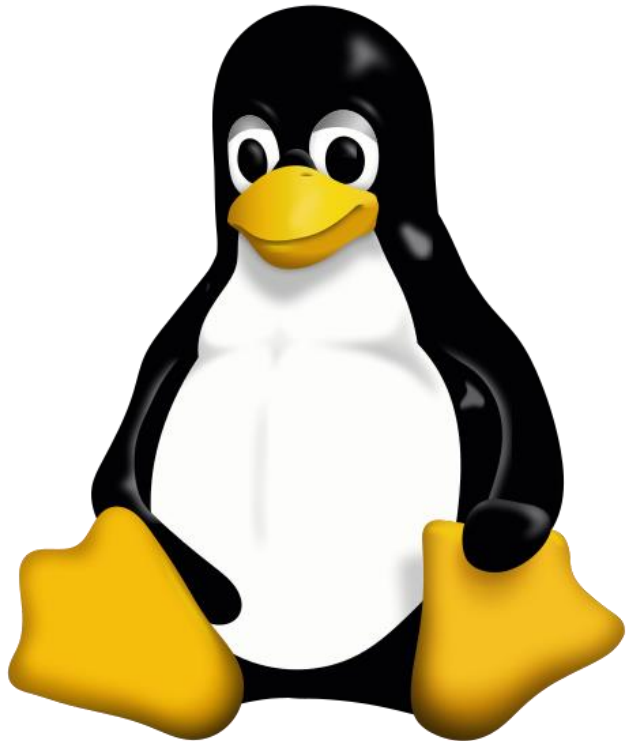
Running programs in the BASH shell

v2018-11

Babraham Bioinformatics



Running programs in Linux

- Two major methods
  - Graphical
  - Command line

- Graphical launches only work for graphical programs accessed through a graphical environment

- Most data processing will be command line based, as will most remote access
  - Graphical programs can still be launched from the command line

Shells

- A shell is a command line interpreter, used to launch software in Linux
- There are many different shells available:
  - BASH
  - CSH
  - ZSH etc.
- Most software will work the same in all shells
- Some functions and automation are different between shells

- We will use the most popular shell, BASH

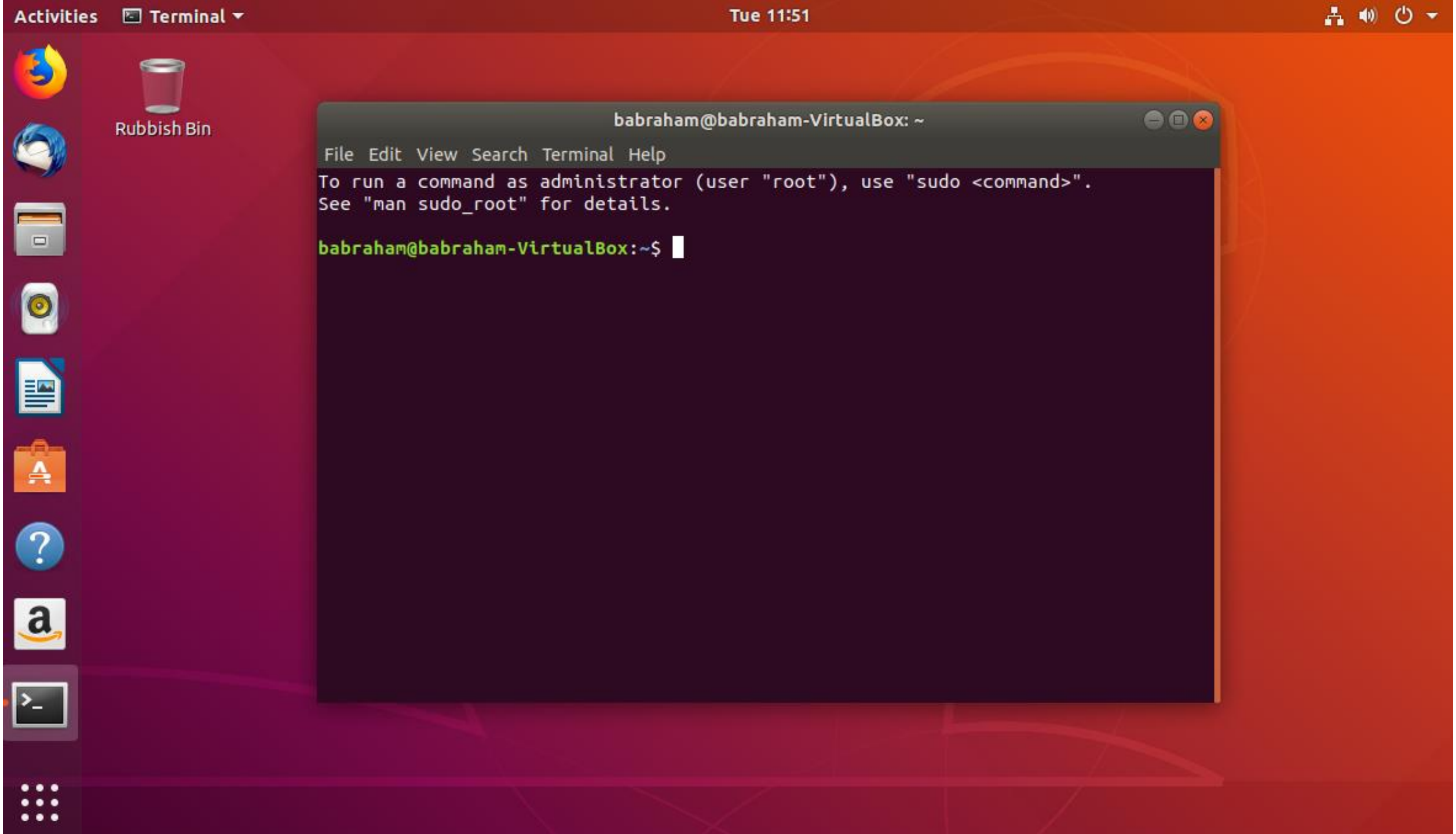
What does a shell provide

- Command line editing and construction tools (eg auto complete)
- History
- Job control
- Configuration management (startup scripts)
- Aliases

- Automation
  - Scripting language
  - Variables, functions etc

Running programs in BASH

- We will be using a graphical terminal running BASH
- Right click and select "Open terminal"
- ..or, from Applications menu find Terminal
- Once running, right click on icon and select "Add to favourites"


Activities
Terminal
Tue 11:51
Rubbish Bin
babraham@babraham-VirtualBox: ~
File Edit View Search Terminal Help
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
babraham@babraham-VirtualBox:~$

Running programs

- Type the name of the program you want to run
- Add on any options the program needs
- Press return - the program will run
- When the program ends control will return to the shell
- Run the next program!

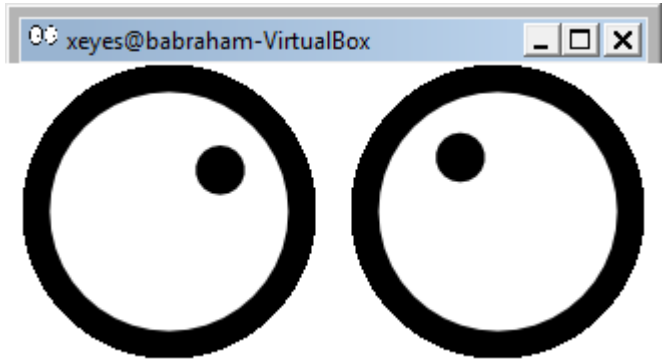
Running programs

```
babraham@babraham-VirtualBox:~$ ls
Desktop  Documents  Downloads  examples.desktop  Music
Pictures  Public  Templates  Videos

babraham@babraham-VirtualBox:~$
```

- Command prompt - you can't enter a command unless you can see this
- The command we're going to run (`ls` in this case, to list files)
- The output of the command - just text in this case

Running graphical programs

```
babraham@babraham-VirtualBox:~$ xeyes
```




```
babraham@babraham-VirtualBox:~$
```

Note that you can't enter another command until you close the program you launched

The structure of a unix command

Program name

Switches

Data
(normally files)

Each option or section is separated by spaces. Options or files with spaces in must be put in quotes.

Command line switches

- Change the behaviour of the program
- Come in two flavours (each option usually has both types available)
  - Minus plus single letter (eg `-x -c -z`)
    - Can be combined (eg `-xcz`)
  - Two minuses plus a word (eg `--extract --gzip`)
    - Can't be combined
- Some take an additional value, this can be an additional option, or use an = to separate (it's up to the program)
  - `-f somfile.txt` (specify a filename)
  - `--width=30` (specify a value)

Manual pages

- All core programs will have a manual page to document the options for the command
- Manual pages are accessible using the man program followed by the program name you want to look up.
- All manual pages have a common structure

Manual Pages (`man cat`)

```
CAT(1)                          User Commands                          CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s) to standard output.

       With no FILE, or when FILE is -, read standard input.

       -A, --show-all
              equivalent to -vET

       -n, --number
              number all output lines

       -T, --show-tabs
              display TAB characters as ^I

       --help display this help and exit

EXAMPLES
       cat f - g
              Output f's contents, then standard input, then g's contents.

       cat    Copy standard input to standard output.
```

Exercise 1

Understanding Unix File Systems

Unix File Systems

- Consists of a hierarchical set of directories (folders)
- Each directory can contain files

- No drive letters (drives can appear at arbitrary points in the file system)

- No file extensions (you can add them, but they're not required)

A simple unix filesystem

/ (Always the top of the file system)

home/ (Directory containing all home directories)

anne/

simon/

Documents/ (All names are case sensitive)

test.txt (A file we want to work with) **/home/simon/Documents/test.txt**

media/

myusb/ (A USB stick added to the system)

Drives in a Linux file system

```
$ mount
/dev/sda1 on / type ext4 (rw)
/dev/sda5 on /state/partition1 type ext4 (rw)
/dev/sda2 on /var type ext4 (rw)
Private-Cluster.local:/ifs/homes on /bi/home type nfs (rw,addr=10.99.101.5)
Private-Cluster.local:/ifs/Scratch/Scratch on /bi/scratch type nfs (rw, addr=10.99.101.5)
Private-Cluster.local:/ifs/Group on /bi/group type nfs (rw, addr=10.99.101.4)
Private-Cluster.local:/ifs/apps-nfs on /bi/apps type nfs (rw, addr=10.99.101.5)
central-cluster.local:/ifs/bioinformatics/pubcache on /bi/pubcache type nfs (rw, addr=10.99.102.7)
central-cluster.local:/ifs/bioinformatics/seqfac on /bi/seqfac/seqfac type nfs (ro, addr=10.99.102.6)
/dev/fuse on /bi/sequencing type fuse (ro,nosuid,nodev,allow_other)
```

Specifying file paths

- Some shortcuts
  - ~ (tilde, just left of the return key) - the current user's home directory
  - . (single dot) - the current directory
  - .. (double dot) - the directory immediately above the current directory

Specifying file paths

- Absolute paths from the top of the file system
  - `/home/simon/Documents/Course/some_file.txt`

- Relative paths from whichever directory you are currently in
  - If I'm in `/home/simon/Documents/Course/`
  - `../../Data/big_data.csv`
    - is the same as `/home/simon/Data/big_data.csv`

- Paths using the home shortcut
  - `~/Documents/Course/some_file.txt` will work for user simon anywhere on the system

Command line completion

- Most errors in commands are typing errors in either program names or file paths
- Shells (ie BASH) can help with this by offering to complete path names for you
- Command line completion is achieved by typing a partial path and then pressing the TAB key (to the left of Q)

Command line completion

Actual files in a folder:

```
Desktop
Documents
Downloads
examples.desktop
Music
Pictures
Public
Templates
Videos
```

If I type the following and press tab:

`De` [TAB] will complete to `Desktop` as it is the only option

`T` [TAB] will complete to `Templates` as it is the only option

`Do` [TAB] will no nothing (just beep) as it is ambiguous

`Do` [TAB] [TAB] will show `Documents` and `Downloads` since those are the only options

`Do` [TAB] [TAB] `c` [TAB] will complete to `Documents`

You should ALWAYS use TAB completion to fill in paths for locations which exist so you can't make typing mistakes

(it obviously won't work for output files though)

Wildcards

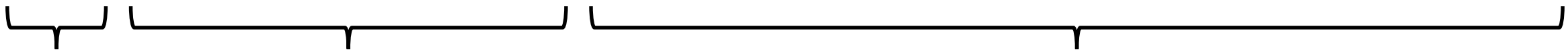
- Another function provided by your shell (not your application)
- A quick way to be able to specify multiple related file paths in a single operation
- There are two main wildcards
  - `*` = Any number of any characters
  - `?` = One of any character
- You can include them at any point in a file path and the shell will expand them before passing them on to the program
- Multiple wildcards can be in the same path.
- Command line completion won't work after the first wildcard

Wildcard examples

```
$ ls Monday/*txt
Monday/mon_1.txt  Monday/mon_2.txt  Monday/mon_3.txt  Monday/mon_500.txt

$ ls Monday/mon_?.txt
Monday/mon_1.txt  Monday/mon_2.txt  Monday/mon_3.txt

$ ls */*txt
Friday/fri_1.txt  Monday/mon_1.txt  Monday/mon_3.txt    Tuesday/tue_1.txt
Friday/fri_2.txt  Monday/mon_2.txt  Monday/mon_500.txt  Tuesday/tue_2.txt

$ ls */*1.txt
Friday/fri_1.txt  Monday/mon_1.txt  Tuesday/tue_1.txt
```

The structure of a Unix command

```
ls  -ltd --reverse D*
```



Program name

Switches

Data
(normally files)

Each option or section is separated by spaces. Options or files with spaces in must be put in quotes.

Programs for manipulating files

Manipulating files

- You will spend a lot of time managing files on a Linux system.
    - Viewing files (normally text files)
    - Editing text files
    - Moving or renaming files
    - Copying files
    - Deleting files
    - Finding files

Viewing Files

- Simplest solution
  - `cat` Sends the entire contents of a file (or multiple files) to the screen.

- Quick look
  - `head` or `tail` will look at the start/end of a file
    - `head -10 [file]`
    - `tail -20 [file]`

- More scalable solution
  - `less` is a 'pager' program, sends output to the screen one page at a time

    - Return / j = move down one line
    - k = move up one line
    - Space = move down one page
    - b = go back one page
    - /[term] = search for [term] in the file
    - q = quit back to the command prompt

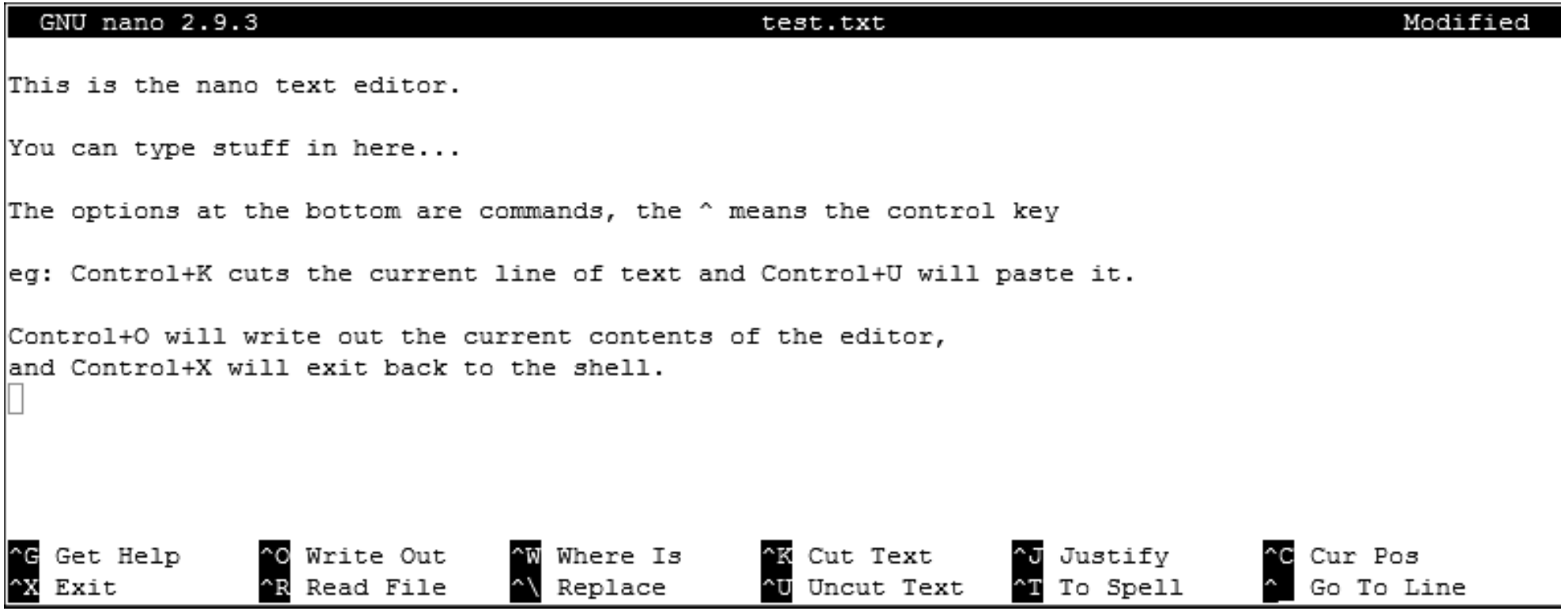
`less -S` (no wrapping)

Editing files

- Lots of text editors exist, both graphical and command line
- Many have special functionality for specific content (C, HTML etc)
- Technically, only the `vi` editor is required to be present in POSIX
- `nano` is a simple command line editor which is nearly always present
- You can try several until you find one you like

Using nano to edit text files

- `nano [filename]` (edits if file exists, creates if it doesn't)



Moving / Renaming files

- Uses the `mv` command for both (renaming is just moving from one name to another)
- `mv [file or directory] [new name/location]`
- If new name is a directory then the file is moved there with its existing name
- Moving a directory moves all of its contents as well
- Examples
  - `mv old.txt new.txt`
  - `mv old.txt ../Saved/`
  - `mv old.txt ../Saved/new.txt`
  - `mv ../Saved/old.txt .`

Copying files

- Uses the `cp` command
- `cp [file] [new file]`
- Operates on a single file
- Can copy directories using recursive copy (`cp -r`)

- Examples
  - `cp old.txt new.txt`
  - `cp old.txt ../Saved/`
  - `cp old.txt ../Saved/new.txt`
  - `cp ../Saved/old.txt .`
  - `cp -R ../Saved ./NewDir`
  - `cp -R ../Saved ./ExistingDir/` (only if ExistingDir exists)

Linking rather than copying

- Copy duplicates the data in a file
  - Can be a problem with big data files
  - Can be a problem if you want edits to propagate
- Links are a way to do 'virtual' copies. You can make the same file appear in more than one place.
- Two types of link
  - Hard link - completely indistinguishable from a copied file
    - Only works on the same disk (can't span file systems)
    - Only works for files (not directories)
  - Symbolic (or soft) link
    - Is visible as a link, but can still be used as if it is a file
    - Can work across file systems
    - Can be used for directories as well as files
    - Normally the safe option to use

Creating symbolic links

- Use the `ln` command with the `-s` modifier
- Usage `ln -s [from] [to]`
- Exactly the same structure as `mv`

- When you list a link you can see where it points, but you can use it like a file

```
$ cat test.txt
This is a test file

$ ln -s test.txt test2.txt
$ ls -l test2.txt
lrwxrwxrwx 1 babraham babraham 8 Sep 11 16:27 test2.txt -> test.txt

$ cat test2.txt
This is a test file
```

Deleting files

- Linux has no undo.
- Deleting files has no recycle bin.
- Linux will not ask you "are you sure"

- Files can be deleted with the `rm` command
- Directories (and all of their contents) can be deleted with `rm -r`

- Examples
  - `rm test_file.txt test_file2.txt`
  - `rm *.txt` (be VERY careful using wildcards. Always run ls first to see what will go)
  - `rm -r Old_directory/`

Finding files

- From the command line there are two main mechanisms
  - Using locate
    - Uses a pre-built index, so is *VERY* quick
    - Needs to be installed and configured (isn't on all distributions)
    - Only works on local drives - not network shares
  - Using find
    - Searches fresh each time
    - Can be slow if you're searching large numbers of files
    - Requires no pre-processing

Using find

- `find [options/filters] [place to start] [action]`

- Options / Filters
  - `-name thisfile.txt`
  - `-name *txt`
  - `-type f -empty`

- Action
  - `-print`
  - `-exec` (any command - include `{}` for filename - need to finish with `\;`)

Find examples

- `find ~ -name *fastq.gz -print`
- `find /etc/ -name *conf -exec wc -l '{}' \;`
- `find ~ -type l -print`

Exercise 2

Finding programs to run

- How does the system know which executable file to run when you just supply a name (like `ls` in this example) rather than a file path?
- What happens when you have two different programs with the same name?

The `PATH` environment variable

- Two new concepts which will be reused a lot:

1. A search path is a list of directories which the shell will iterate over to try to find something (a program in this case). It is an ordered list and the first hit will be returned.

2. Shells use "environment variables" as a small scale storage and configuration system. They are a set of key/value pairs which provide named pieces of data which influence the behaviour of the shell or programs which it runs. Some environment variables are created by default, but you can modify these, or add your own.

The `PATH` environment variable

- You can see the current PATH contents by running:
  - `echo $PATH`
  - `/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin`
- These are the directories which are trusted to contain good programs which you can launch just using their name
- Other programs can be launched using their location (rather than name)
- You can find the location of a named program using `which`
  - `which ls` (produces `/bin/ls` which is where the `ls` program is)

Setting the `PATH`

- `PATH` is an environment variable, you can set it the same as any other variable, using the export function
- `export PATH=/one/folder:/another/folder`
- Normally you want to just add to the existing `PATH`, so you can include the current `$PATH` in the definition
- `export PATH=/new/folder/:$PATH`
- You can write this into `~/.bashrc` to make the change permanent.

Exercise 3