Part I

Linux basics

Suggested reading

- The World Wide Web has a huge amount of information about Linux and how to use it.
- A lot of it is more-or-less accurate.
- Some of it may be wrong.
- Some of it may be unhelpful.
- Some may be dangerous (to your data or spare time)

Linux is probably best picked up as you go along, but it is helpful to have some idea of how things can fit together or what is (im)possible.

History

- Strictly speaking, Linux is only the kernel (interface between user applications and hardware)
- More properly, you will be using GNU/Linux, an operating system made up of a collection of free (as in speech and as in beer) software packages which use Linux as the kernel.
- Even more properly, you will be using a particular distribution which consists of GNU/Linux plus some other software which makes up a complete user-friendly operating system.
- GNU stands for GNU's Not Unix, and was developed as a collection of free software to replace Unix, developed at AT&T's Bell Labs.

For brevity I shall (incorrectly) talk about Linux throughout. This nomenclature is common.

Distributions

There are many distributions of Linux, specialised for different purposes. Largely, if you're used to one, then you will be able to quickly adapt to another.
Major distributions include

- Ubuntu (from Canonical) used in CSC. Probably the most user-friendly.
- Red-Hat Linux: Commercial distribution of Linux suitable for large companies and networks who need paid support
- Fedora: Based on Red-Hat and freely available
- Scientific Linux: Based on Red-Hat and put together at CERN. Includes packages suitable for scientific use.
- Debian: Distribution on which Ubuntu is based.

GUI/Windows

For basic use, Linux has drop-in replacements for most Windows/Mac software. Examples are:

- Firefox - Web browser
- Thunderbird - Email client
- Libreoffice - Office suite (Documents, Presentations, Spreadsheets)
- GIMP (GNU Image Manipulation Package) - Paint/Photopaint equivalent
- Inkscape - Vector drawing package

If you really need high-performing graphics or office software, then you will probably need to pay for it, but the above should serve most needs.

Unix philosophy

Most of what I shall discuss about Linux is based on the command line, where you type simple commands to be carried out by the OS. The Unix philosophy is:

> *Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*

So, Linux programs generally work in very similar ways to each other, can be combined and linked to form larger programs, and are designed to do one thing and to do it well.

Basics

- You can open a terminal (icon is usually a black screen with a cursor in it) in Linux and see a prompt such as:

  `pmb39@apollo:~$`

- This says that the user `pmb39` is on the computer called `apollo`, is currently in his home directory `~`,
  and can type instructions after the $.
- This prompt can be changed, but this initial setup is common. I shall usually abbreviate it to $

Directories

To list the contents of the current directory:

```
$ ls
Documents Downloads Music Videos
```

To change directory:

```
$ cd Documents
```

To see what directory you're currently in: (print working directory)

```
$ pwd
/home/raid/pmb39/Documents
```

More directories

To create a directory:

```
$ mkdir MPhil
$ cd MPhil
$ mkdir MiniProject_1
```

or

```
$ mkdir -p MPhil/MiniProject_1
$ cd MPhil/MiniProject_1
```

will make both directories at once, even if `MPhil` doesn't exist initially.

Linux directory structure

- The Linux directory structure starts from "root" labelled /
- Directory names are separated using the / character.
- Directories form a tree structure.
- Your home directory is probably `/home/pmb39` or similar.
- Executables (or binaries) can be found in `/usr/bin`
- Libraries of functions used by many programs can be found in `/usr/lib`
- Other useful directories may be available depending on your system setup.
- For example, `/opt` may contain different versions of programs, such as more up-to-date compilers, libraries, etc. `/usr/local` may be used for the same purpose.
- In CSC we use `/lsc/opt` to store these.

Linux files

Essentially everything in Linux is a file, and represented by a string of bytes. To see what a file contains:

```
$ cat myFile
Hello.  This is a test file!
$
```

man pages

All commands in Linux are documented in the "man" pages:

```
$ man ls
$ man man
```

(Try these; see how many ways there are to list a directory.)

Use the arrow keys to navigate, and type / to start searching.

Also, C-functions are documented:

```
$ man atan2
```

Command-line tips

- Use arrow keys to move the cursor left and right.
- Up and down arrows will take you back to earlier commands which can be edited as necessary.
- Pressing TAB will attempt to autocomplete the current command or directory you are typing.
- Pressing TAB twice will list all possible completions.

```
$ cat Ex [Press TAB twice]
Examples/ Exercise_1.tex Exercise_1.tex~ Exercise_2.tex
Exercise_2.tex~
$ cat Exe [Press TAB once]
$ cat Exercise_
```

Text editors

- `gedit` - Equivalent of Notepad - simple to use
- `emacs` - Can be used as a simple text-editor, but very powerful
- `vi/vim` - Non-GUI editors, very powerful but have a steep learning curve
- `geany` - Closer to an IDE than a plain text-editor
- `atom` - Free IDE from GitHub, but read the licence agreement first
- `sublime` - GUI editor with some powerful multi-select and context-aware editing - per-user licence $80, but free to evaluate.
- `VSCode` - Visual Studio Code - Very powerful code-editor from Microsoft. Free but with telemetry/user-tracking.

Tip: Never claim that one text-editor is better than all others. Wars have been fought over less.

Your identity

- Use `whoami` to check your username.
- Use `groups` to check which groups you are a member of.
- Use `passwd` to change your password.

Multi-user systems

- By nature, Linux is a multi-user OS, so many people may be using a computer at any one time.
- Use `who` to see who is currently logged in (and from where).
- Use `last` to see who logged in most recently.
- To see what programs are taking most processing time, use `top` (press 'q' to quit).
- Play nicely; you should not use up resources (CPU time, memory, disk I/O) to others' detriment, so:
- Use `nice -19` in front of long-running commands to allow a local user's commands to take priority.

Stdin, stdout and stderr

Every program in Linux has three standard file handles:
By default:

- stdin (standard-in) takes input from the terminal and puts it into the program.
- stdout (standard-out) is ordinary output and is directed into the terminal.
- stderr (standard-error) is output indicating errors and is directed into the terminal.

Redirecting input/output

If you have a program which takes some user-input and produces some output:

```
$ ./myProgram
Enter simulation-type (1=Heat-equation, 2=Advection,
3=Reactive-flow):  2
Enter end-time:  3
T=0.001
T=0.002
T=0.003
...
```

you may get bored of typing the same input, and you may wish to store the output.

Redirecting input/output

- Use file-redirection to save typing:

  ```
  $ cat myInput | ./myProgram >> myOutput
  ```

  where the file `myInput` contains what you would type as input, then the standard-output will end up in `myOutput`.
- If `myOutput` does not exist, it will be created; if it exists, the output will be appended to it.
- To overwrite the contents of an existing file, use:
  `$ cat myInput | ./myProgram > myOutput`
- To redirect standard-error, use `2> myErrors`
- To combine stdout and stderr, use `>& myOutputAndErrors`
- To direct stdout and stderr to different files, use
  `2> myErrors > myOutput`

Piping

- The | symbol in the previous slide is the "pipe" symbol.
  A useful way of thinking of multiple commands combined is that the output of one is piped into another, and this can be continued as necessary.
- The `tee` command allows you to tee-off (like a tee-junction) output into another file:
  `./myProgram |& tee myProgram.out`
  will allow you to see the output from your program in the terminal whilst simultaneously storing it to the file `myProgram.out`.
- The extra & indicates that both standard-out and standard-error will be piped into `myProgram.out`

Simple commands

Linux has many very simple tools:

- `cat`: concatenate one or more files and display to stdout
- `sort`: Sort a set of lines
- `uniq`: Remove duplicate adjacent lines
- `spell`: From text provided to stdin, or from a file, return a list of incorrectly spelled words

Complex operations from small components

- How can I spell-check my written assignment report?
  `spell Project_1.tex`
  returns a long list of mis-spelled words.
- How can we remove duplicates?
  `spell Project_1.tex | sort | uniq`
- Omitting `sort` would only remove *adjacent* duplicate words.
  `spell Project_1.tex | sort | uniq >> Project_1_typos`
- Each tool does a simple thing, but combined they can do a very wide range of things.
- In practice, you might use `detex Project_1.tex | spell ...` to remove LaTeX commands from the file first.

Other commands

- `rm`: Removes a file `rm ./MyThesis.tex`
  For a directory: `rm -rf ./AllMyCode/` (recursive, force)
- `mv`: Move a file: `mv Thesis.tex Chapter1.tex`
- `cp`: Copy a file: `cp Thesis.tex Thesis_old.tex`
  For a directory: `cp -r Thesis/ Thesis_old/` (recursive)
- Wildcards: `rm *.pdf` Remove all files ending in `.pdf`

  `mv simOutput?.png ./Pictures/`
  `?` matches a single character, so `simOutput13.png` will not be moved.
- Use `ls` to check what files will be affected if using wildcards.

Even more useful commands

- `grep`: Search a file or directory for a word or phrase:
  ```
  $ grep -r "FIXME" ./MPhil_SourceCode
  ```
- `find`: Find files:
  ```
  $ find ./MPhil -name "MyPlot.png"
  ```
- Note the lack of consistency about whether the directory being searched is placed at the start or the end.
- `diff`: Compare the contents of two files:
  ```
  $ diff Thesis_1.tex Thesis_2.tex
  2011c2096
  < references, and output files.
  ---
  > documentation, and other output files.
  ```
- As ever, more information is available by using `man`.

File permissions

As with any OS, there are some files you have permission to view and edit, and some you don't. If you run:

```
$ ls -l
total 570408
-rw-r--r--  1 pmb39 csc 9628 Jan 24  2013 AirBubble.hdf
drwxr-x---  2 pmb39 csc 4096 Nov 13  2013 CAD_data
-rw-r--r--  1 pmb39 csc 3372 Aug  8  2013 cornerShock.hdf
-rw-r--r--  1 pmb39 csc 8290 Apr 11  2013 gcc-4.7.3.tar.bz2
drwxr-xr-x  2 pmb39 csc 4096 Jun  5 11:56 GrandCanyon
-rw-r-----  1 pmb39 csc 2632 Jun  5 16:35 ideal_SodProblem
drwxr-x---  3 pmb39 csc 4096 May 20 17:43 output
```

then the contents of the first few columns show what permissions are needed to access the file.

Permissions detail

A file has both an owner and a group. For example, `pmb39` is a normal user, and is a member of the `csc` group.
For each of user, group, and other, permissions as to whether they can read, write, or execute the file are given.

- `-rw-r--r-- pmb39 csc` indicates that `pmb39` can both read and write the file, and anyone else can read it.
- `-rwxr-xr-x root root` indicates that the `root` user can write to the file, and everyone can read and execute it.
- `drwxr-xr-x pmb39 csc` indicates that this is a directory and `pmb39` can write to it, and everyone else can read it and execute (descend into) it.

Changing permissions

- In order to change the permissions given to a file, use `chmod`
- This is followed by `[ugo]` `[+-]` `[rwx]`:
  - `u` - user
  - `g` - group
  - `o` - other
- So, to remove read permissions from everyone but yourself, use `chmod go-r ./myFile`
- To give yourself execute permissions, use `chmod u+x ./myFile`

Changing file group

- You may be a member of multiple groups:
  ```
  pmb39@apollo:~$ groups
  csc netadmin www-admin
  ```
- You can change the group to which a file belongs by using
  `chgrp www-data ./myFile.html`

Symbolic links

- Occasionally, you may wish to make a file or directory available in more than once place, but any changes to one version need to be mirrored in the other.
- The easiest way to accomplish this is via a symbolic link:

```
$ ln -s /data/apollo/pmb39 /home/pmb39/apollo_data
$ ls -l /home/pmb39/apollo_data
lrwxrwxrwx 1 pmb39 csc 19 Aug 26 15:27 apollo_data
                                    -> /data/apollo/pmb39
```

- This effectively redirects references to `/home/pmb39/apollo_data` to `/data/apollo/pmb39`.
- It does *not* create a copy.

Permissions defaults

- On the CSC network, all files you create are by default readable by you and anyone else in the `csc-mphil` group (but no-one else), but only writable by you.
- Some programs (e.g. Firefox) may further restrict any files they create to be only readable by you.
- The defaults may well be different on other systems.
- Take care to check before entering sensitive data.

Extensions or lack thereof

- Often, Linux does not rely on file extensions (`.txt`, `.exe`, `.doc`) as much as Windows.
- Extensions may be used by some programs for ease of use, e.g. (`.doc`, `.png`, etc.)
- Generally, extensions are merely another way of labelling a file, not a way of forcing what can be done with it.
- Use `file` to give an indication of what is in a file:

```
$ file ShockBubbleCollapse_80us.png
ShockBubbleCollapse_80us.png:  PNG image data, 1024 x
937, 8-bit/color RGB, non-interlaced
$ file /usr/bin/gcc-5
/usr/bin/gcc-5:  ELF 64-bit LSB executable, x86-64,
```

- There is nothing to stop you doing `cat /usr/bin/gcc-5` but this will very likely mess up your terminal.