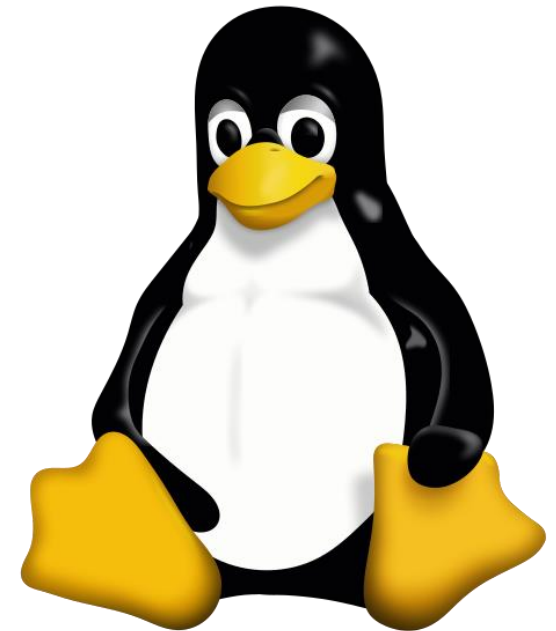
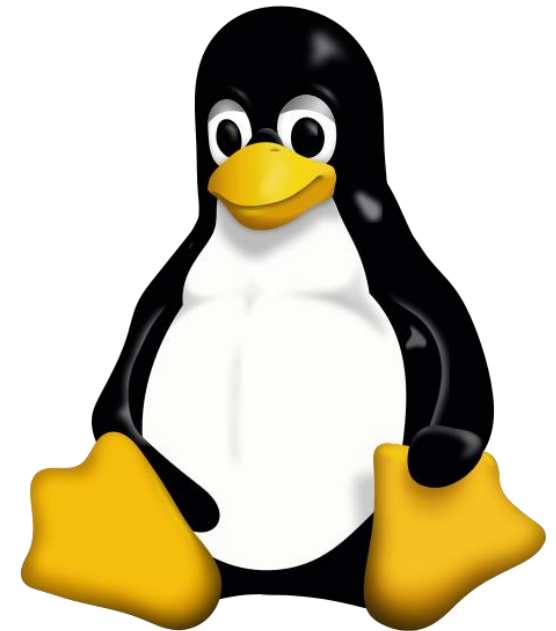


An Introduction to Unix

v2023-01

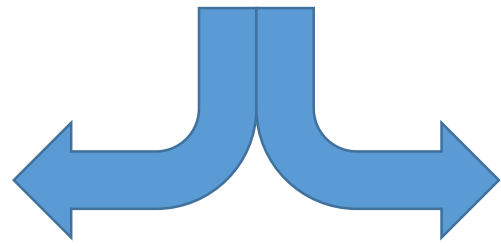


Terminology and Distributions



UNIX®

00011110 00011110 00011110 00011110 00011110 00011110 00011110 00011110



 **debian**
 **ubuntu**

- Admin tools
- Bundled Software
- Support duration / cost



Most Popular Distributions

- Ubuntu
 - Based on Debian
 - Originally aimed at desktop users
 - Friendly and well supported
 - Regular updates (some with long term support)
- CentOS / Alma / Rocky
 - Free versions of the RedHat commercial distro
 - Mostly aimed at servers
 - Stable software and long term support
 - Used in many large environments

ubuntu 

 **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)

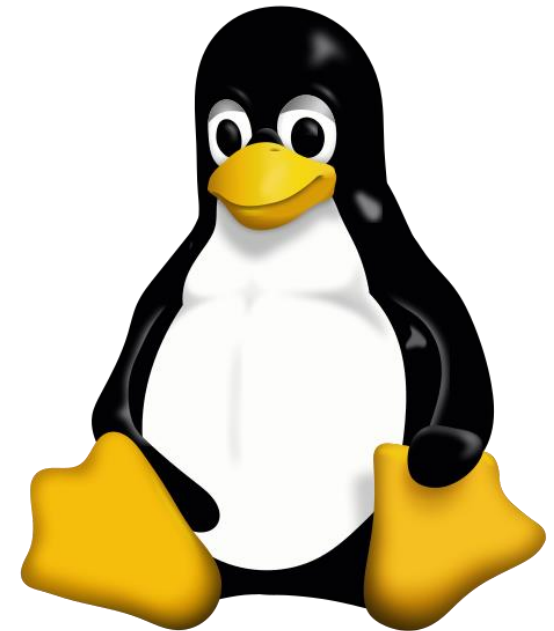
- **Virtual Machine**

- 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 and OS images available

Connecting to Linux Installations

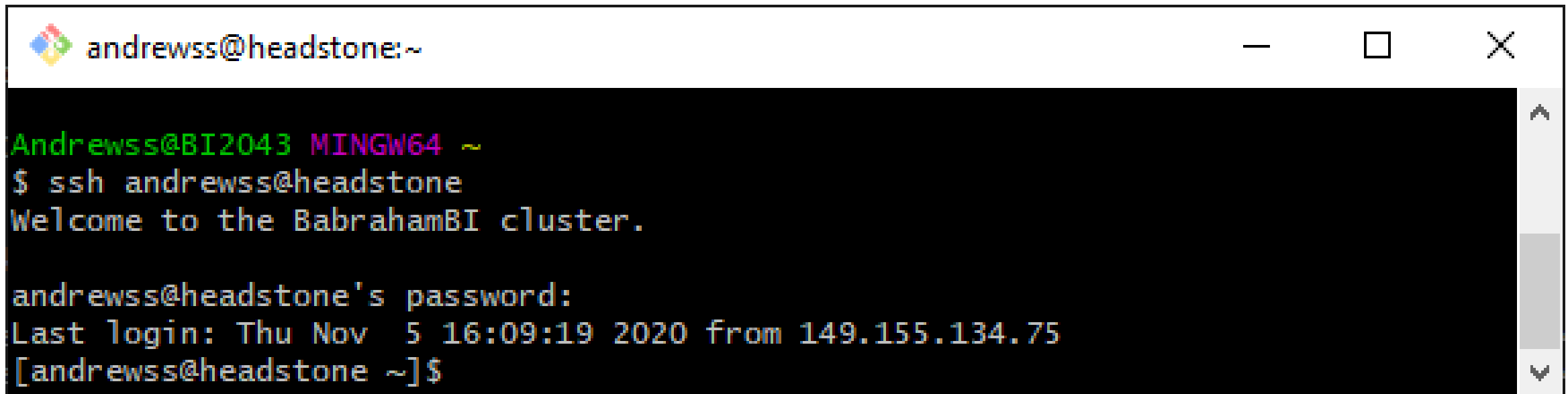


Connecting to a remote Linux installation

- SSH – "Secure shell"
 - Encrypted connection
 - Text based interface
- OSX or Linux
 - Use the terminal program which comes with the OS
- Windows
 - Git Bash (<https://gitforwindows.org/>)
 - PuTTY (<https://www.putty.org/>)
- Authentication
 - Username + Password + 2 Factor Auth
 - Username + SSH key + 2 Factor Auth

SSH + Password connection

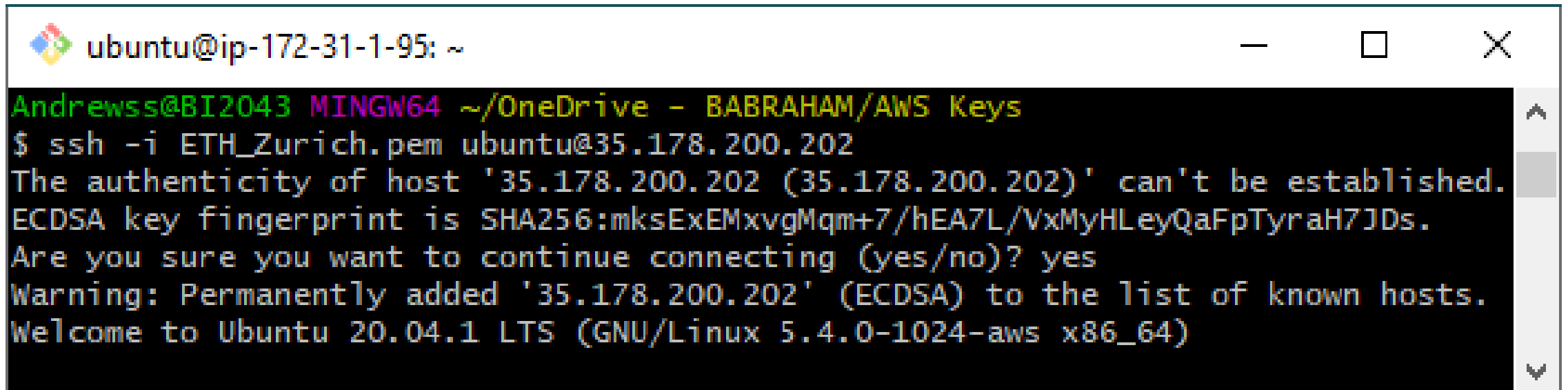
- `ssh username@server.address`
- [Will be promoted for password]



```
andrewss@headstone:~  
Andrewss@BI2043 MINGW64 ~  
$ ssh andrewss@headstone  
Welcome to the BabrahamBI cluster.  
  
andrewss@headstone's password:  
Last login: Thu Nov  5 16:09:19 2020 from 149.155.134.75  
[andrewss@headstone ~]$
```


SSH + Key connection

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

A terminal window with a title bar that reads "ubuntu@ip-172-31-1-95: ~". The terminal content shows a user named "Andrewss@BI2043" in a "MINGW64" environment at "~\OneDrive - BABRAHAM/AWS Keys". They execute the command "\$ ssh -i ETH_Zurich.pem ubuntu@35.178.200.202". The terminal displays a warning about the host's authenticity, showing the ECDSA key fingerprint: "SHA256:mksExEMxvgMqm+7/hEA7L/VxMyHLeyQaFpTyr aH7JDs". The user responds "yes" to the question "Are you sure you want to continue connecting (yes/no)?". The terminal then shows a warning: "Warning: Permanently added '35.178.200.202' (ECDSA) to the list of known hosts." and finally displays the Ubuntu welcome message: "Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-1024-aws x86_64)".

```
ubuntu@ip-172-31-1-95: ~
Andrewss@BI2043 MINGW64 ~\OneDrive - BABRAHAM/AWS Keys
$ ssh -i ETH_Zurich.pem ubuntu@35.178.200.202
The authenticity of host '35.178.200.202 (35.178.200.202)' can't be established.
ECDSA key fingerprint is SHA256:mksExEMxvgMqm+7/hEA7L/VxMyHLeyQaFpTyr aH7JDs.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '35.178.200.202' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-1024-aws x86_64)
```

Graphical Connections

- Single application windows

- X11
- Sits on top of SSH

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



<https://www.xquartz.org/>

<https://sourceforge.net/projects/vcxsrv/>

- Virtual Desktop

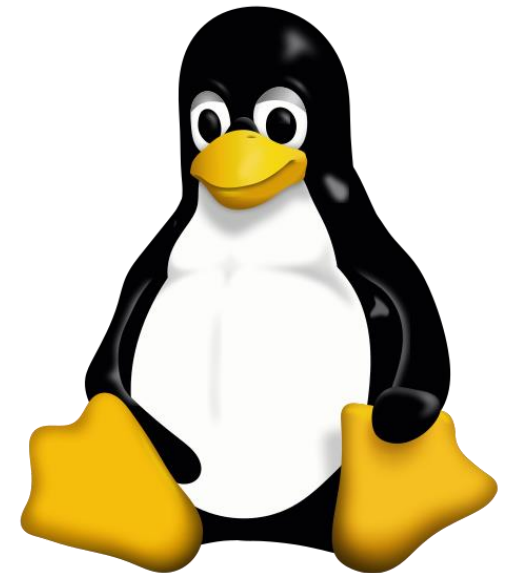
- VNC
- Stand alone application or
- Browser based desktop



REALVNC
Apache Guacamole™

Exercise 1

Running programs in the BASH shell



Launching 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 here"
 - ..or click the black square in the toolbar
 - ..or, from Applications menu find Terminal Emulator


```
student@ip-172-31-1-95: ~
File Edit View Search Terminal Help
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
student@ip-172-31-1-95:~$ █
```

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

```
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

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.

Help Files

- Non-core programs (including most analysis tools) won't have a manual page
- They will often have their own internal help which looks very similar
- You can usually access this using the `--help` (or `-h`) switch

Help Files (fastqc --help)

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

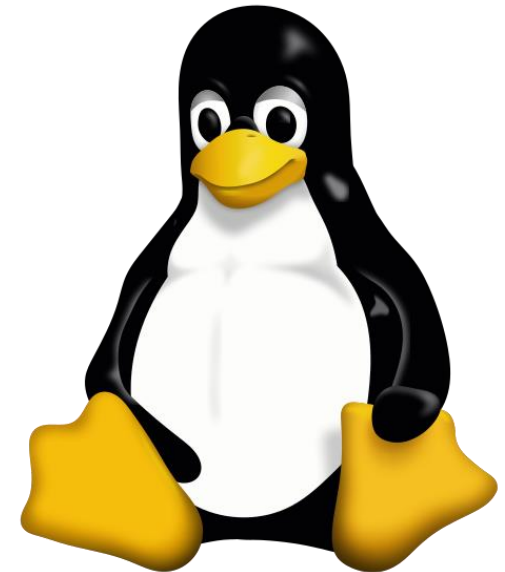
FastQC reads a set of sequence files and produces from each one a quality control report consisting of a number of different modules, each one of which will help to identify a different potential type of problem in your data.

The options for the program are 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



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)

Creating and moving into directories

- Every Unix session has a '**working directory**' which is a folder where the shell looks for file paths
- You can see your current working directory with `pwd`
- Your initial working directory will be your home directory (eg `/home/user`)
- You can change your working directory with `cd [new working directory]`
- Running `cd` on its own takes you back home
- You can create a new directory with `mkdir [new directory name]`

Changing the working directory

```
[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

- 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

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

- **Absolute paths from the top of the file system**

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

- **Relative paths from your current director** (`home/simon/Course/`)

```
Data/big_data.csv
```

- **Paths using the home shortcut**

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

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

List of files / folders:

Desktop
Documents
Downloads
Music
Public
Published
Templates
Videos

T [TAB] → Templates

P [TAB] → Publ

Do [TAB] → [beep]

Do [TAB] [TAB] → Documents Downloads

Doc [TAB] → 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

- A quick way to be able to specify multiple related file paths
- 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 your 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
```

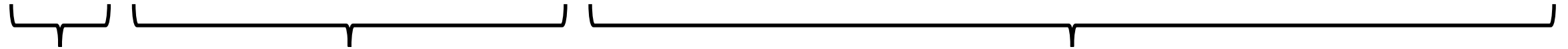
```
Friday/fri_1.txt  Monday/mon_1.txt  Monday/mon_3.txt  Tuesday/tue_1.txt  
Friday/fri_2.txt  Monday/mon_2.txt  Monday/mon_500.txt  Tuesday/tue_2.txt
```

```
$ 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.

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 [file]` 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 | <code>less -S</code> (no wrapping) |
| • 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

Using nano to edit text files

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

```
GNU nano 2.9.3          test.txt          Modified
This is the nano text editor.
You can type stuff in here...
The options at the bottom are commands, the ^ means the control key
eg: Control+K cuts the current line of text and Control+U will paste it.
Control+O will write out the current contents of the editor,
and Control+X will exit back to the shell.
□

^G Get Help      ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit         ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^  Go To Line
```

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
- 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 links and soft (or symbolic) links
- We will always use soft links as they're more flexible.

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

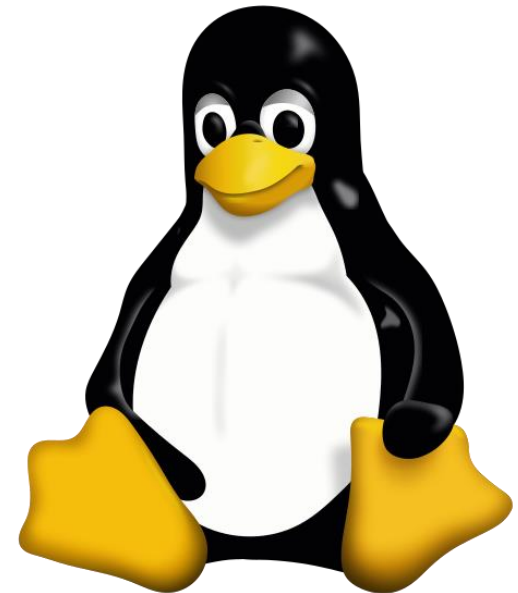
- Linux has no undo.
- Deleting files has no recycle bin.
- Linux will not ask you "are you sure"

- 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/`

Exercise 3

More advanced BASH usage



What we know already

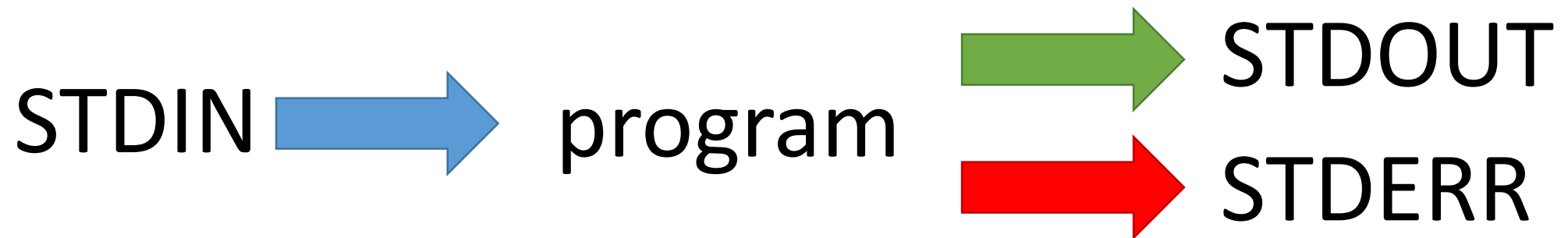
- How to run programs
- How to modify the options for a program using switches
- How to supply data to programs using file paths and wildcards

What else can we do

- Record the output of programs
 - Check for errors in programs which are running
 - Link programs together into small pipelines
 - Automate the running of programs over batches of files
-
- All of these are possible with some simple BASH scripting

Recording the output of programs

- Three data streams exist for all Linux programs
 - STDIN (Standard Input - a way to send data into the program)
 - STDOUT (Standard Output - a way to send expected data out of the program)
 - STDERR (Standard Error - a way to send errors or warnings out of the program)



- By default STDOUT and STDERR are connected to your shell, so when you see text coming from a program, it's coming from these streams.

Redirecting standard streams

- You can redirect using arrows at the end of your command
 - `> [file]` Redirects STDOUT
 - `2> [file]` Redirects STDERR
 - `2>&1` Sends STDERR into STDOUT

```
$ 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  
./bash_logout  
./local  
./local/share  
./local/share/icc  
./local/share/icc/edid-33d524c378824a7b78c6c679234da6b1.icc
```

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 you can write to but just discards all data sent to it
 - `might_fail > /dev/null`
 - `chatty_app 2> /dev/null`

Linking programs together with pipes

- Part of the original UNIX design was to have lots of small programs doing specific jobs, and then to link them together to perform more advanced tasks.
- Pipes are designed to do this by connecting STDOUT from one program to STDIN on another

Linking programs together using pipes

- Pipes are a mechanism to connect the STDOUT of one program to the STDIN of another. You can use them to build small pipelines
- To create a pipe just use a pipe character | between programs

```
$ ls | head -2
```

```
Data
```

```
Desktop
```

Useful programs for pipes

- Whilst you can theoretically use pipes to link any programs, there are some which are particularly useful, these are things 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

```
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 it is common to need to re-run the same command multiple times for different input/output files.
- Some programs will support being provided with multiple input files, but many will not.
- You can use the automation features of the BASH shell to automate the running of these types of programs

The BASH `for` loop

- Simple looping construct
 - Loop over a set of files
 - Loop over a set of values
- Creates a temporary environment variable which you can use when creating commands

Examples of `for` loops

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

```
for value in {5,10,20,50}
do
  run_simulation --iterations=$value > ${value}_iterations.log 2>&1
done
```

```
for value in {10..100}
do
  run_simulation --iterations=$value > ${value}_iterations.log 2>&1
done
```

Job Control

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

- 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

- `prog_to_run` (starts in foreground)
- `prog_to_run &` (starts in background)

- Control of running jobs
 - `jobs` (lists the jobs in this shell)
 - Control + Z (suspends the current job)
 - `bg` (puts the current suspended job into the background)
 - `fg [number]` (puts the selected job back in the foreground)

Job Control - nohup

- `nohup prog_to_run &`
 - Merges `STDOUT` and `STDERR`
 - Disconnects from the terminal
 - Can't be killed when the terminal exits
 - Output appended to `nohup.out`
 - Can redirect with `> logfile.txt`

Exercise 4