Imperial College London

UNIVERSITY OF OXFORD

24 May 2024

# Introduction to Unix and Project Organisation

Brendan Martin

## Overview of Sessions

Names: Efthymios, Andrea, Alex, Brendan

1. Day 1
   - Intro to the shell
   - Project organisation: directory structure and code layout
   - Version control: git and interactive GitHub setup
   - Short task and questions
2. Day 2
   - Conda
   - Latex and Overleaf
   - Reproducibility: seeding, logbook and notebook best practices
   - Configuration files
   - Making things faster: why do it + optimisation + profiling
   - Workshop + questions

## Outline

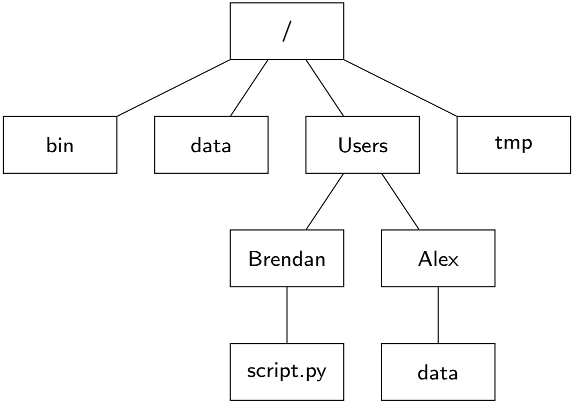
1. Introduction to the Unix shell

2. Project organisation and management

## Why should you learn the shell?

1. Because so much else depends on it. e.g.
    - Installing software
    - Running Programs
    - Version control
    - Controlling remote machines
2. It allows you to delegate repetitive tasks to your computer. e.g.
    - Naming/renaming multiple files
    - Moving copying files between directories
    - Searching for files and data within files
3. Reproducibility. Automate your entire workflow into a pipeline using bash scripts.

## Navigating Files and Directories



- Directories and files are stored in a tree-like structure
- The root of the tree is denoted / in Linux and C:\in Windows
- To check where you are in the tree: `pwd`
- To list the files in your current directory: `ls`

## Path examples

The location of a particular file or directory within the file system is called its *path*

Examples:

`/Users/Brendan/script.py`

`/Users/Alex/data`

If my `pwd` is `/Users/Brendan/` and I type `ls`, the output will be:

## Path examples

The location of a particular file or directory within the file system is called its *path*

Examples:

`/Users/Brendan/script.py`

`/Users/Alex/data`

If my `pwd` is `/Users/Brendan/` and I type `ls`, the output will be: `script.py`

## Path examples

The location of a particular file or directory within the file system is called its *path*

Examples:

`/Users/Brendan/script.py`

`/Users/Alex/data`

If my `pwd` is `/Users/Brendan/` and I type `ls`, the output will be: `script.py`

If my `pwd` is `/Users/` and I type `ls`, the output will be:

## Path examples

The location of a particular file or directory within the file system is called its *path*

Examples:

`/Users/Brendan/script.py`

`/Users/Alex/data`

If my `pwd` is `/Users/Brendan/` and I type `ls` , the output will be: `script.py`

If my `pwd` is `/Users/` and I type `ls` , the output will be: `Brendan Alex`

## Changing between directories

To change between directories: `cd /PATH/TO/DIRECTORY`

Table 1: Relative Paths

| | |
|---|---|
| Root | / |
| Current directory | . |
| One directory up the tree | .. |
| Two directories up the tree | ../.. |
| User's home directory | ~ |

Examples:

If my `pwd` is `/Users/Brendan` and I want to change to `/data` , I can either do `cd /Users/data` or `cd ../../data`

## Options

Options let you modify the command.

For example if your `pwd` is `/data` and you want to list the files in `/Users/Brendan/` without changing directories, you can use the `-F` option with `ls` :

```
ls -F /Users/Brendan/
```

If you want to list *all* files in your `pwd` *including* hidden files (that start with a `.` e.g. `.gitignore` ) then you can use

```
ls -a
```

## Working with files and directories

- mkdir MyDir: Make a new directory called MyDir
- mv /PATH/TO/MY/FILE /PATH/TO/NEW/LOCATION: move a file or directory from one location to another
- cp /PATH/TO/MY/FILE /PATH/TO/NEW/LOCATION: copy a file from one location to another
- cp -r /PATH/TO/MY/FOLDER /PATH/TO/NEW/LOCATION: copy a directory from one location to another
- rm /PATH/TO/MY/FILE: remove a file (permanently!)
- rm -r /PATH/TO/MY/FOLDER: remove a directory and all of its contents
- touch /PATH/TO/MY/FILE: create an empty file

## Wildcards

Used to perform actions on more than one file at a time, or to find part of a phrase in a text file.

They can be used in conjunction with almost any command (e.g. mv, cp, rm)

? this can represent any *single* character
e.g. `mv f?l.csv ..`

* this can represent any number of characters (including zero, in other words, zero or more characters)
e.g. `rm *.csv`

{} apply the command to anything matching an element within the brackets "list"
e.g. `rm {*.csv, ?.pdf}`

## Where do the executable binary files for the Linux commands live?

The commands that we have been using such as pwd, ls, mv, are executable binary files that usually live in the directory /usr/bin/.

You can check the location of any executable file associated with a given command using which
e.g.

```
which ls
which R
```

## Where do the executable binary files for the Linux commands live?

The commands that we have been using such as pwd, ls, mv, are executable binary files that usually live in the directory /usr/bin/.

You can check the location of any executable file associated with a given command using which
e.g.

```
which ls
which R
```

How does linux know where to find ls? i.e. why do we not have to always type /usr/bin/ls?

## Variables and Environment Variables

As in all programming languages, it is useful to be able to store information in the form of variables which can then be accessed later on.

You can create "shell variables" in your current shell using
MYVAR="myvarname"

MYVAR can be accessed later on using $MYVAR, for example, to print to screen, use echo $MYVAR.

## Variables and Environment Variables

As in all programming languages, it is useful to be able to store information in the form of variables which can then be accessed later on.

You can create "shell variables" in your current shell using
MYVAR="myvarname"

MYVAR can be accessed later on using $MYVAR, for example, to print to screen, use echo $MYVAR.
However, when I close my current shell, MYVAR will be forgotten.

If I want the variable to be remembered by all shells, I need to make it an "environment variable" by adding the line

export MYVAR="myvarname"

to a file called .bashrc

## How does linux know where to find ls?

Answer: There is an environment variable in your .bashrc file called PATH which contains the directories through which linux looks for an executable file called ls.

## Running Python and R scripts

To run a python script, you can write

```
/PATH/TO/python /PATH/TO/myscript.py
```

However, since the python executable file is in a directory contained in the PATH environment variable, I don't need to type out /PATH/TO/python and can instead write

```
python /PATH/TO/myscript.py
```

Usually I am in the same directory as myscript.py, and can just write

```
python myscript.py
```

Similarly, for R, I can write

```
Rscript myscript.R
```

## Adding arguments to your python command

If your python script requires you to read in some data from data1.csv, you can specify data1.csv as the first *argument* of your python command:

```
python myscript.py data1.csv
```

This is useful if you want to perform the same python script to a different dataset, data2.csv.

Rather than going into the python script and manually changing it to read in data2.csv instead of data1.csv, you can simply change the first argument of your python command to

```
python myscript.py data2.csv
```

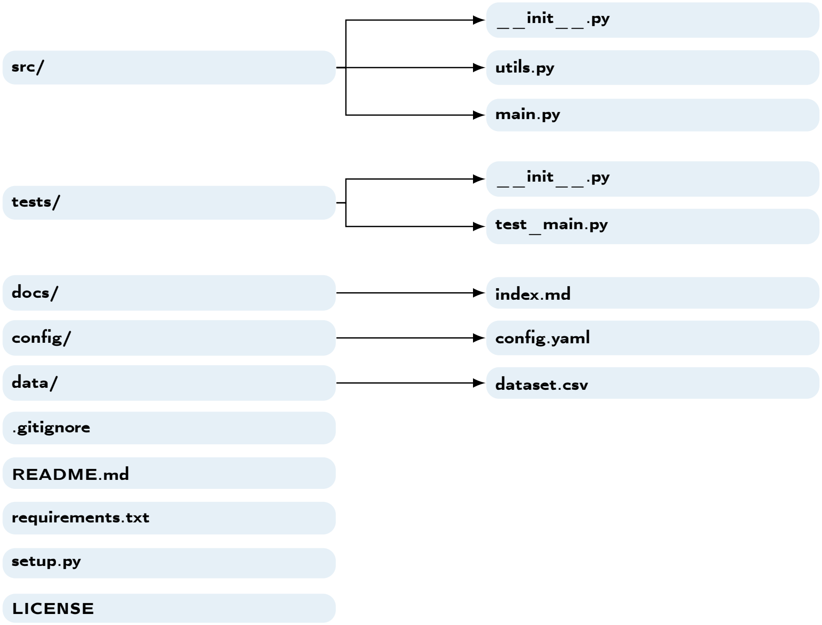
Within your python script, you can read in the first argument of the command using sys.argv[1]

## Further Topics

- Piping commands
- Loops
- Shell scripting
- Finding things using find and grep
- Building pipelines using make

Online resources: Software Carpentry
`https://software-carpentry.org/lessons/index.html`

## Project Organisation



## Project Organisation: Cookiecutter

- A python package that automates setting up your project directory structure
- Cookiecutter Data Science is particularly useful:
  `https://cookiecutter-data-science.drivendata.org/`

## Linux and git task

1. Go to `https://github.com/bmartin9/MScComputingIntro`
2. Fork the repository to your Github account
3. Clone from your Github account to your local machine
4. Create a ./data and ./src directory
5. Move all data to the data directory and all scripts to the src directory
6. Ensure you can run compute_means.py using both datasets
7. Create a .gitignore file and add ./data to it so that nothing from ./data gets pushed to Github
8. Add, commit and push your changes

THANK YOU!

## References

Software Carpentry. `https://software-carpentry.org/lessons/index.html`. Accessed: 2024-05-24.