 'hard mode' version of this tutorial. Upon reaching the login screen press **Ctrl + Alt + F2**, your screen should turn black and you should be shown a login prompt. You will not have GUI for the rest of the tutorial. If you're not running Linux, this won't work, but you can try to restrict yourself to the terminal anyway.

Terminal Basics

One of the most useful linux commands is `man`. This brings up the **man**ual pages for a given command and shows options and example usages if you are unsure of the syntax. To **q**uit the man page, press **q**. As an example, enter the command `man ls` to see the manual pages for the **lis**t command. This command shows the names of all the files and folders in the current directory. You should see in the man page the option `ls -a` which shows all files.

Quit out of the man page and try both **ls** and **ls -a**. You should notice that the second command lists quite a few more files than the first, all of them prefixed with a full stop.

Question 1: Dot Prefixing

- What does it mean when a file is prefixed with a dot? (The manual pages for `ls` might shed some clues)
- Is this true on other operating systems?

`mkdir` is the **m**a**k**e **dir**ectory command. As the command suggests it creates a new directory with whatever name you specify. For example `mkdir info1910` creates a new directory in the current folder called ‘info1910’. Directories can be **rem**oved with the `rmdir` command.

Next we want to be able to navigate between directories. For this we have the **c**hange **d**irectory command `cd`. To move to the info1910 directory you would simply `cd info1910`. Along with chanding directories, we can print our **p**resent **w**orking **d**irectory with the `pwd` command.

Question 2: Terminal Navigation

Using **cd ..** navigate upwards from your current directory to the topmost or ‘root’ directory, pay attention to the path that you took and then try to navigate back to your home directory. If you get lost `cd ~` will automatically take you back.

You can also revert to the directory you were immediately previously in with `cd -`.

Tab Completion

Typing out the full names of each folder every time is somewhat redundant. To work around this Linux features tab completion. Pressing tab will automatically complete your current command or path if it can be uniquely determined. If it cannot be uniquely determined, pressing tab twice will list all possible completions. Fancier shells also contain inbuilt completions for arguments passed to commands along with device drivers and other useful features.

From the info1910 directory type `cd ../D` and then press tab twice. You should see the following output.

```
Desktop/ Documents/ Downloads/
```

Changing this to `cd ../Doc` and then pressing tab once should complete the command. Once you've entered this command use tab completion to navigate back to your info1910 directory.

If your screen is looking full you can use the `clear` command or *Ctrl + L* to clear the screen.

Files

As stated above, in Unix everything is either a file or a process. Files can be read, written to and executed. To see what permissions a file has use the **-l** option for the `ls` command.

```
-rw-r--r-- 1 root root   387 Dec 10 10:17 Makefile
```

Here this Makefile has read and write permissions for the user, and read permissions for group and anyone. File permissions can be changed using the `chmod` command.

There are a number of commands that can be used to read and write to files, the most basic of which are **cat** and **touch**. The cat command con**cat**enates files and prints the output, allowing the easy reading of plaintext files while touch changes the timestamps of a file, if the file doesn't exist then it creates it. **less** also reads text files, but places the user in an environment that is virtually identical to the one for **man**. It can be escaped using *q* in the same manner.

Existing files can be **cop**ied using `cp` **mo**ved with `mv`, and **rem**oved with `rm`. Directories are removed using `rmdir`. Though there are a few flags that can be passed to the rm command to grant it the same utility. Hidden files are prefixed with a . and can be viewed using the -a flag with the ls command. These hidden files are also called 'dotfiles' and tend to be configuration files for various programs.

To add text to a file we can use the **echo** command. Echo prints whatever input it is given, for example

```
echo "Hello World!"
```

Question 3: Recursive Remove

Make a new directory titled 'delete_me', and in it create some empty files with the `touch` or `echo` command. Move back to your home directory and using a single bash command, remove the 'delete_me' directory and all it's subdirectories and files. The manual pages will likely be a great help here.

Piping and Redirection

We can redirect the output of any Linux command to any other Linux command, or to a file.

As per the Linux philosophy, text can be directed between processes and to files. In this case the > and >> symbols write and append to files respectively.

```
echo "Hello World!" > hello.txt
```

As all inputs and outputs from processes are text based, the output from one command can be 'piped' as the input to another command.

```
cat my_file.txt | less
```

The above command takes the output of cat and displays it using the 'less' command, which controls in a similar manner to the `man` command you saw earlier.

Question 4: Dictionary Search

Linux comes with selection of dictionaries they can be found at **usr/share/dict**. Grep, pipes and word counts will help you here.

- Find all words in the dictionary containing the substring 'them', count how many of them there are.
- Help me at hangman by searching that file for all words that contain no vowels. Put them in a file for me.

Question 5: Repeat

Linux comes with a number of configuration files that control the behaviour of just about everything on the computer. You might have noticed that the 'up' and 'down' arrow keys allow you to view previously entered commands. This feature is supported by the '.bash_history' file (you may have a slightly different file depending on your shell, check your home directory). Use `cat` to have a look in the file and see that your previously entered commands are all there.

- Count how many times you've used the grep command.
- Using the `head` and `tail` commands, along with `/bin/bash`, write a single line of bash that re-runs the last entered command.

For future reference, the shortcut `!!` expands to the previously entered command. Try it.

Vim and Emacs

To edit a file from the command line, there exist two common text editors. These are **vim** and **emacs** and you should learn to use at least one of them.

Vim is smaller, and somewhat less featured while Emacs rivals operating systems in size but is more bloated.

Before you open and use either of them, it is incredibly useful to know in advance how to close them again. In vim you will be looking to press the escape button and then to enter the ':q' command, or the ':q!' command if you're really stuck.

In Emacs you're going to need to press 'Ctrl + X Ctrl + C', then either save or clear the buffer and answer 'yes' if prompted whether you'd like to exit anyway. In the emacs prompt control is abbreviated as a capital C.

To use either to edit a file simply enter **vim my_file** or **emacs my_file**.

In vim press the 'i' key to enter 'insert' mode, and type what text you need to. Then press escape to return to the 'command mode' from where you can save with ':w' and quit with ':q'. You can also combine these commands to ':wq' to save and close the file and vim.

In emacs you can immediately type your text into the 'buffer' which will be periodically auto-saved to the file. You can also force it to save using 'C-x C-s' or 'Ctrl + x Ctrl + s', and then quit.

This is a very brief introduction to these two editors and you should practice using them and learning other commands and shortcuts for each in your own time.

The UNIX Filesystem

In the root directory you can find a number of different folders with specific functions.

- *home* Contains the home directory for each user. Your home directory contains your Documents, Downloads and other directories. When logged in the system variable $HOME is associated with your home directory. Try **cd $HOME** to see it being used.
- *bin* contains binary executable files that execute most of the processes on the system. Having a look in here you should see a number of core unix commands such as ls, touch, cat and su.
- *boot* is often a separate boot partition rather than just a directory, you can check this using **lsblk**. This partition manages how the kernel is loaded when the computer boots. You should not modify anything in this folder unless you are sure that you know what you are doing, a mistake can render your operating system non-functional.
- *lib* contains common shared libraries and kernel modules, such as firmware and cryptographic functions.
- *dev* is the device folder. This contains files that communicate between the operating system and any connected devices. If you plug a usb into your Linux system you will find and be able to mount it from here.
- *root* The home folder for the root user, otherwise identical to a regular user's home folder.

Homework

There were possibly quite a few new concepts and commands in this tutorial. Without practice it's likely that you will forget them all by next week.

Your homework is this; stop using a file browser. Spend the next week navigating and launching programs exclusively with the terminal while you familiarise yourself with the environment and learn the commands that will be relevant on a day to day basis.

There are many more useful commands out there, that you will hopefully learn and accumulate over time.