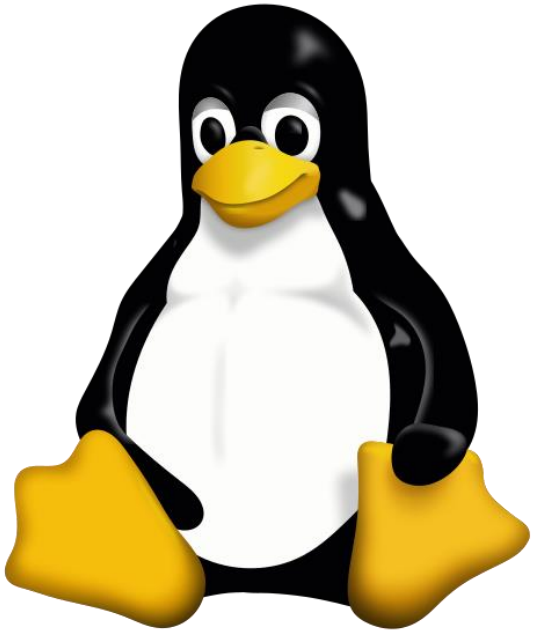
An Introduction to Unix

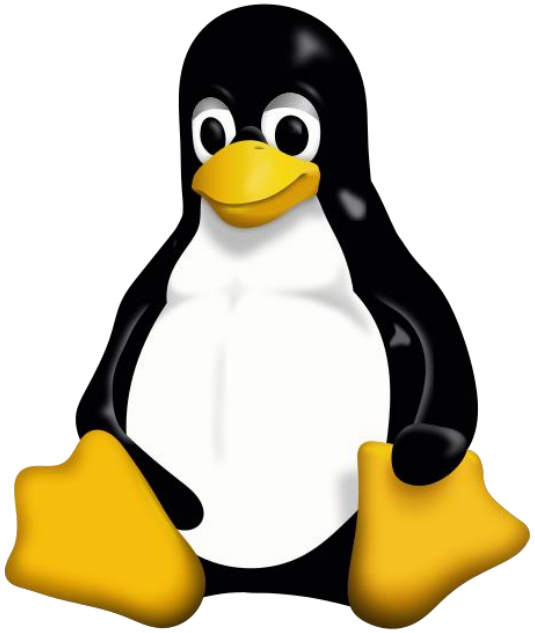
Sarah Inglesfield, Simon Andrews

v2024-08

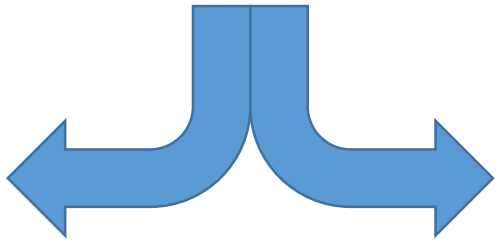


Babraham Bioinformatics

Terminology and Distributions



Babraham Bioinformatics



- Admin tools
- Bundled Software
- Support duration / cost

debian
ubuntu

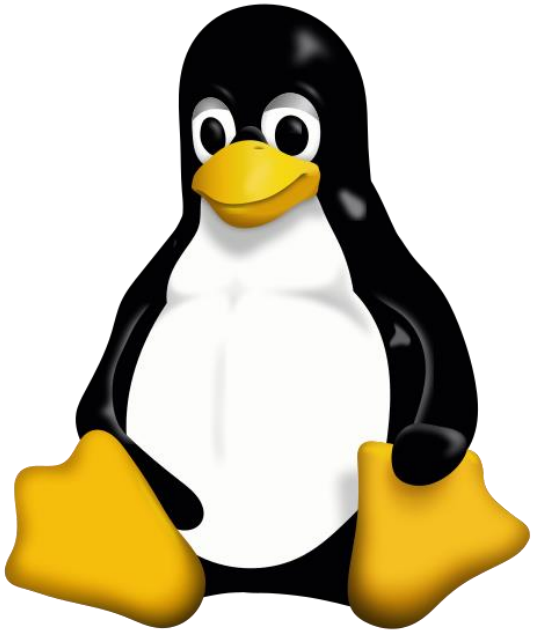
redhat®
CentOS
AlmaLinux
Rocky Linux™

Types of Linux installation

Bare Metal 	• Physical hardware • CD / DVD / USB / Network installation • Can be physically accessible (desktop) or remote (server / cluster)
Virtual Machine 	• Runs within another operating system • Portable / disposable • Install from ISO / Network
Cloud 	• Virtual machine on someone else's hardware • Amazon / Google are the main providers • Range of available hardware

Connecting to Linux Installations

Babraham Bioinformatics



Local vs Remote Connections

Local Linux Machine

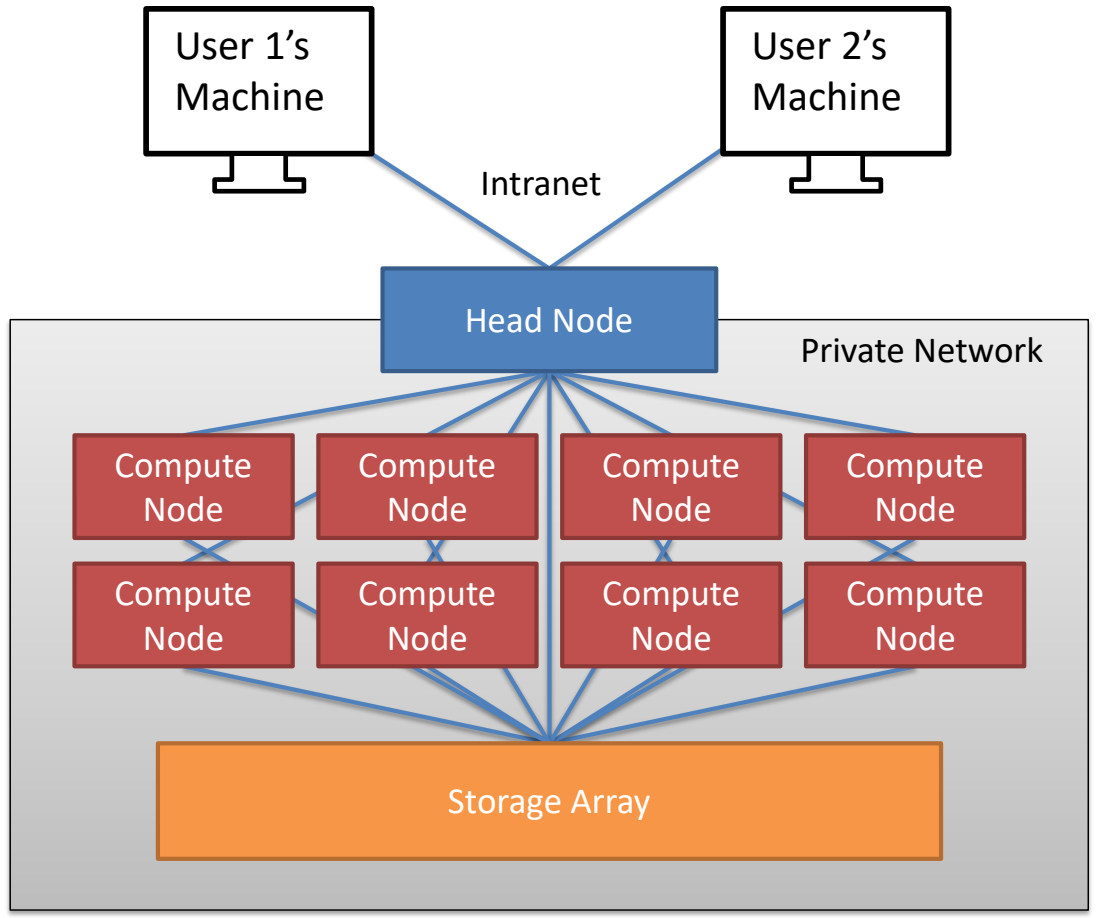
username:
password:

Remote Linux Machine

username:
password:

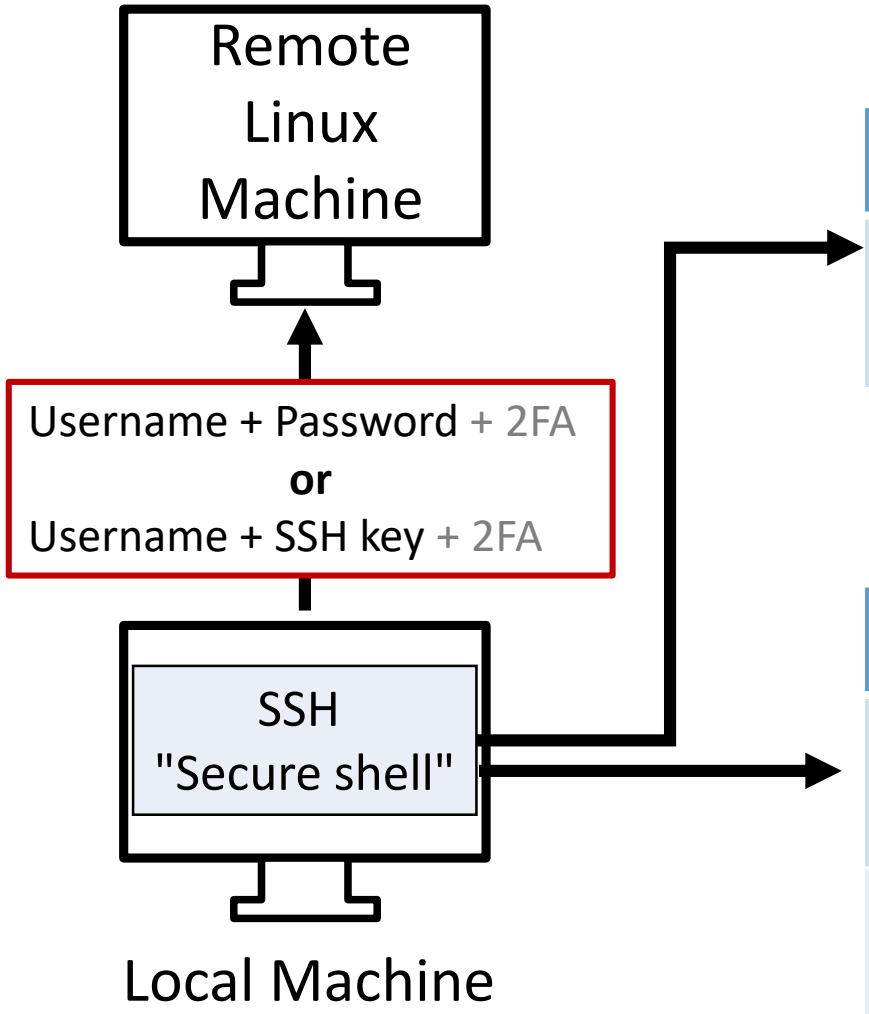
?

Local Machine



e.g. working on clusters

Connecting to a remote Linux installation

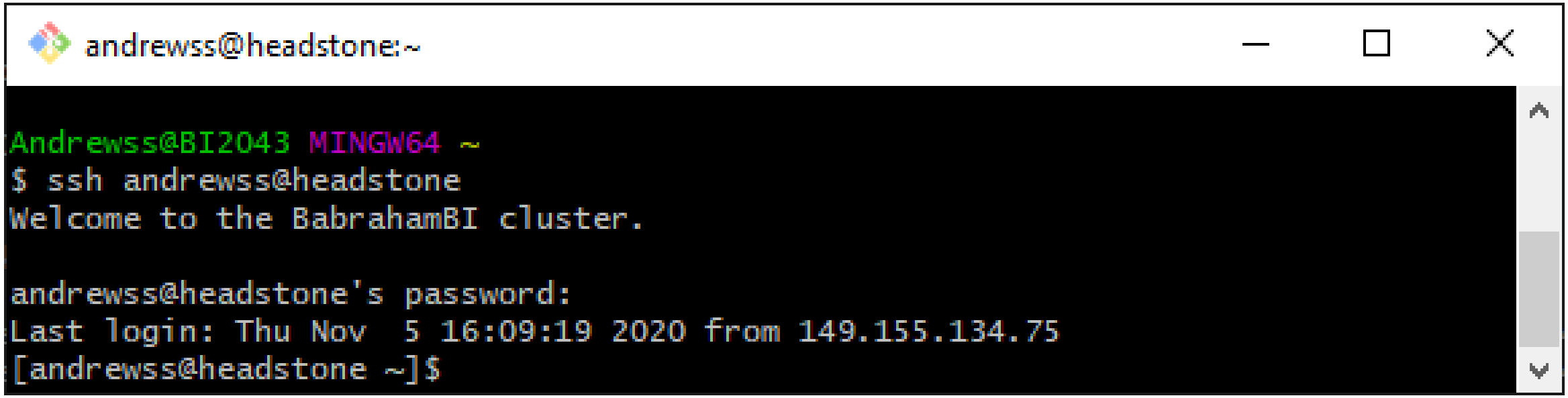


What?
• Encrypted connection • Text based interface

How?	
OSX or Linux	Use the terminal program which comes with the OS
Windows	Git Bash (https://gitforwindows.org/) PuTTY (https://www.putty.org/)

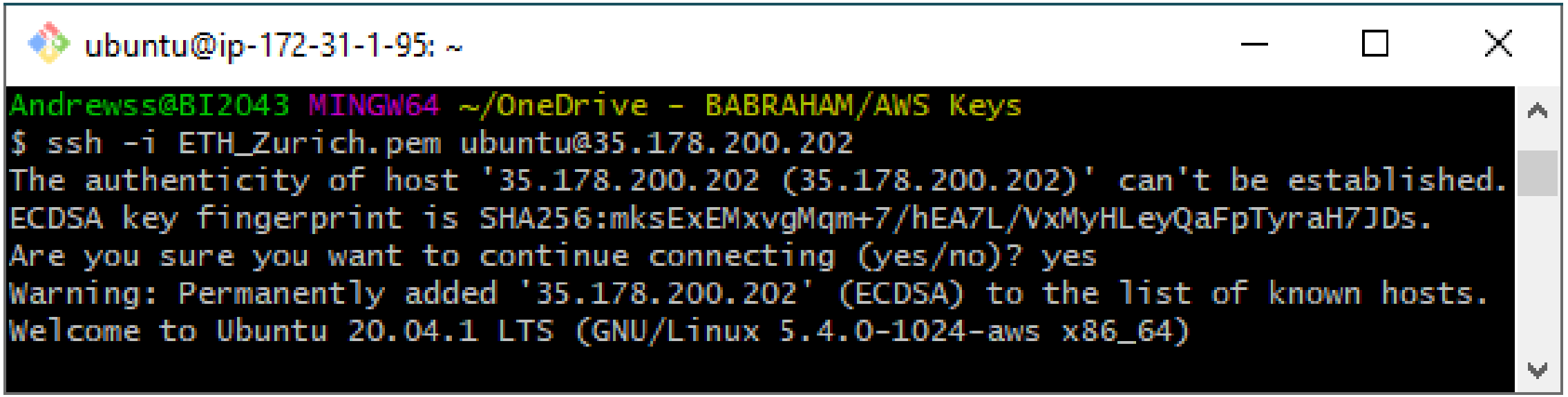
SSH + Password connection

- `ssh username@server.address`
- `[Will be promoted for password]`

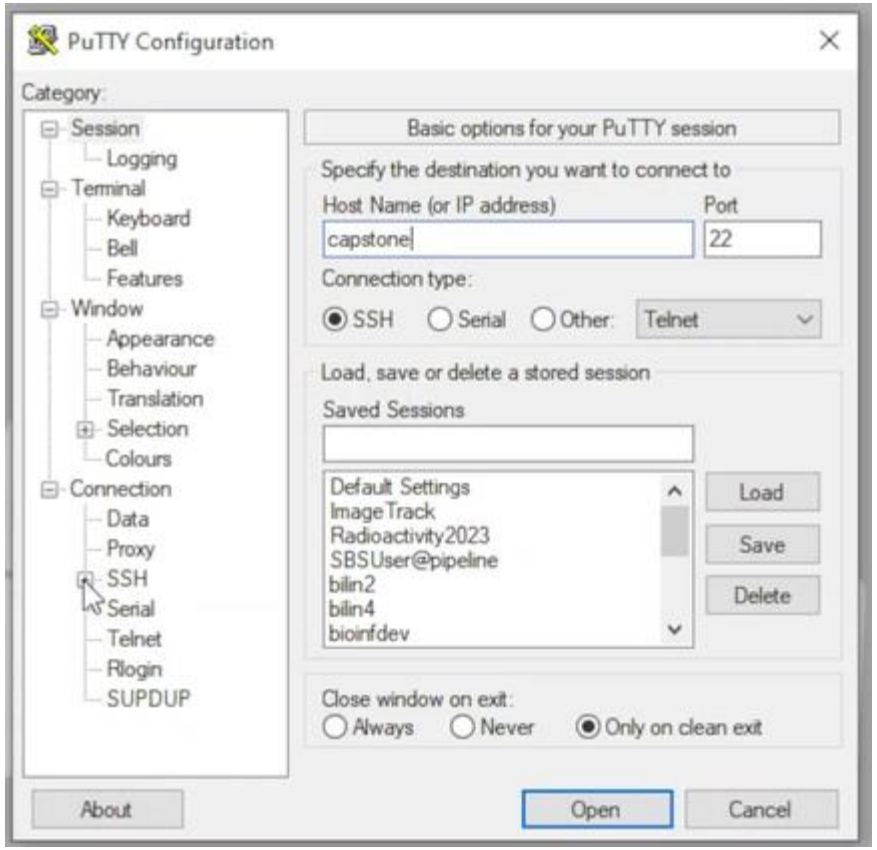


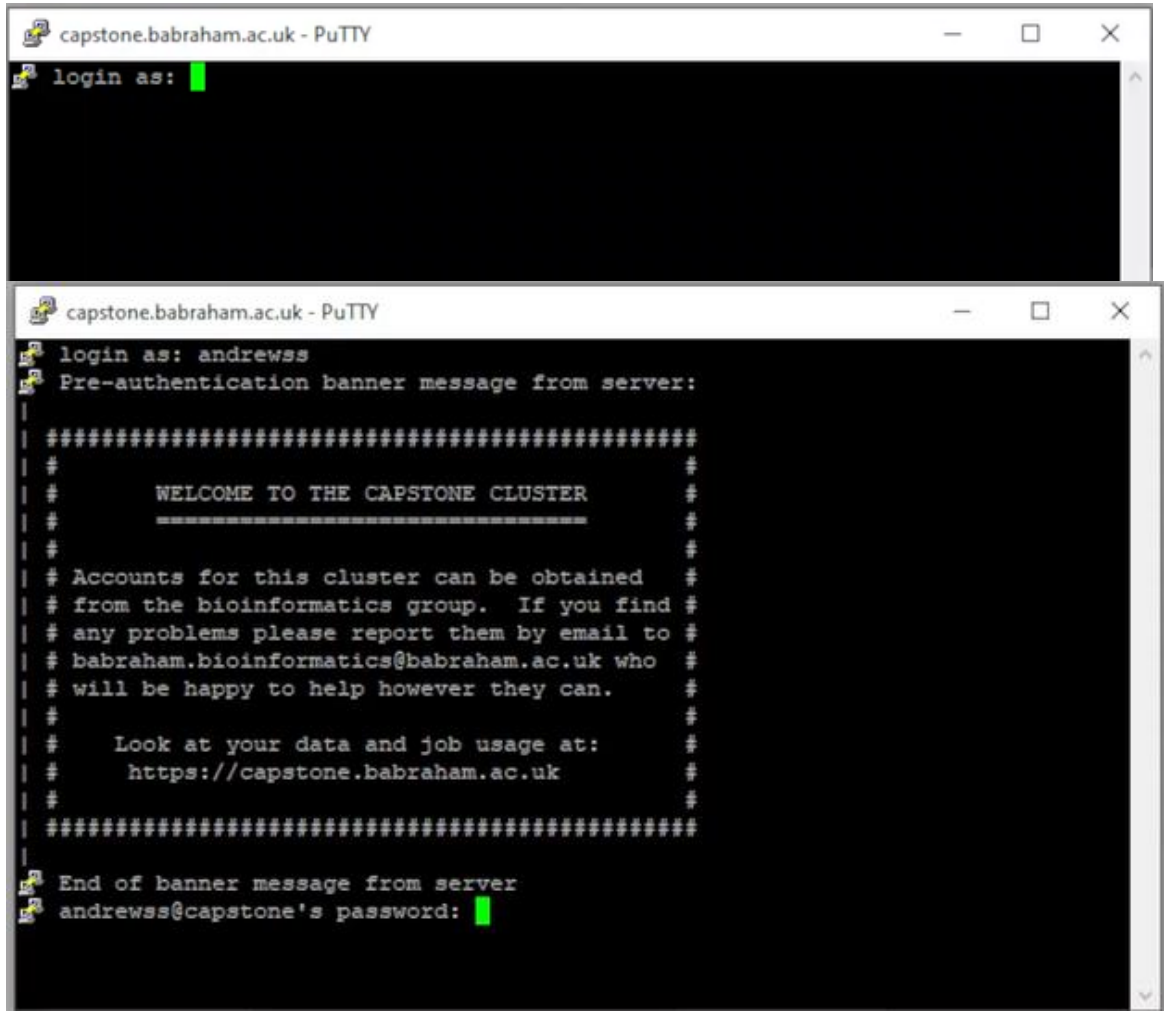
SSH + Key connection

- `ssh -i [key_file.pem] username@server.address`

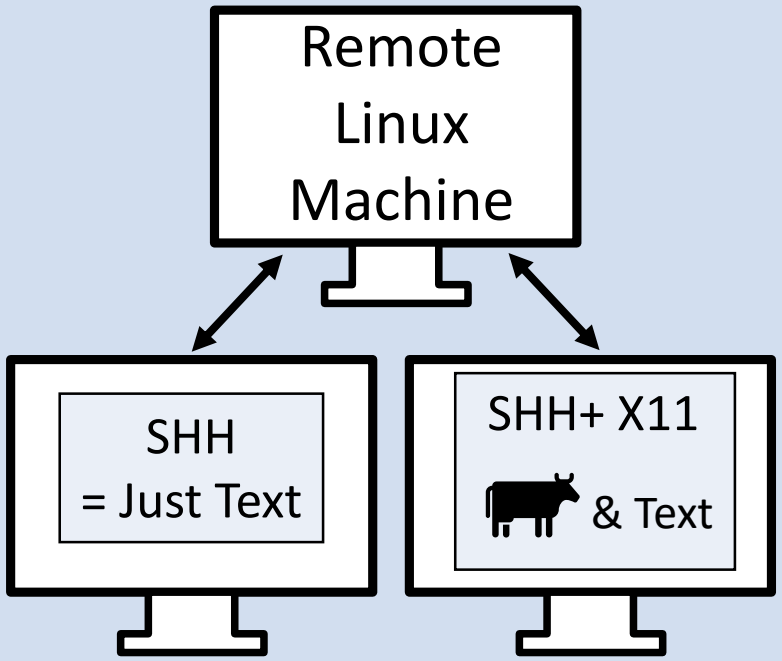


SHH Using PuTTY





SHH with Graphical Connections

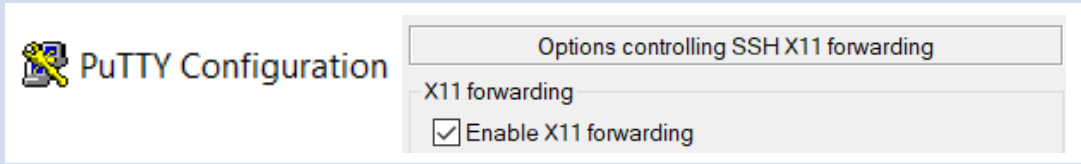


Single application windows

- X11
- Sits on top of SSH

https://www.xquartz.org/
https://sourceforge.net/projects/vcxsrv/

```
ssh -YC -i [key_file.pem] username@server.address
```



REALVNC

Apache Guacamole™

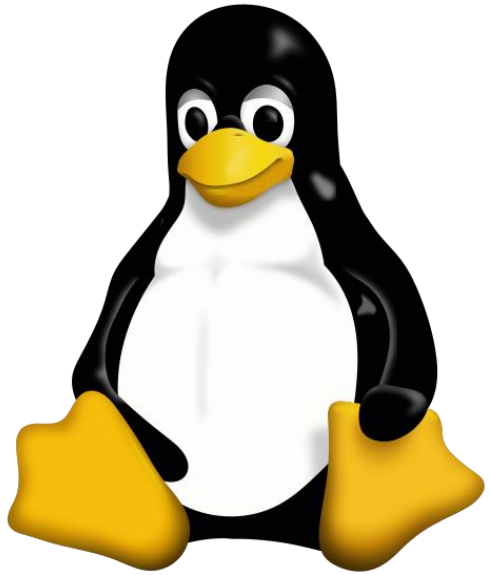
Virtual Desktop

- VNC
- Stand alone application or
- Browser based desktop

Exercise 1

Running programs in the BASH shell

Babraham Bioinformatics



Launching programs in Linux

Two major methods:

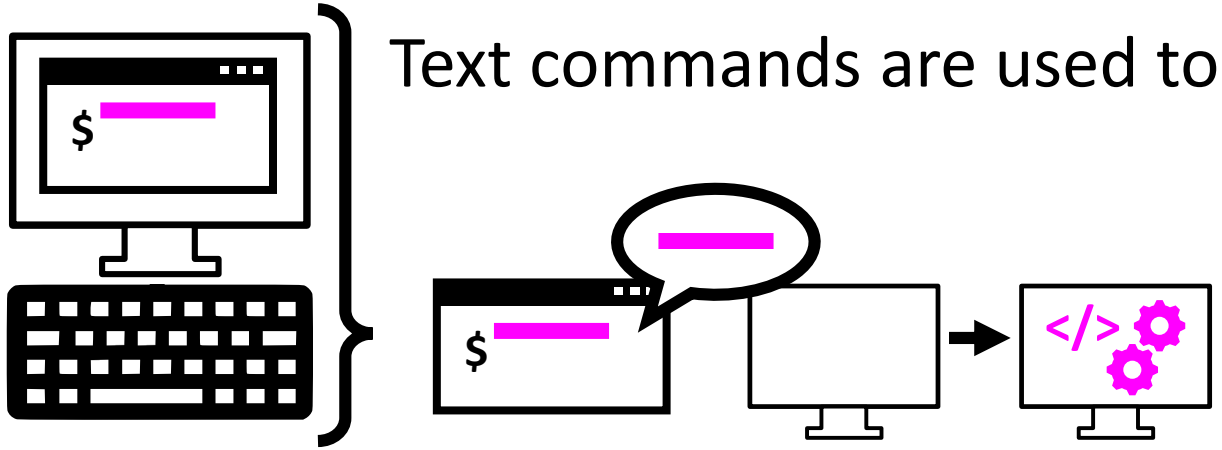
	Graphical	Command Line
Requires?	Full Graphical Environment e.g. virtual desktop	Command Line Linux Environment
How to Launch a program?	Click an icon	Type commands into an interpreter
Works for: Graphical Programs Non-Graphical Programs		

Most data processing and remote access will be command line based

For this we need an interpreter….

Shells

A shell is a command line interpreter, used to launch software in Linux



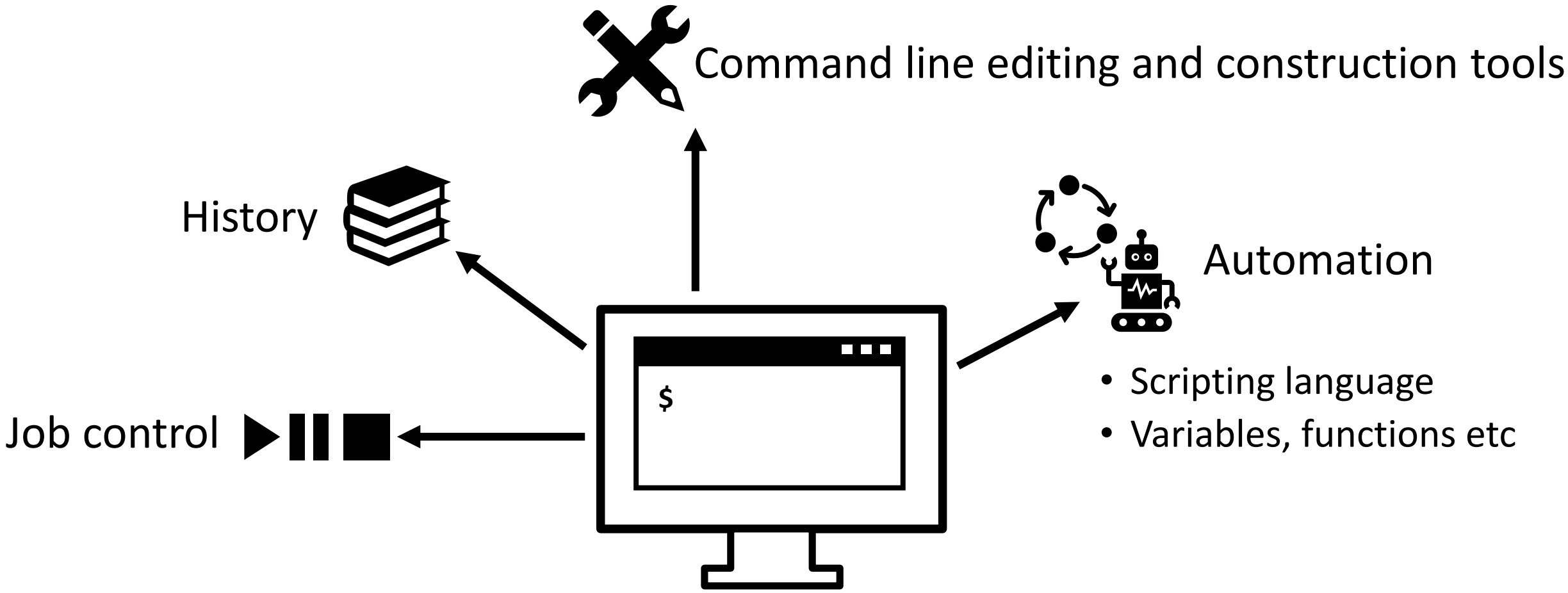
Text commands are used to launch programs

Many different shells available:

- Largely similar in how they launch programs
- Differ in some of their automation/ other clever functions

We will use the most popular shell: **BASH**

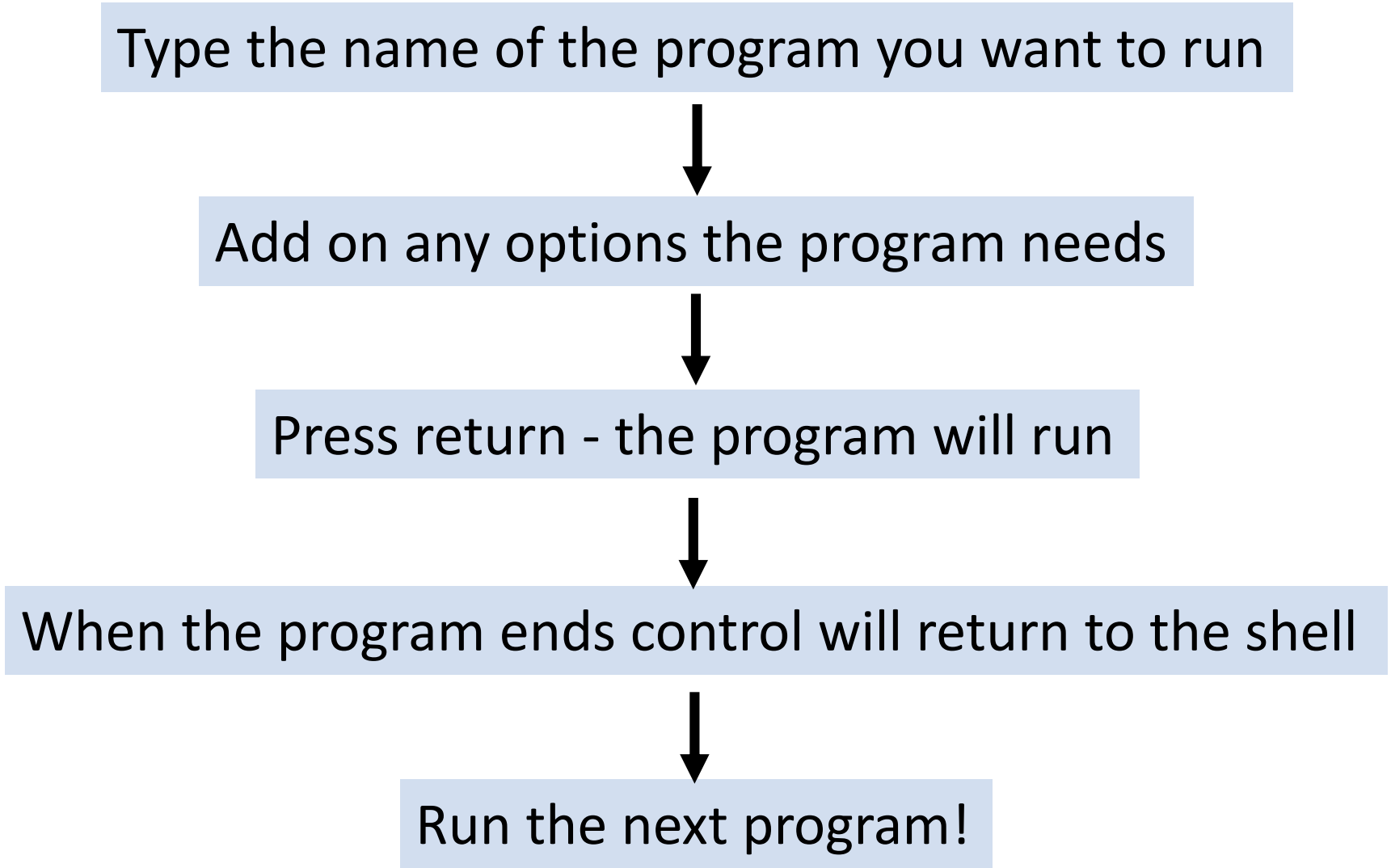
What Does a Shell Provide?



What does a Shell look like?

We will be using a graphical terminal running BASH

Running programs



Running programs

```
student@ip1-2-3-4:~$ ls
Desktop  Documents  Downloads  examples.desktop  Music
Pictures  Public  Templates  Videos

student@ip1-2-3-4:~$
```

- 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

```
student@ip1-2-3-4:~$ xeyes
```

```
student@ip1-2-3-4:~$
```

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

To change the behaviour of the program must write the appropriate switch

- Different options are represented by short and/or long forms (usually both)

Short Form	Long Form
Minus plus single letter `-x -c -z` Can be combined `-xcz`	Two minuses plus a word `--extract --gzip` Can't be combined

- Switches can be binary (on/off) or take an additional value

Binary (on/off)	+ Additional Value

Switch alone specifies the behaviour
`--gzip`
	An additional value is provided after the switch
	`-f somfile.txt (specify a filename)`
	`--width=30 (specify a value)`
	Use a [space] or = to separate

Figuring Out Options…

Programmes usually come with documentation for their options and usage

Core Programs	Non-Core Programs
Included with the install	Additional installs e.g analysis tools
Manual page (always)	Help Page (usually)
`man [program]`	`[program] --help (or -h)`

These pages all follow a very similar structure…

Manual Pages vs Help Pages

```
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

       -b, --number-nonblank
              number nonempty output lines, overrides -n

       -e     equivalent to -vE

       -E, --show-ends
              display $ at end of each line

       -n, --number
              number all output lines

       -s, --squeeze-blank
              suppress repeated empty output lines

       -t     equivalent to -vT

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

       -u     (ignored)

       -v, --show-nonprinting
              use ^ and M- notation, except for LFD and TAB

       --help display this help and exit

       --version
              output version information and exit

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

Name

Synopsis

Description

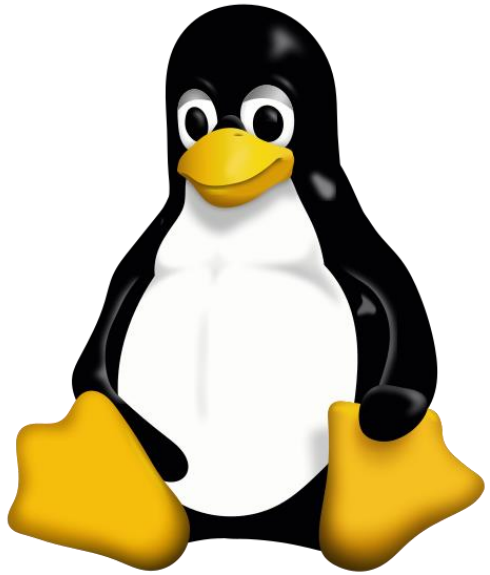
Examples

```
          FastQC - A high throughput sequence QC analysis tool

SYNOPSIS

      fastqc seqfile1 seqfile2 .. seqfileN

   fastqc [-o output dir] [--(no)extract] [-f fastq|bam|sam]
          [-c contaminant file] seqfile1 .. seqfileN

DESCRIPTION

   FastQC reads a set of sequence files and produces from each one a quality
   control report consisting of a number of different modules, each one of
   which will help to identify a different potential type of problem in your
   data.

   If no files to process are specified on the command line then the program
   will start as an interactive graphical application.  If files are provided
   on the command line then the program will run with no user interaction
   required.  In this mode it is suitable for inclusion into a standardised
   analysis pipeline.

   The options for the program as as follows:

   -h --help       Print this help file and exit

   -v --version    Print the version of the program and exit

   -o --outdir     Create all output files in the specified output directory.
                   Please note that this directory must exist as the program
                   will not create it.  If this option is not set then the
                   output file for each sequence file is created in the same
                   directory as the sequence file which was processed.
```

Exercise 2

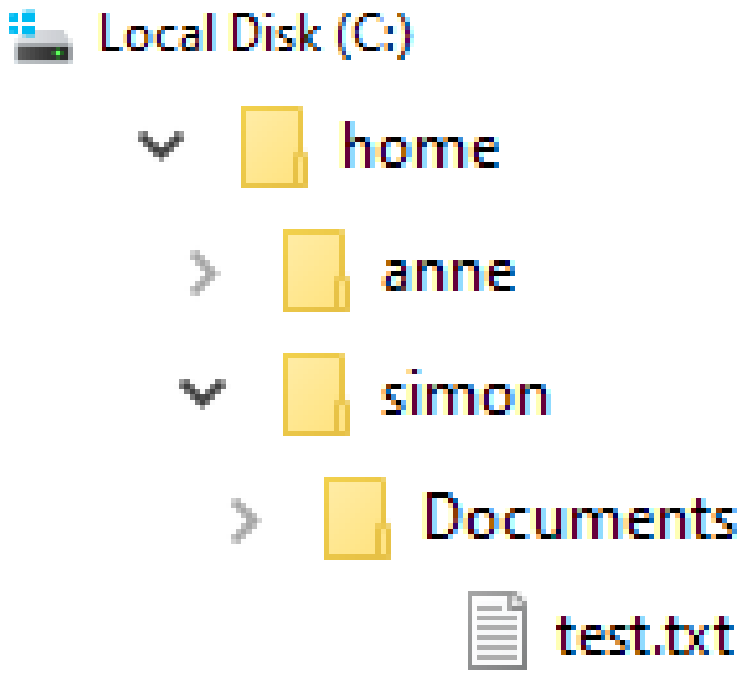
Understanding Unix File Systems

Babraham Bioinformatics



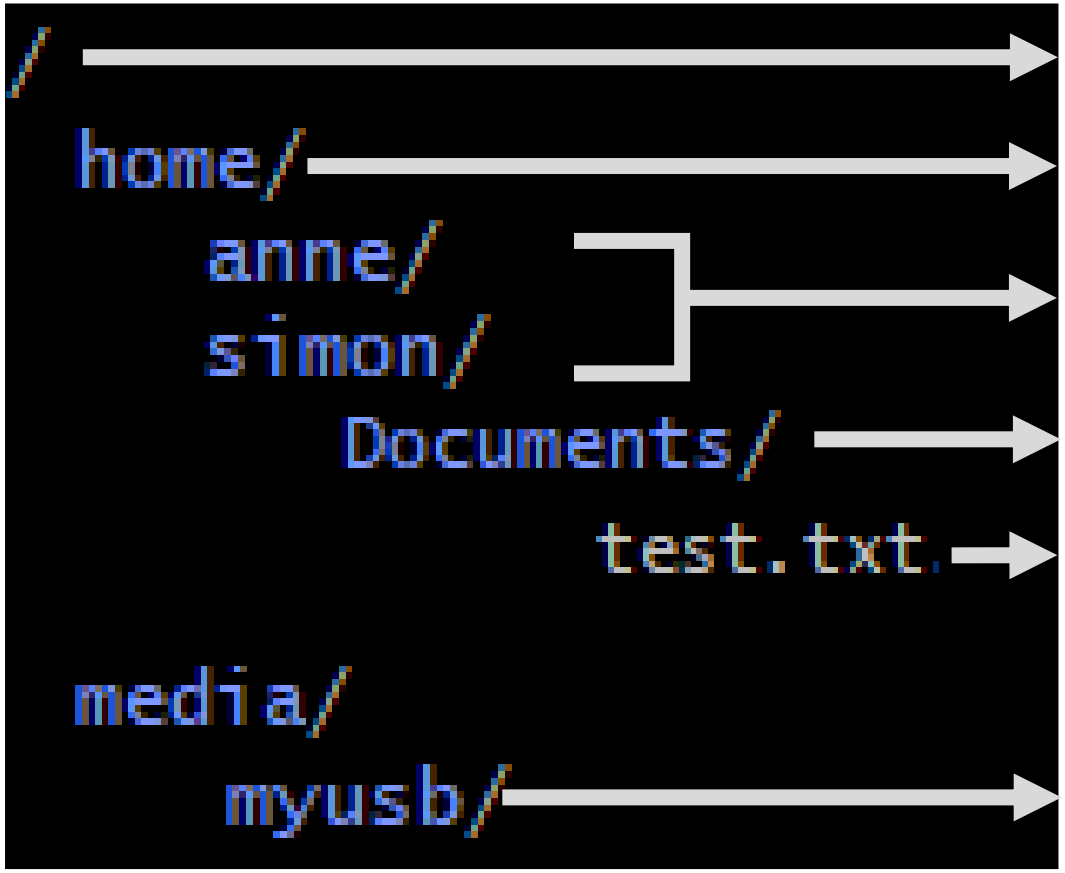
Unix File Systems vs Other File Systems

A Familiar Picture…



Standard OS File System	Same in Unix?
Hierarchical Directories	☑
Each Directory can contain files	☑
Use drive Letters	☒
Need file extensions e.g. .txt	☒

A Simple Unix Filesystem



= Root Directory = Always the top of the file system

= Directory containing all users home directories

= Directory containing all users home directories

= A Directory – note names are case sensitive

= A text file we want to work with

= A USB stick added to the system

How do we write this in our shell? = Path	`$ ls /home/simon/Documents/test.txt`

Navigating The File System

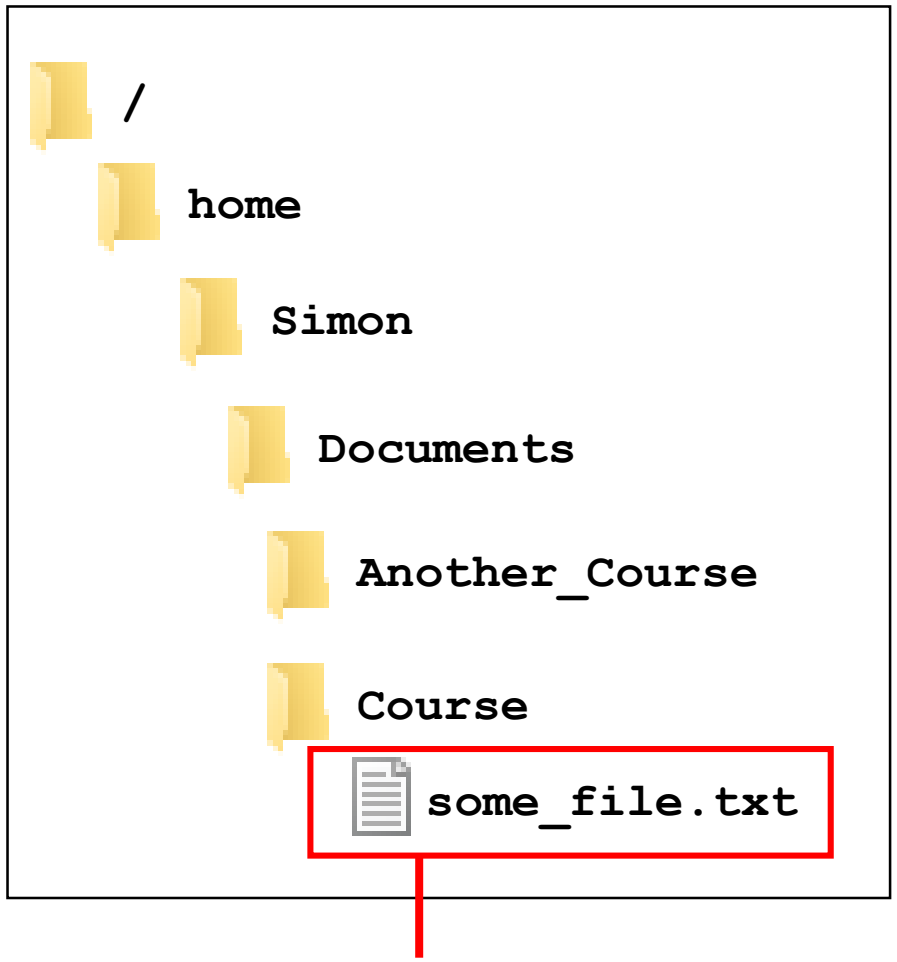
- Every Unix session has a '**working directory**'
  - This is a folder where the shell looks for file paths
- Your initial working directory will normally be your home directory (eg `/home/user`)
- There are some useful commands to help navigate the system:

Task	Command
What is my current working directory?	**pwd**
I want to make a new directory	**mkdir [name of directory to make]**
I want to move into a different directory	**cd [location to move to]**
I want to go home	**cd**

Navigating The File System – An Example

```
[andrewss@server ~]$ pwd
/home/andrewss

[andrewss@server ~]$ mkdir Simon

[andrewss@server ~]$ cd Simon
[andrewss@server Simon]$ pwd
/home/andrewss/Simon

[andrewss@server Simon]$ cd
[andrewss@server ~]$ pwd
/home/andrewss
```

Specifying File Paths



How can we refer to this file?

Options:

1. Absolute paths from the top of the file system e.g.

   `/home/simon/Documents/Course/some_file.txt`

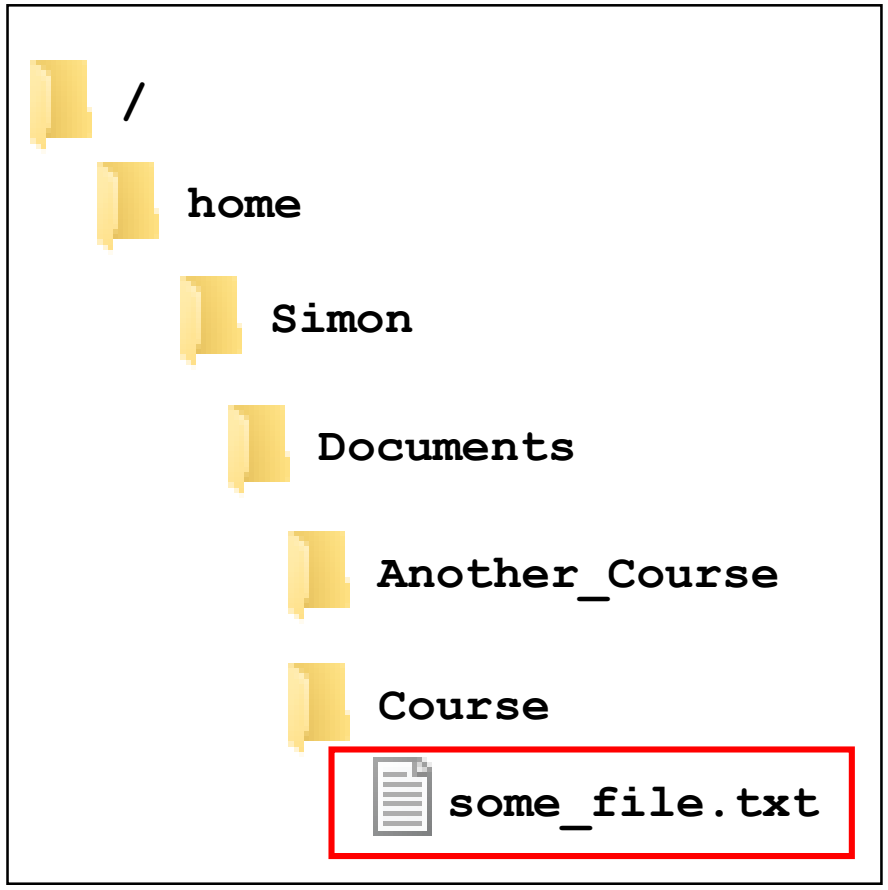
2. Relative paths from your current directory e.g.

   if we are in Course = `some_file.txt`

   if we are in Documents = `Course/some_file.txt`

3. Paths using typing shortcuts

Specifying file paths - Shortcuts

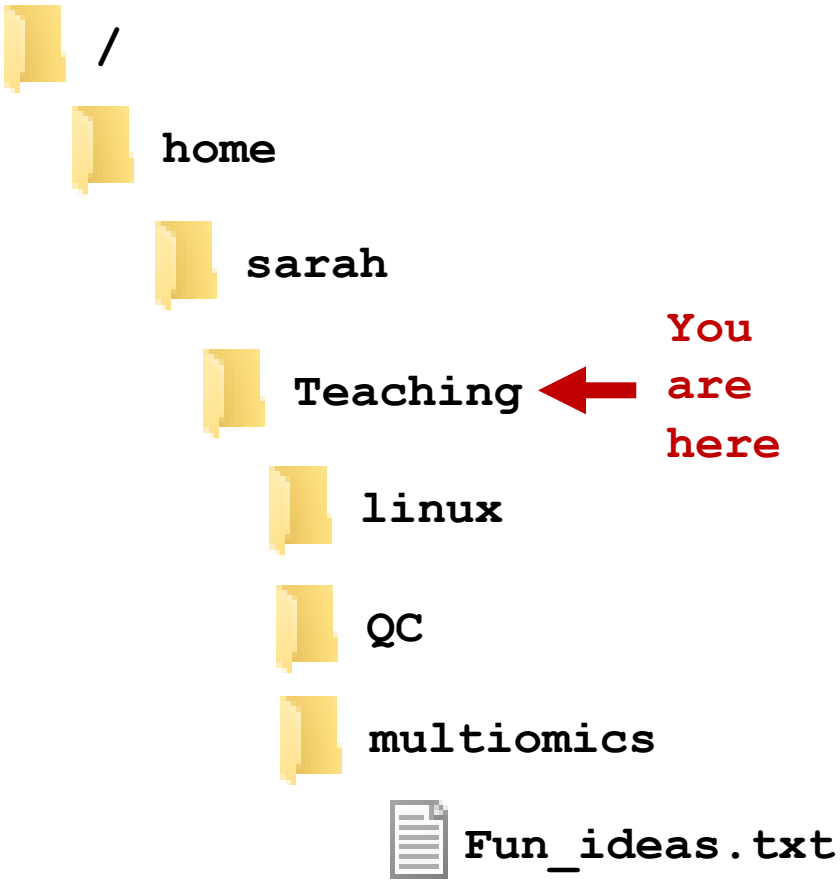


Some Useful Shortcuts:	
~	The current user's home directory
.	The current directory
..	The directory immediately above the current directory

If we were in **Another_Course**

```
~/Documents/Course/some_file.txt
../Course/some_file.txt
```

Specifying File Paths – Question:



Which Path (or Paths!) will specify my "Fun_ideas.txt"?

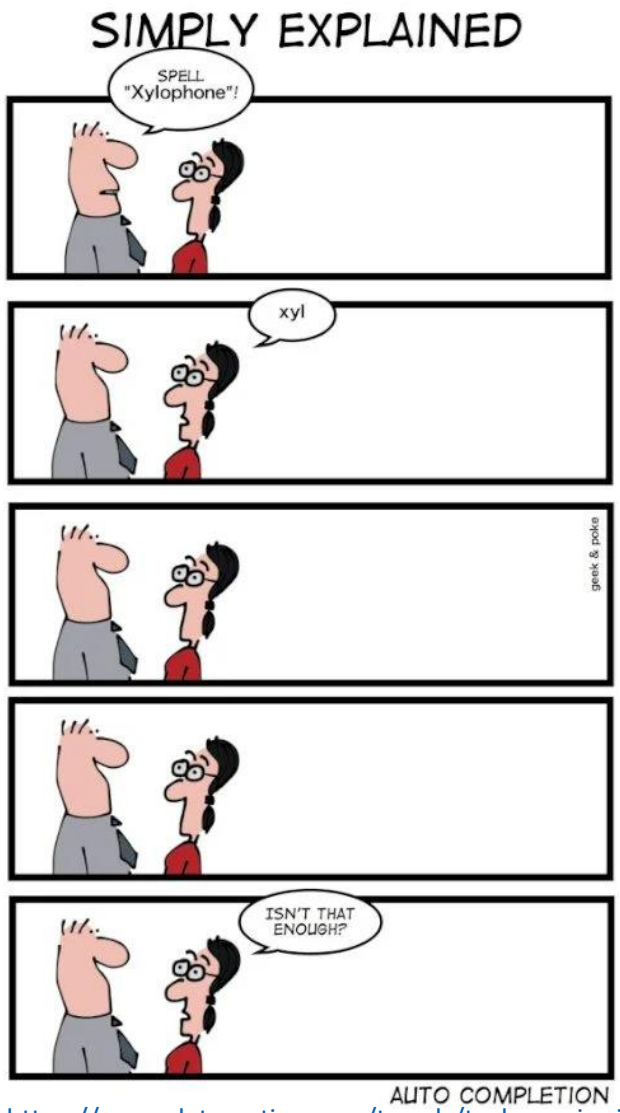
A /home/sarah/teaching/multiomics/`Fun_ideas.txt`

B ~/Teaching/multiomics/`Fun_ideas.txt`

C multiomics/`Fun_ideas.txt`

It's easy to make mistakes when typing paths

Command line completion…



https://www.datamation.com/trends/tech-comics-is-auto-complete-enough/

…Is Basically the shell's version of Autocomplete

- Most errors in commands are typing errors in either program names or file paths
- Shells (ie BASH) can help by completing paths for us

How?

Type a partial path then press the TAB key

Hooray for the TAB Key!

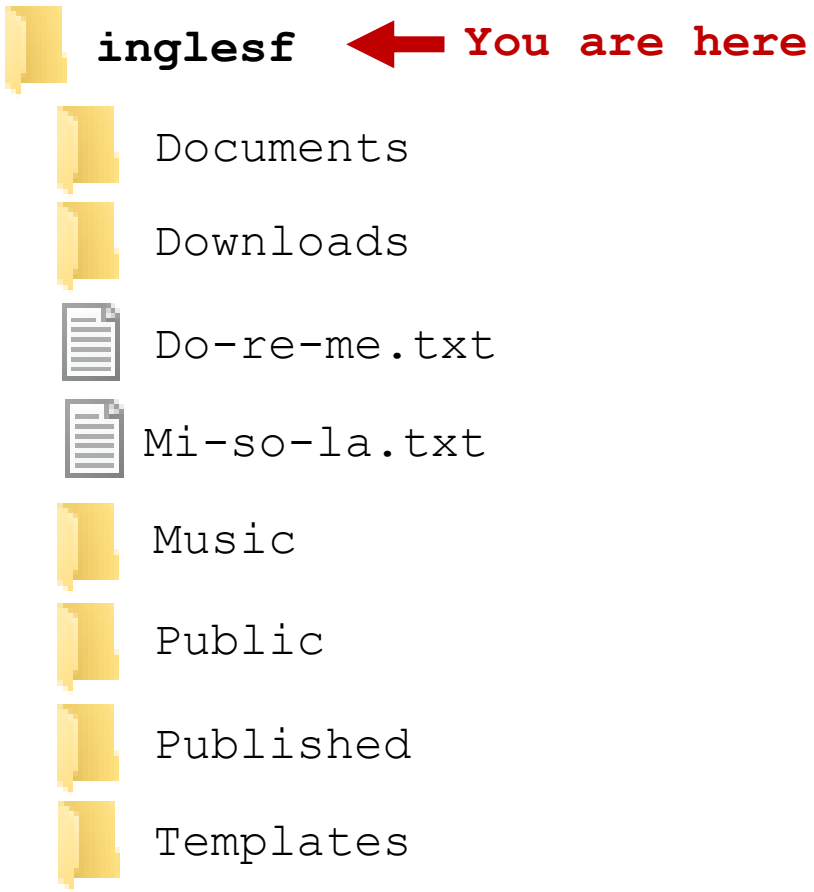
Command line completion- Examples

How Tab Complete Will Work:
`T` [TAB] → `Templates`
`P` [TAB] → `Publi`
`Do` [TAB] → [beep]
`Do` [TAB] [TAB] → `Documents Downloads Do-re-me.txt`
`Doc` [TAB] → `Documents`

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

(so it won't work for output files!)

Command line completion- Question



Which Is the Shortest Way to Specify Mi-so-la.txt?

A M[TAB]

B Mi [TAB]

C Mi-so-la [TAB]

Specifying **Multiple** File Paths – **Wildcards**

Sometimes we want to refer to more than one file / location

 2024_report.txt  2023_report.txt  2019_report.txt

Common part of name
Unique part of name

Use Wild cards to substitute for unique parts of related file paths

- Shell will expand them before passing them on to the program

Wildcard	Meaning	Example
?	**One** of **Any** character	**202?_report.txt**
*	**Any** number of **Any** characters	**20*_report.txt**

Could be more ambiguous here e.g. **20*** , ***.txt** or even *

But it depends what else this path would capture!

Using Wildcards

How do we use them:	
At any point in the path	✅
Multiple wildcards can be in the same path	✅
Do make sure expression captures files of interest specifically!	✅
Command line completion won't work after the first wildcard	❌

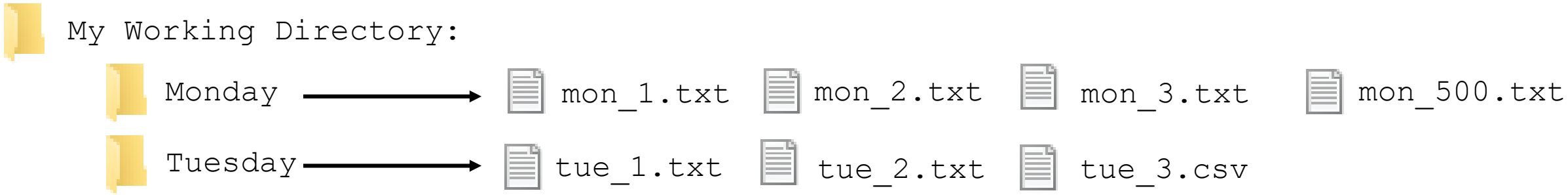
```
ls  -ltd --reverse D*
```

Program name

Switches

Data (normally files)

Using Wildcards - Questions



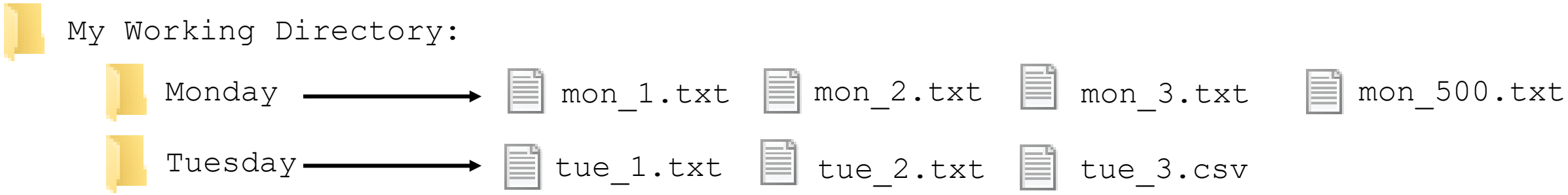
How can I list only text files from Tuesday?

A `ls Tuesday/*`

B `ls Tuesday/?.txt`

C `ls Tuesday/*txt`

Using Wildcards - Questions



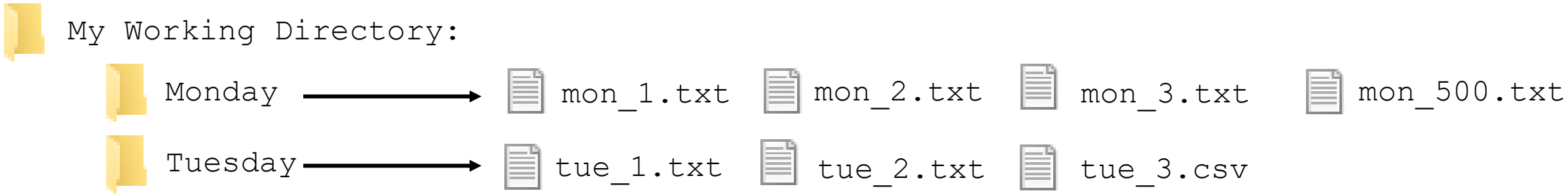
What files will "`ls Monday/mon_?.txt`" return?

A `mon_1.txt` `mon_2.txt` `mon_3.txt`

B `mon_1.txt` `mon_2.txt` `mon_3.txt` `mon_500.txt`

C `tue_1.txt` `tue_2.txt` `tue_3.csv`

Using Wildcards - Questions



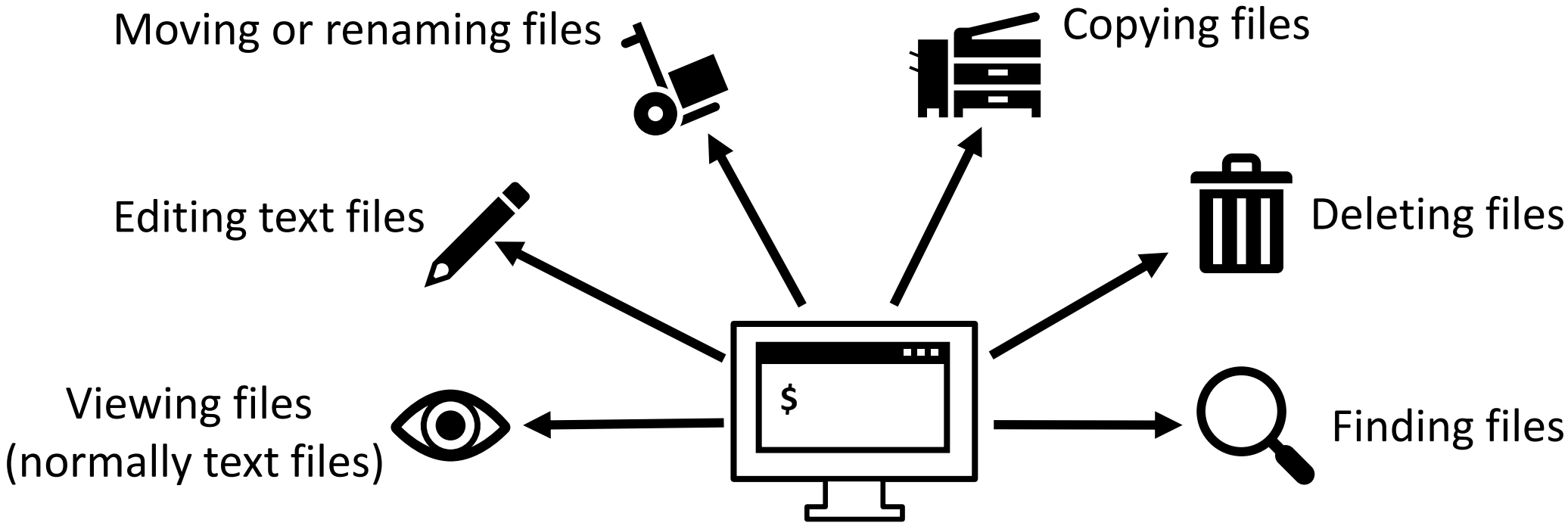
How can I list all the text files in both Monday and Tuesday?

A `ls ?day/*`

B `ls *`

C `ls */*txt`

Manipulating files



You will spend a lot of time managing files on a Linux system

Viewing Files

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

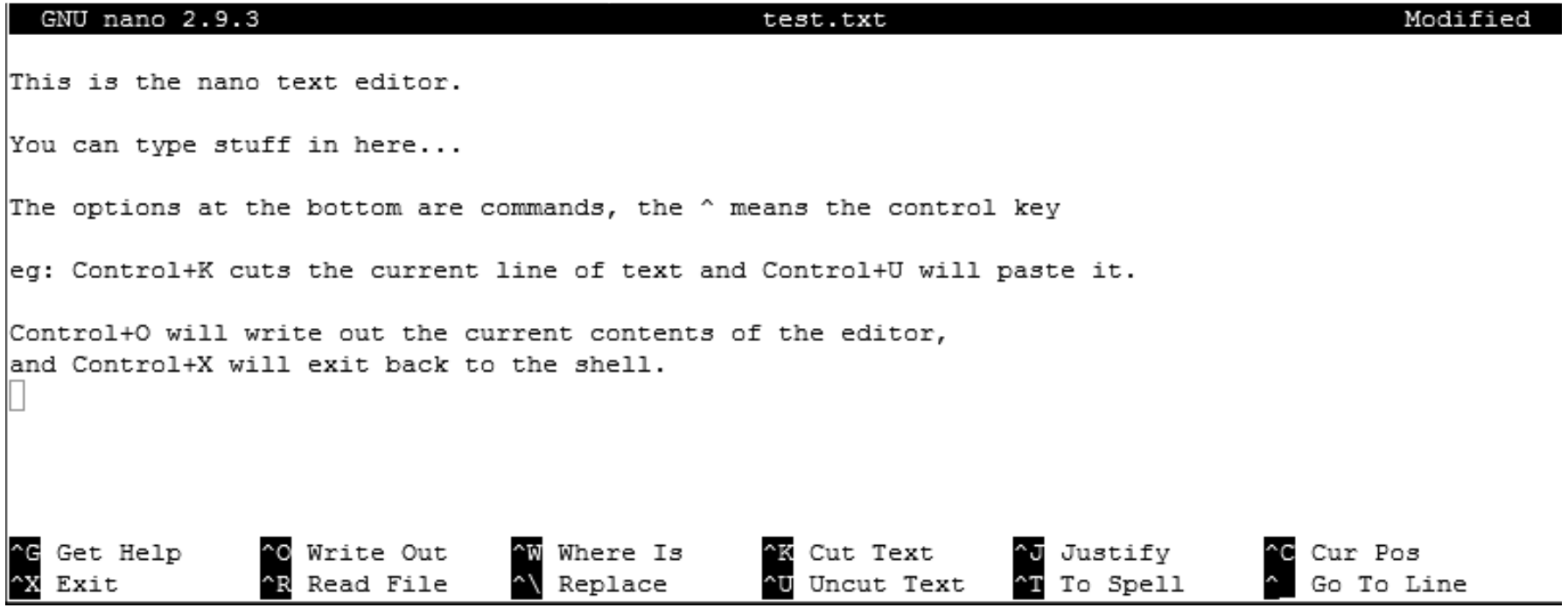
Quick look	
`head -[number] [file]`	Look at the first X lines of the file
`tail -[number] [file]`	Look at the last X lines of the file

More scalable solution	
`less [file]`	A 'pager' program, sends output to the screen one page at a time
`-s`	A useful switch that stops line wrapping
Navigation inside less:	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

Editing files

- Lots of text editors exist, both graphical and command line
- Many have special functionality for specific content (C, HTML etc)
- `nano` is a simple command line editor which is always present

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



Moving / Renaming files

- Use `mv` command for both (renaming = moving from one name to another)

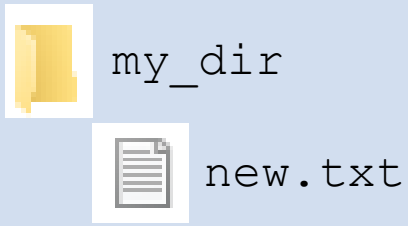
```
mv [existing file or directory] [new name/location]
```

- Good to Know….
  - If “location” is a existing directory, the file is moved there with its existing name
  - Moving a directory moves all of its contents as well
  - Shortcuts can help to form the path of where you want to move files to/from

The return of useful shortcuts!		
.	The current directory	Useful for “pull” moves
..	The directory immediately above the current directory	Useful for “push” moves

Moving / Renaming files – “Push”

You are here

Start	Command	Outcome
	**mv old.txt new.txt**	
	mv old.txt ../Saved/	
	mv old.txt ../Saved/new.txt	

Moving / Renaming files – “Pull”

Start	Command	Outcome
	**mv ../my_dir/old.txt .**	

You are here

Copying a file

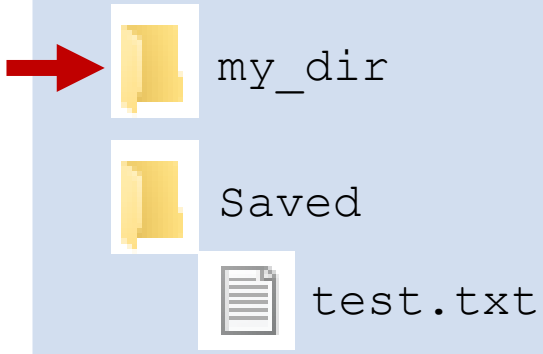
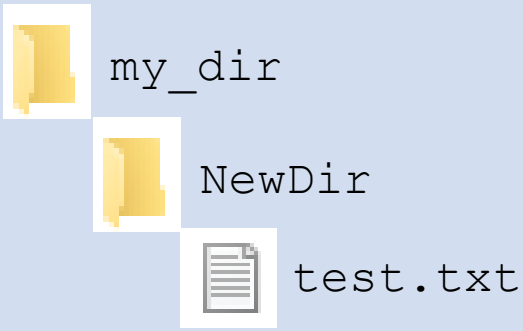
- Use `cp` command on a single file

```
cp [existing file] [new name/location]
```

Start	Command	Outcome
 my_dir  old.txt	**cp old.txt new.txt**	my_dir  old.txt  new.txt

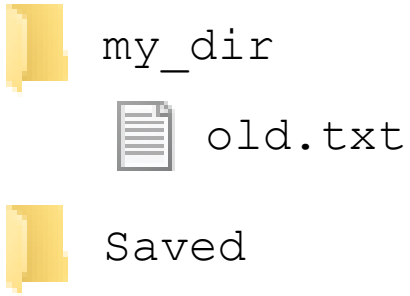
Copying Directories with recursive copy

`cp -r [existing directory] [new name/location]`

Start	Command	Outcome
	**`cp -r ../Saved NewDir`**	
	**`cp -r ../Saved ExistingDir/`** **(only if ExistingDir exists)**	

*remember the original “Saved” directory will also still exist

Copying files: Match the Command with the Desired Action

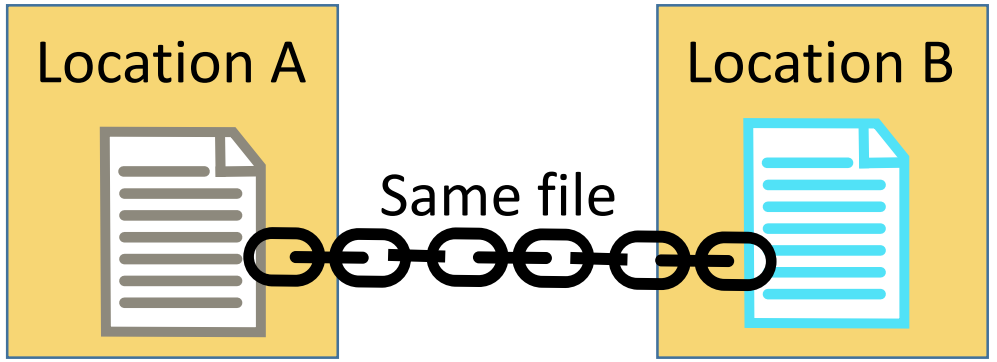


Working Directory	Action
my_dir	**1) Copy old.txt to Saved**
my_dir	**2) Copy old.txt to Saved and call it new.txt**
Saved	**3) Copy old.txt to Saved**

Command
A) `cp old.txt ../Saved/new.txt`
B) `cp ../Saved/old.txt .`
C) `cp old.txt ../Saved/`

Linking rather than copying

- Copy duplicates the data in a file
  - Can be a problem with big data files
- Links are a way to do 'virtual' copies



- Two types of link, hard links and soft (or symbolic) links
  - We will always use soft links as they're more flexible

```
ln -s [from] [to]
```

Start	Command	Outcome
mydir test.txt	`ln -s test.txt test2.txt`	mydir test.txt test2.txt

Working with symbolic links

When you list a link you can see where it points…

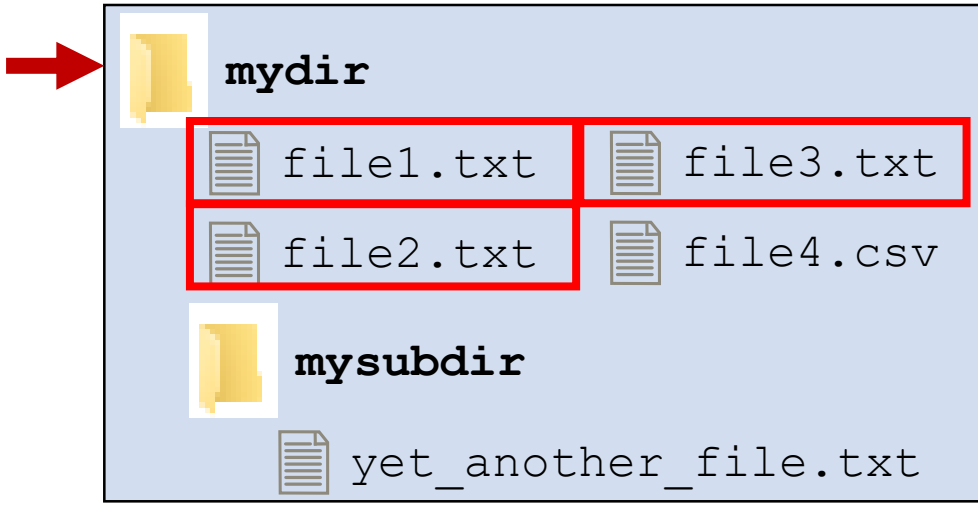
```
$ ls -l test2.txt
lrwxrwxrwx 1 babraham babraham 8 Sep 11 16:27 test2.txt -> test.txt
```

…but you can use it like a file

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

Finding Things with `find`

find	[starting point]	[global options]	[other arguments]
	Location to start from	**modify** the behaviour of find	**Tests** of what to look for
		• How to handle symbolic links **-H, -L, -P** • How to handle how deep to search in the filesystem **-maxdepth -mindepth**	• Find a given file name **-name [filename]** • Find matches belonging to a user **-user [username]** • Find matches of a certain type **- d -f**



Example:

```
find .-maxdepth 1 -name '*.txt'
```

Deleting files

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

Use the `rm` command to delete files and directories (and all of their contents)

```
rm [name(s) of file to delete]
```

```
rm -r [name(s) of directory to delete]
```

Examples

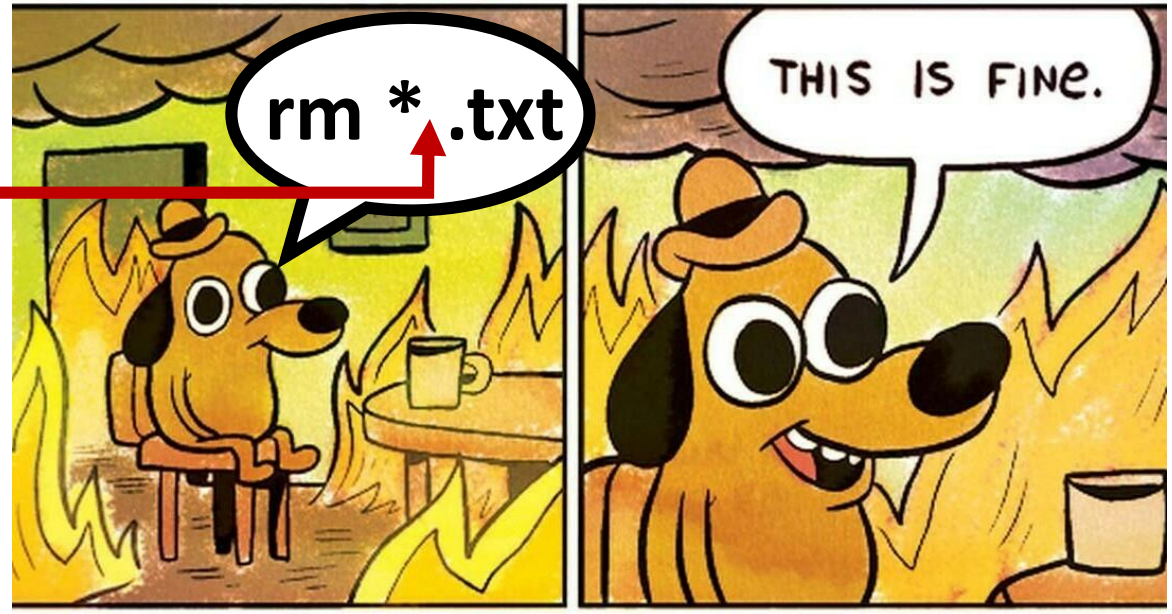
- `rm test_file.txt test_file2.txt`
- `rm -r Old_directory/`

Deleting files – With Wildcards

You can use the wildcard shortcuts to delete multiple files or directories

```
rm *.txt
```

be **VERY** careful using wildcards



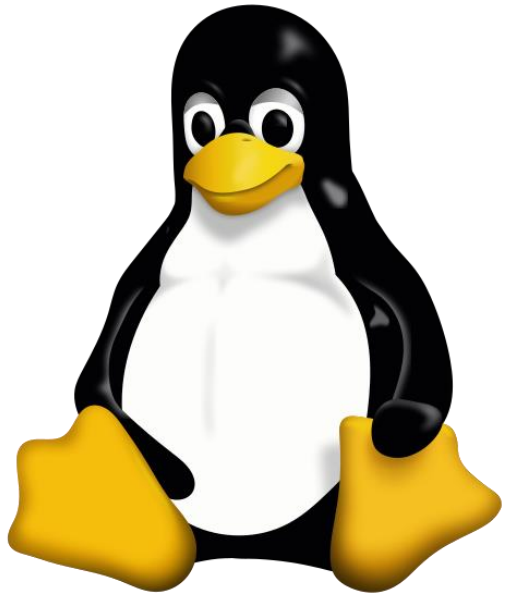
Accidental space

Always run `ls` first to see what will go

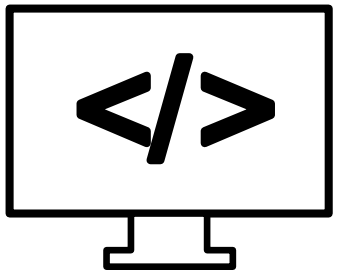
Exercise 3

More advanced BASH usage

Babraham Bioinformatics



What we know already

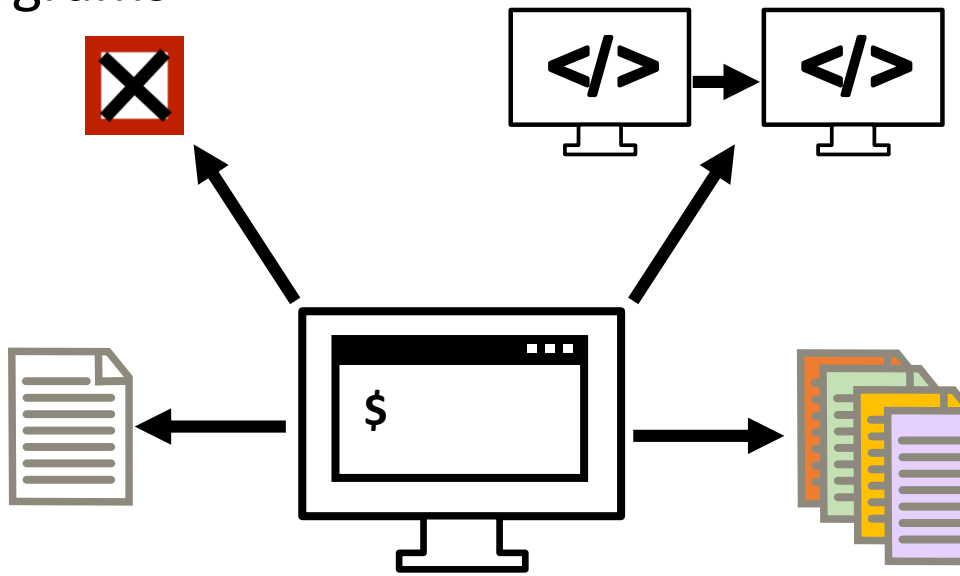


- How to run programs
  - How to modify the options for a program using switches
  - How to supply data to programs using file paths and wildcards

How Can we Usefully Build on this?

What else can we do…

Check for errors in programs which are running



Link programs together into small pipelines

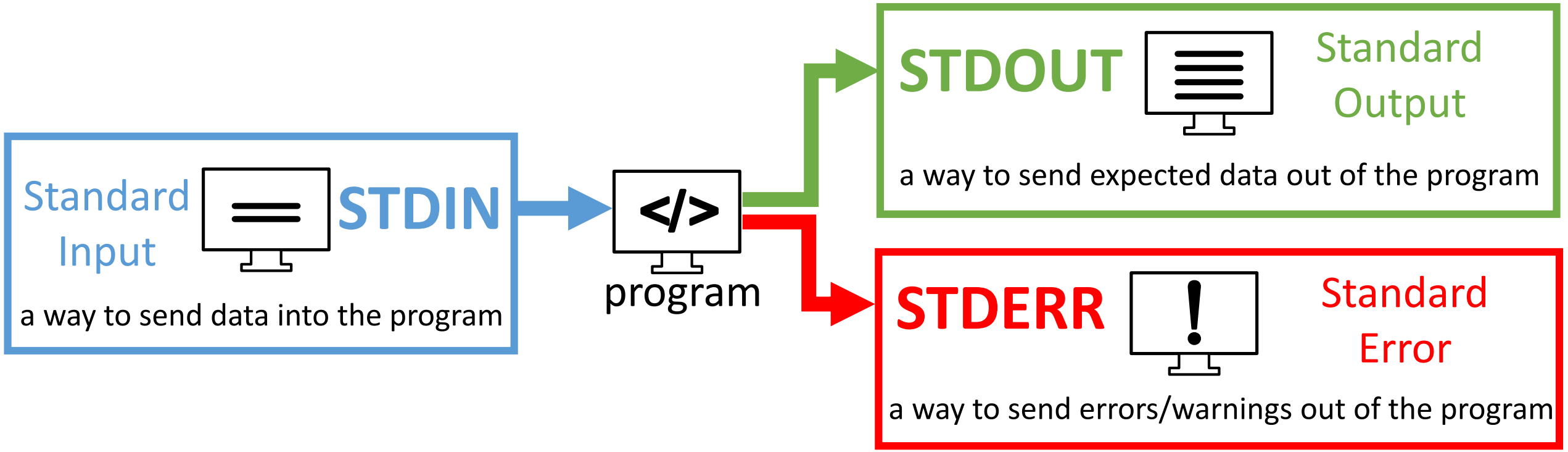
Record the output of programs

Automate the running of programs over batches of files

All possible with a bit more knowledge of the BASH Shell

Communicating with Programs

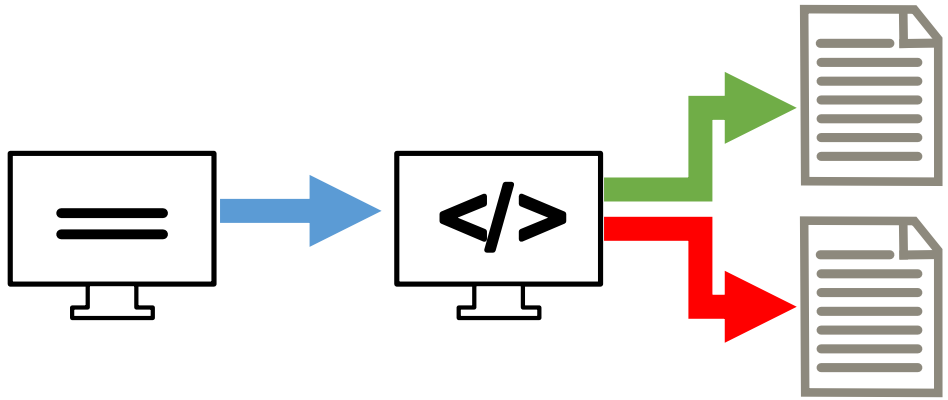
Three data streams exist for all Linux programs:



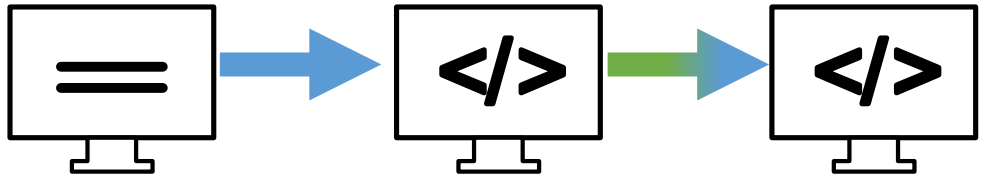
By default STDOUT and STDERR are connected to your shell

so when you see text coming from a program it's from these streams

Communicating with Programs

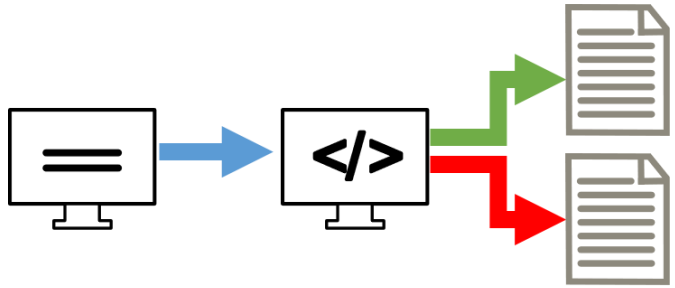


Redirecting standard streams to files



Redirecting standard streams to other programs

Redirecting standard streams



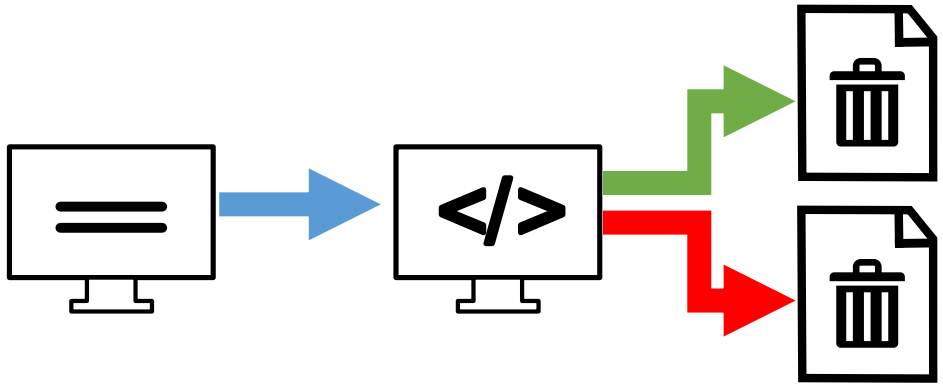
You can redirect using arrows at the end of your command

`>  [file]`	Redirects STDOUT
`2> [file]`	Redirects STDERR
`> [file] 2>&1`	Sends STDERR into STDOUT
`< [file]`	Redirects STDIN

```
$ find . -print > file_list.txt 2> errors.txt

$ ls
Data  Desktop  Documents  Downloads  errors.txt  examples.desktop  file_list.txt
Music  Pictures  Public  Templates  Videos

$ head file_list.txt
.
./Downloads
./Pictures
./Public
./Music
```

Throwing stuff away

- Sometimes you want to be able to hide output
  - STDOUT - I just want to test whether something worked
  - STDERR - I want to hide progress / error messages



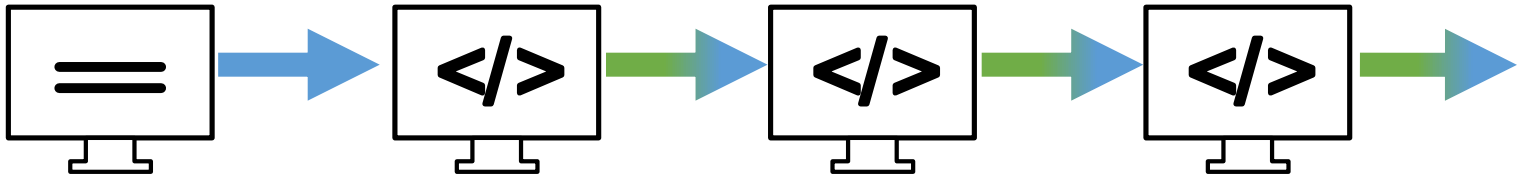
Linux defines a special file `/dev/null`
Which just discards all data sent to it

Throw away the STDOUT	`program > /dev/null`
Throw away the STDERR	`program 2> /dev/null`

Linking programs together with pipes

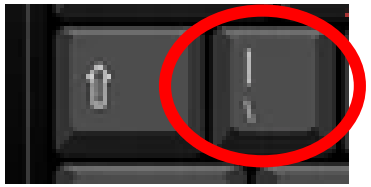
UNIX was designed to have lots of small programs doing specific jobs

Which could be linked together to perform more advanced tasks



Do this by connecting STDOUT from one program to STDIN on another

This is done with Pipes |



```
$ ls | head -2
Data
Desktop
```

Useful programs for pipes

- You can theoretically use pipes to link any programs
- But there are some which are particularly useful, like:

`wc`	to do word and line counting
`grep`	to do pattern searching
`sort`	to sort things
`Uniq`	to deduplicate things
`less`	to read large amounts of output
`zcat/gunzip/gzip`	to do decompression or compression

Small example pipeline

Take a compressed fastq sequence file, extract from it all of the entries containing the telomere repeat sequence (TTAGGG) and count them

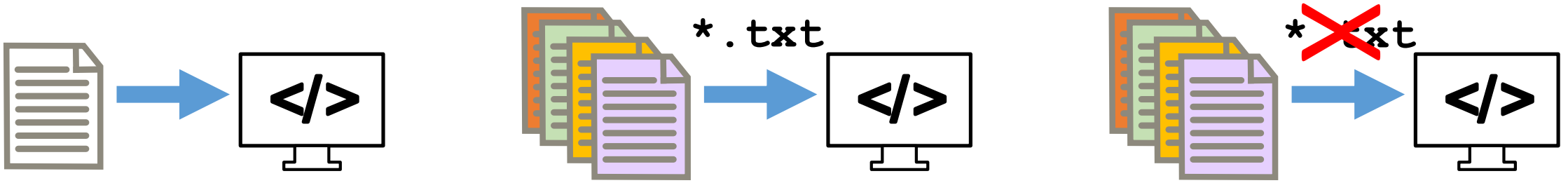
Decompress the fastq file → **Find the pattern** → **Count the matches**

```
zcat file.fq.gz | grep TTAGGGTTAGGG | wc -l
```

```
$ zcat file.fq.gz | wc -l
179536960

$ zcat file.fq.gz | grep TTAGGGTTAGGG | wc -l
3925
```

Iterating over files

When processing data often need to re-run the same command multiple times for different input/output files.



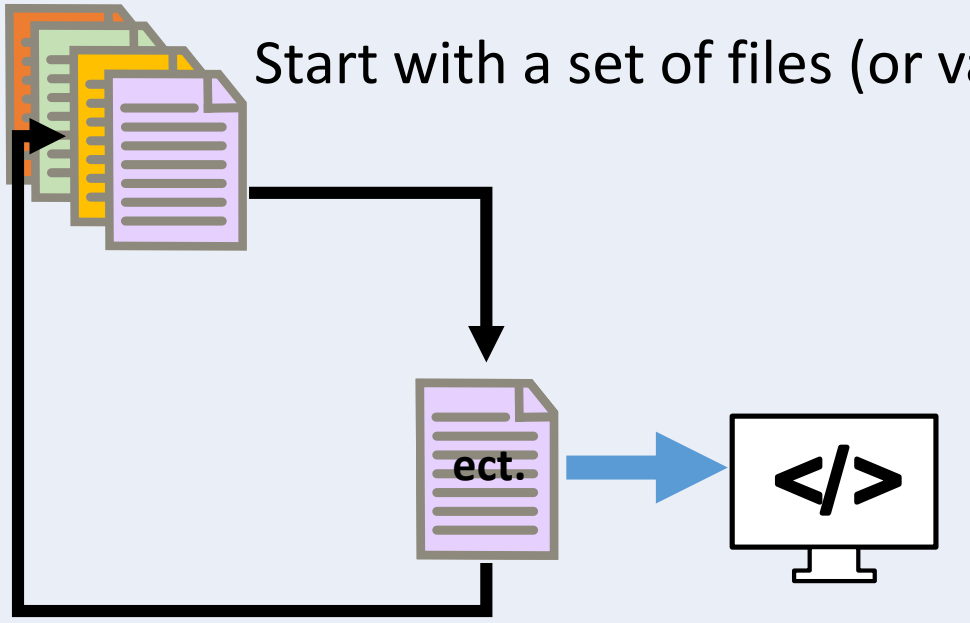
Some programs support being provided with multiple input files i.e. wildcards!

BUT MANY DON'T!

Instead we use the automation features of the BASH shell to automate running these programs

The BASH `for` loop

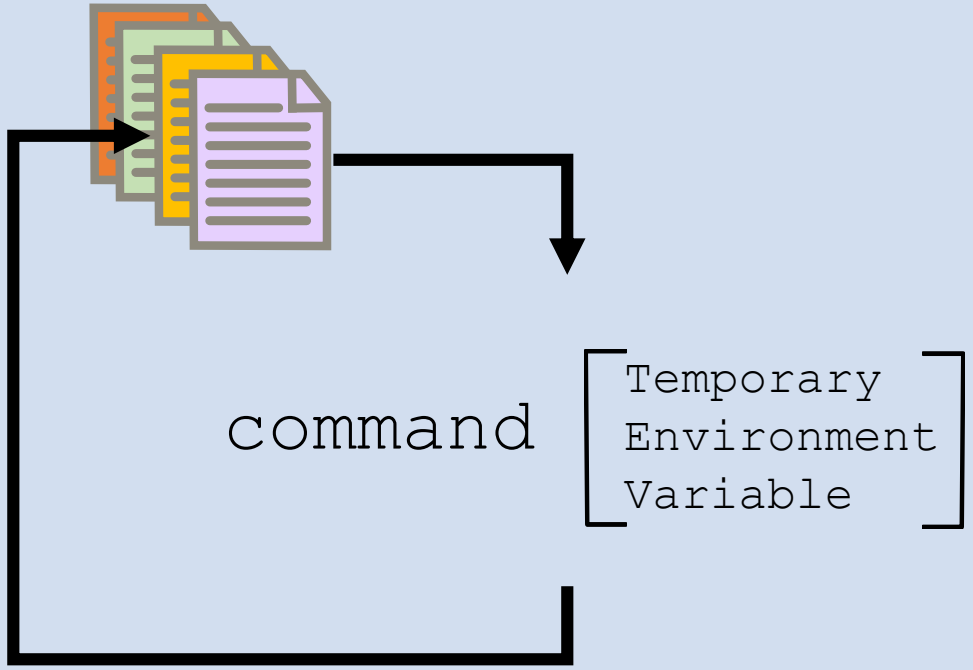
What?



Loop over these to do some function
for each in turn

How?

Use Simple looping construct



Write commands using a special variable

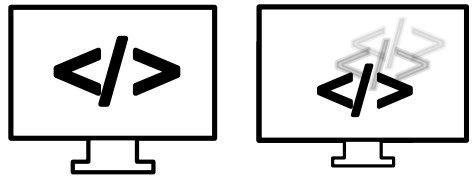
Takes on the value of each item in turn

Example of BASH **for** loops

```
for file in *txt
  do
    echo $file
    grep .sam $file | wc -l
  done
```

Job Control

- By default you run one job at a time in a shell
- Shells support multiple running jobs



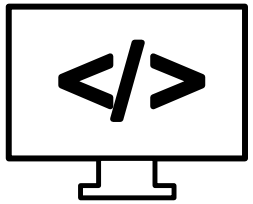
`jobs`	lists the jobs in this shell

States of job:	
Running - foreground	shell has the attention of the job
Running - background	output goes to the shell but other jobs can run
Suspended	job exists but is paused, consumes no CPU
Running - disconnected	output is no longer attached to the shell

Job Control

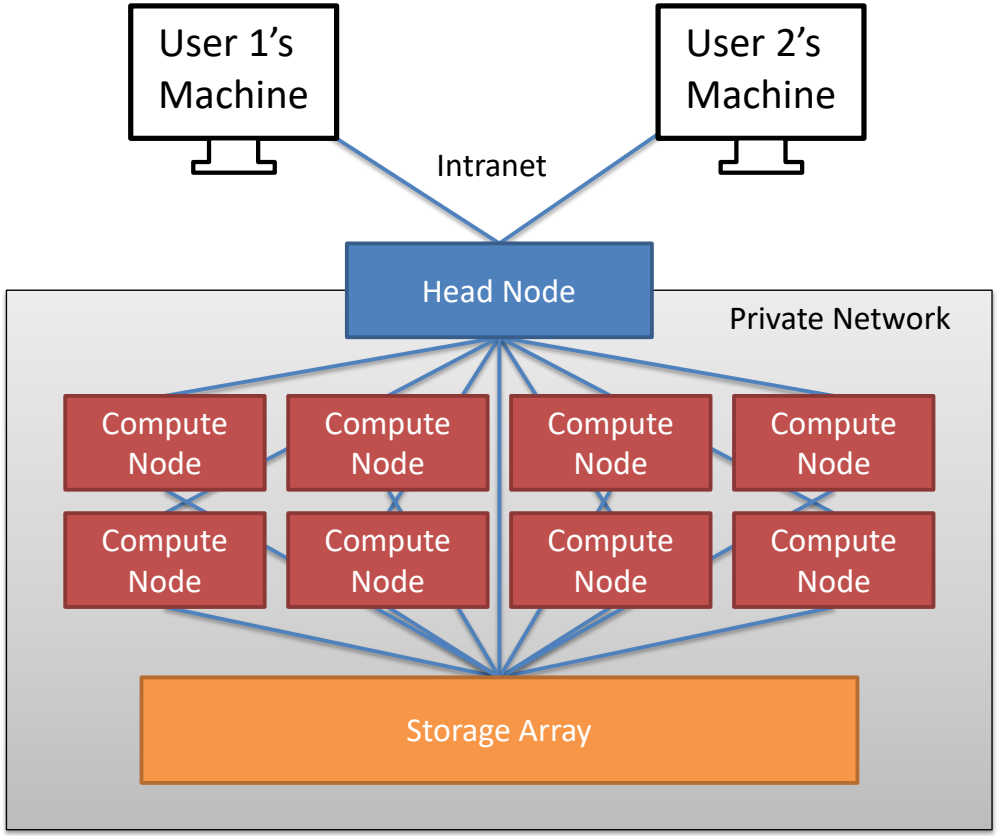
Working Directory	
`prog_to_run`	starts in foreground
`prog_to_run &`	starts in background
`Control + Z`	suspends the current job
`bg`	Send a job to the background
`fg`	Bring a job to the foreground
`nohup prog_to_run` `nohup prog_to_run > log.txt &`	disconnects, logs to `nohup.out` … or redirect to your choice of file **Means it can't be killed when terminal exits**

More Extended Job Control on Clusters

Control on a single machine 

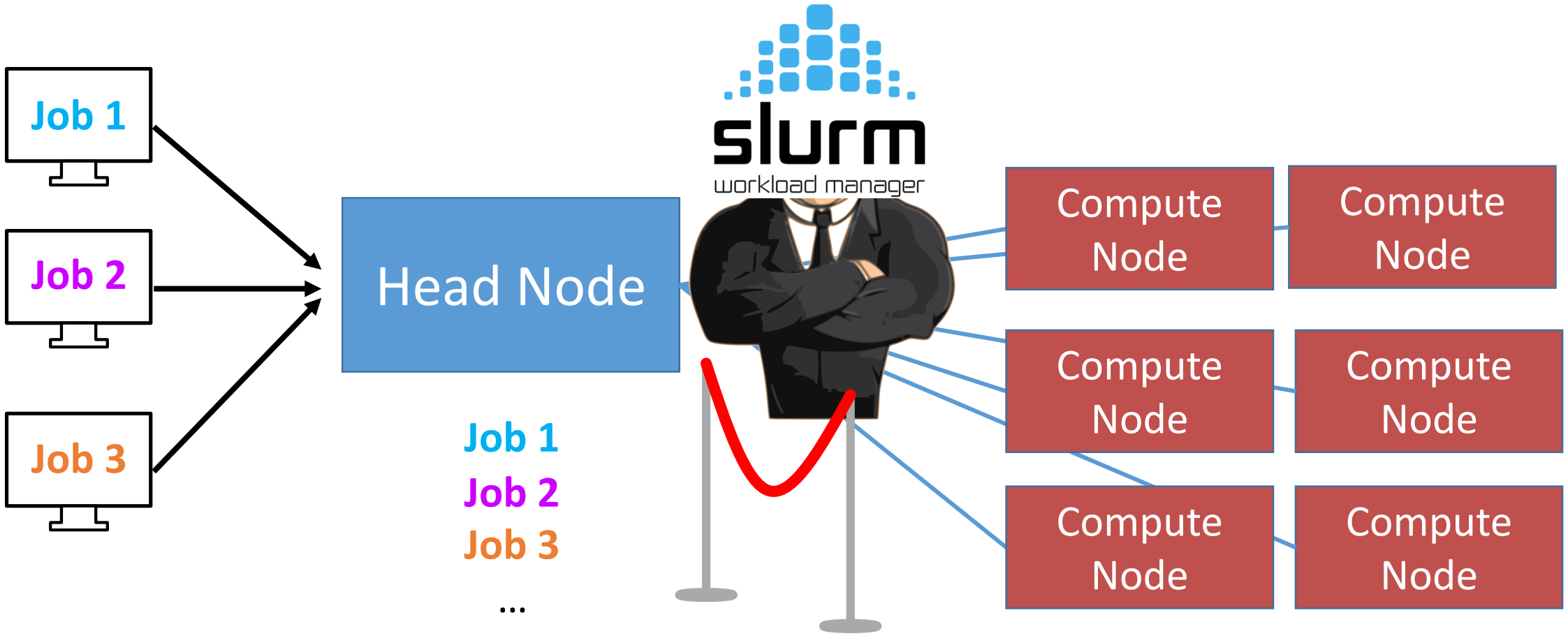
vs

Control on a Cluster



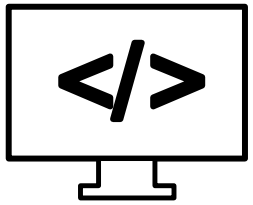
- Same for small jobs we can run on the Head Node
  - e.g. nohup, fg, bg
- Need a bit more control for bigger jobs
  - Workload managers
  - Workflow managers

Workload managers – Cluster Queues



Workload managers – Cluster Queues

Submitting a job directly



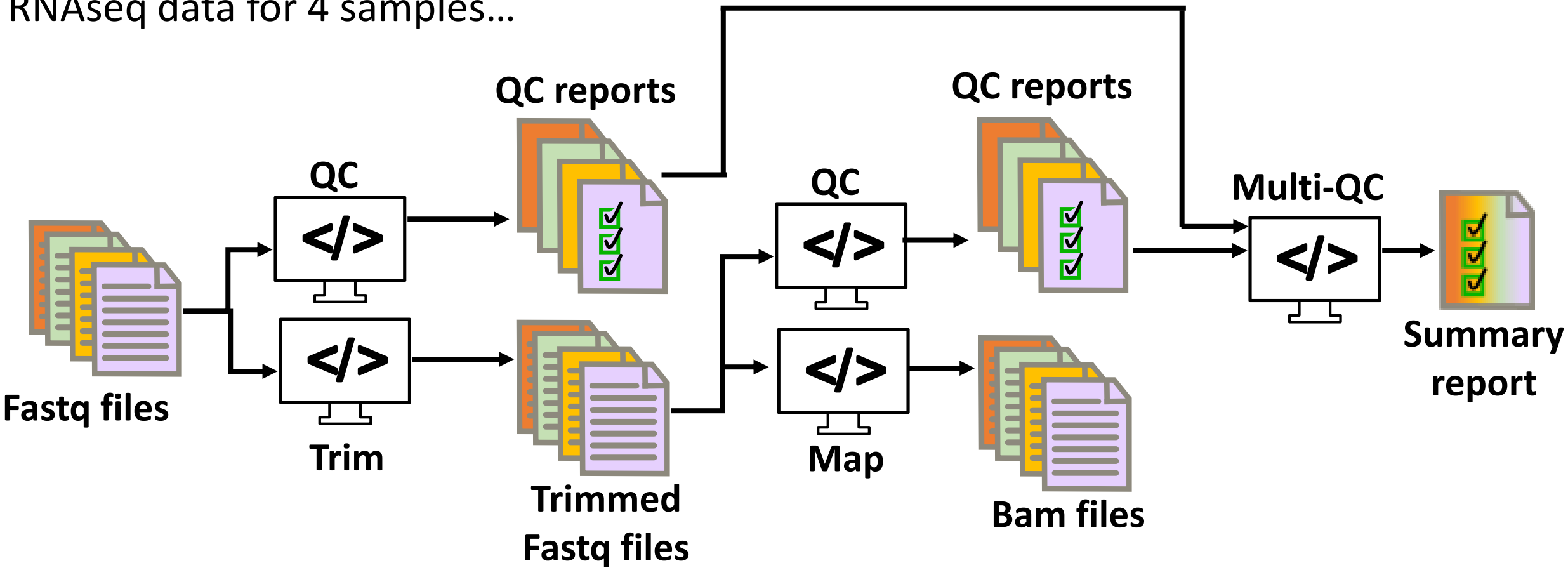
```
fastqc data.fq.gz
```

Submitting a job to a workload manager


```
ssub
    -o f.log
    --cores=2
    --mem=5G
    fastqc data.fq.gz
```

Workflow Mangers – Beyond 1 job…

Imagine you have generated
RNAseq data for 4 samples…



…A lot to coordinate!

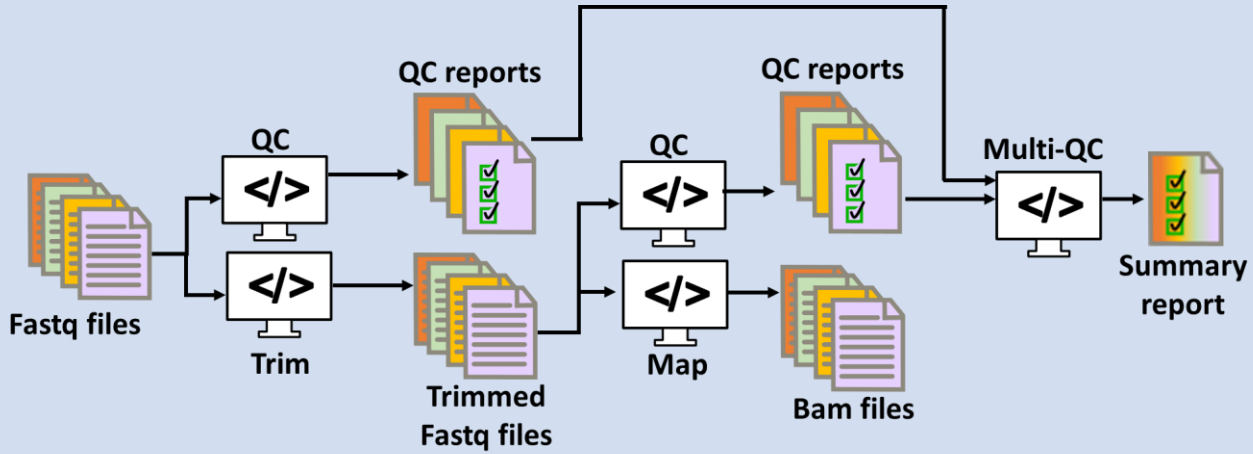
Workflow Mangers

- Larger Scale Automation
- Multiple Programs
- Multiple Files
- Integrates with Clusters

nextflow

snakemake
A framework for reproducible data analysis

Turn this…



…Into this!

```
nf_rnaseq --genome GRCh38 *fastq.gz
```

```
executor >  slurm (21)
[15/929bd5] process > FASTQC (lane8_DD_P9_TGACCA_L008)
[b9/674ced] process > FASTQ_SCREEN (lane8_FF_P4_ATCACG_L008)
[ca/b39d14] process > TRIM_GALORE (lane8_FF_P9_CGATGT_L008)
[c0/4dcaf9] process > FASTQC2 (lane8_FF_P9_CGATGT_L008)
[58/879cf5] process > HISAT2 (lane8_FF_P9_CGATGT_L008)
[c4/cfe1f1] process > MULTIQC
Completed at: 05-Feb-2021 08:47:47
Duration    : 4m 2s
CPU hours   : 1.9
Succeeded   : 21
```

Workflow completion notification

Run Name: jovial_bartik

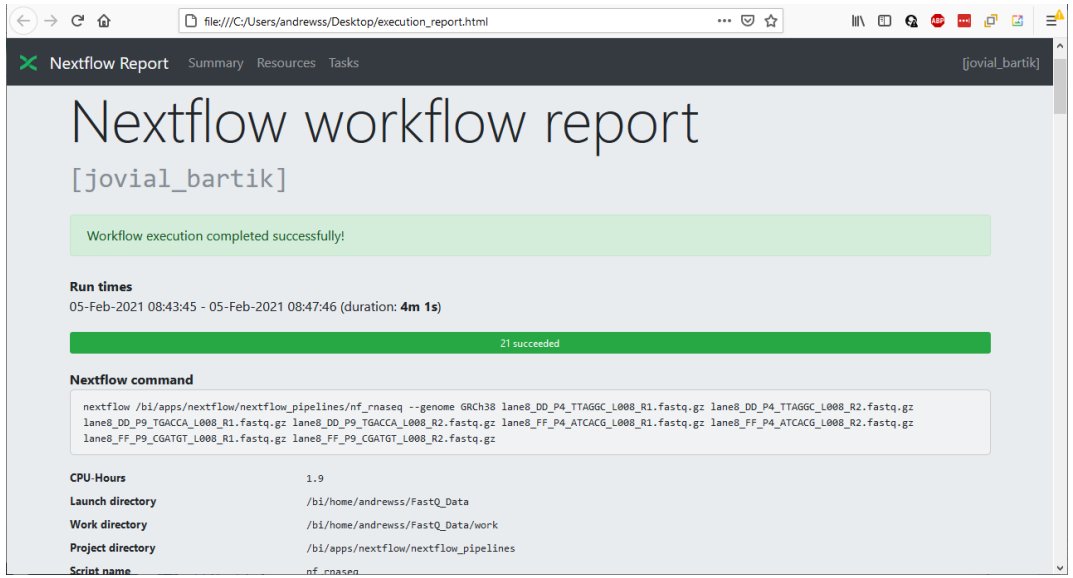
Execution completed successfully!

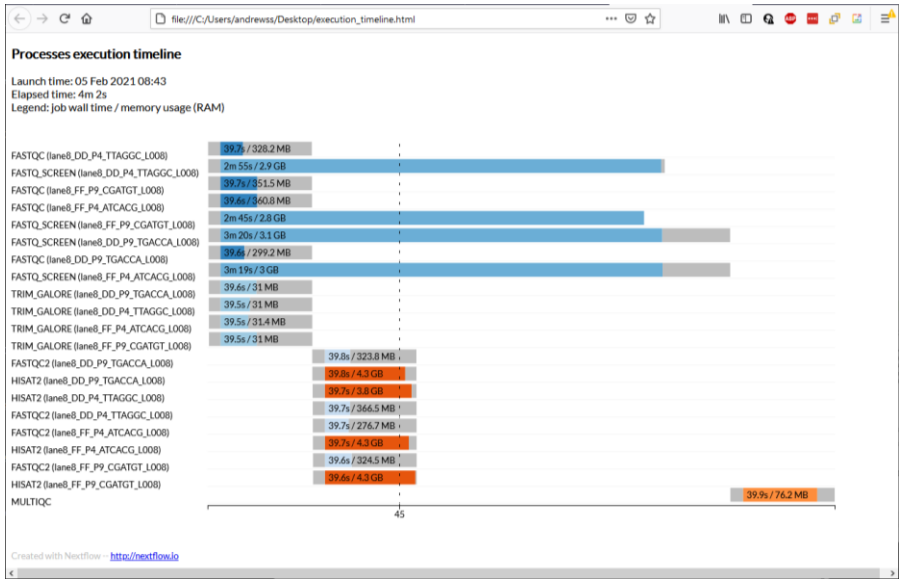
The command used to launch the workflow was as follows:

```
nextflow /bi/apps/nextflow/nextflow_pipelines/nf_rnaseq --genome GRCh38
lane8_DD_P4_TTAGGC_L008_R1.fastq.gz lane8_DD_P4_TTAGGC_L008_R2.fastq.gz
lane8_DD_P9_TGACCA_L008_R1.fastq.gz lane8_DD_P9_TGACCA_L008_R2.fastq.gz
lane8_FF_P4_ATCACG_L008_R1.fastq.gz lane8_FF_P4_ATCACG_L008_R2.fastq.gz
lane8_FF_P9_CGATGT_L008_R1.fastq.gz lane8_FF_P9_CGATGT_L008_R2.fastq.gz
```

Execution summary

Launch time	05-Feb-2021 08:43:45
Ending time	05-Feb-2021 08:47:46 (duration: 4m 1s)
Total CPU-Hours	1.9
Tasks stats	Succeeded: 21 Cached: 0 Ignored: 0 Failed: 0





Exercise 4