Introduction to Linux Part 1

Daniel Grose
STOR-i, Lancaster University

Lancaster University

EPSRC
Engineering and Physical Sciences Research Council

STOR-i
excellence with impact

What is Linux?

- Linux is a Free and Open Source Unix-like operating system
- Developed in 1991 by Linus Torvalds as a hobby whilst studying at the University of Helsinki
- Used in a wide variety of technology — servers, desktops, laptops, mobile phones, wristwatches
- Source code can be modified, used and redistributed
- Uses tools developed by the Free Software Foundation

excellence with impact

Some Linux Basics

- There is no one singular “Linux” operating system – there are dozens of different flavours (occasionally called distributions or “distros”)
- At the University, you’ll find Ubuntu, Fedora, CentOS, Debian, Redhat, BioLinux, Scientific Linux and many more
- They are similar in but not identical, so things that work in Ubuntu won’t necessarily work on CentOS
- In STOR-i, we primarily use Ubuntu, through the University’s MyLab service.
- Capitalisation is important! A is different to a, myFile.txt is different to myfile.txt

Features of a Unix-like OS

Multitasking –
Processes multiple jobs simultaneously, rather than one after another in a sequence

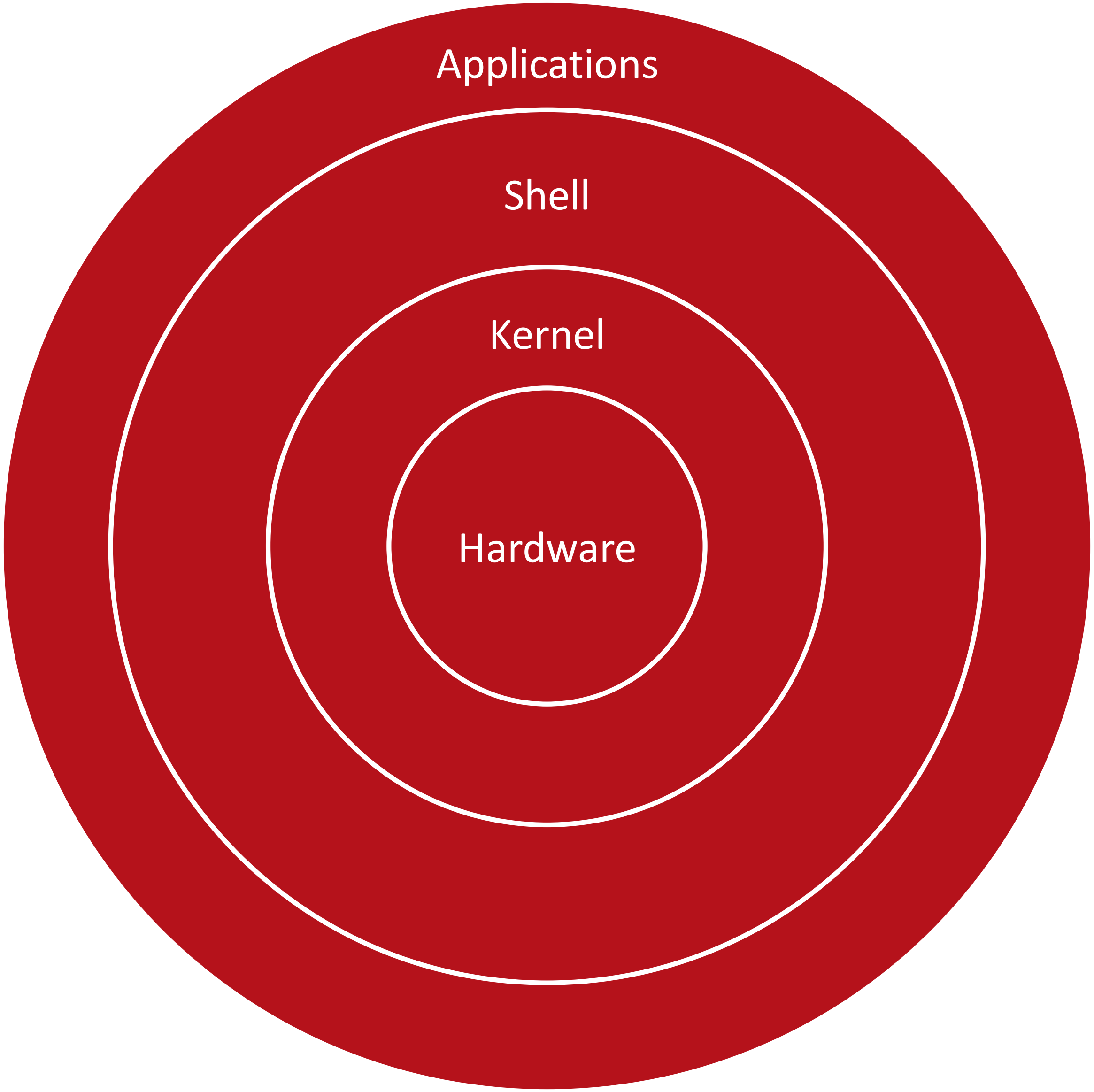
Multiuser –
Allows multiple users access to the system simultaneously

Networking –
Inter-machine communication and sharing of files and data

X Window System –
Also known as X11 or simply X, a windowing system that is highly customisable

excellence with impact

Components of a Unix-like OS



Kernel

- The core of the operating system
- Manages devices, memory, user and system processes
- Schedules the use of system resources
- Stores information about the computer system and the networks it’s connected to
- Directly interacts with the hardware and prevents other processes from doing so

Shell

- The shell is a utility that provides an interface between the user and the kernel
- Shells act as command interpreters of user input
- Shells are Command Line Interfaces (CLIs)
- There are many different shells, but we'll be using the Bourne Again Shell, or bash
- More on this later!

Files and Filesystems

Filesystem

- In Linux, everything is a file
- This includes devices, configuration files, applications etc.
- Linux is an extensionless operating system, so file extensions such as .txt aren’t required, but are usually added for simplicity and compatibility
- A file is hidden if it starts with a .
- All files appear under the root directory, /
- /dev contains devices
- /etc contains configuration files
- /home contains user files and profiles
- /bin contains installed binaries (applications that can be executed)

File Attributes and Permissions

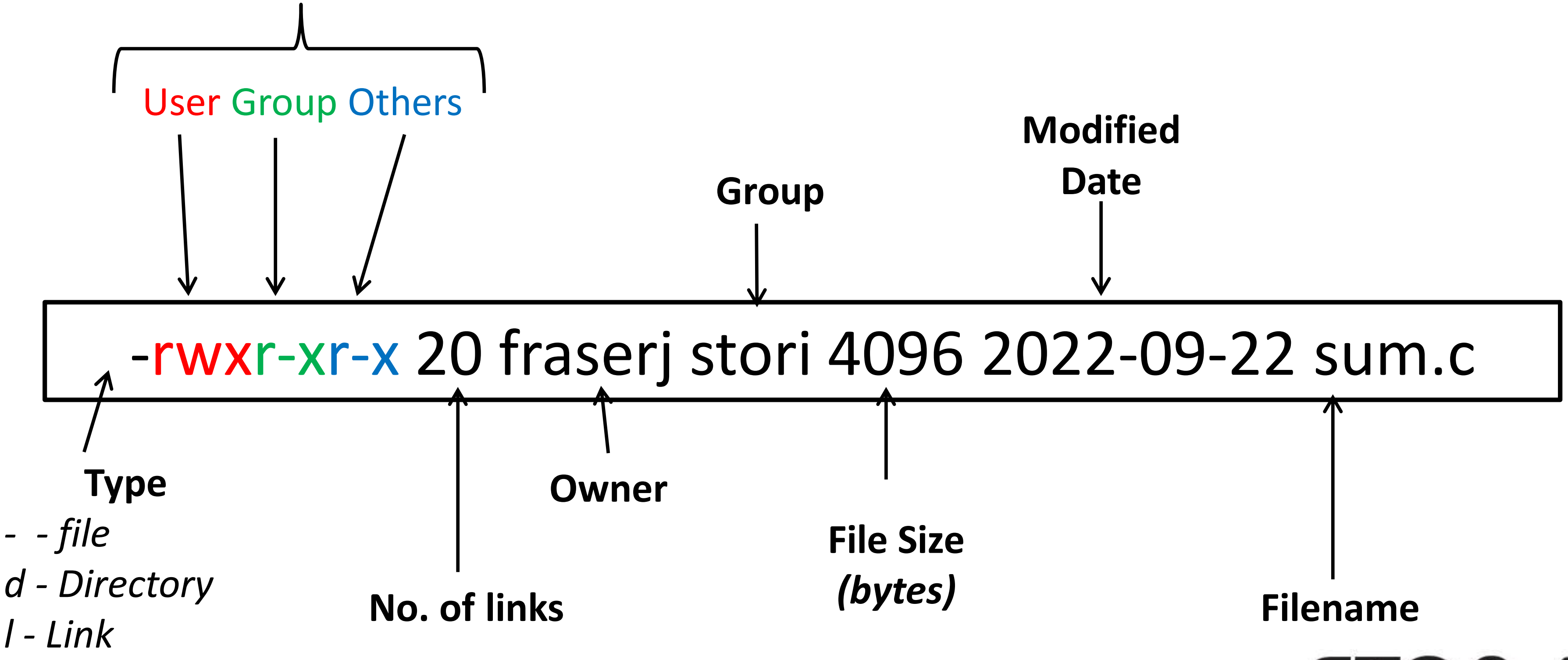
Permissions

r - Read

w - Write

x - Execute

- - No Permission



User Group Others

```
-rwxr-xr-x 20 fraserj stori 4096 2022-09-22 sum.c
```

Group

Modified Date

Type

- - file

d - Directory

l - Link

No. of links

Owner

File Size *(bytes)*

Filename

Commands and the Command Line

Command Syntax

- Most commands can be given options and arguments
- Commands in these slides will be in the following format:
  $ command [-options] <arguments>
- $ is the last character of the command prompt, you don’t need to type this!
- Square brackets refer to options (i.e. not mandatory)
- Arguments are given in <>

Basic Commands

- ls – list files
- cp – copy files
- mv – move files
- rm – remove files
- cd – change directory
- mkdir – make directory
- rmdir – remove directory
- ln – create link to a file

ls – List files

- Used to list files and directories and their attributes
- To list files in the current directory:
  $ ls
- To list files in another directory (relatively):
  $ ls ./scripts
- To list files in another directory (absolutely):
  $ ls /home/fraserj/scripts
- Useful flags:
  -l – includes a lot more information about files
  -a – lists all files, including hidden files
  -c – lists files in columns
  -h – list sizes in “human readable” format

STOR-i
excellence with impact

cp, mv, rm – Copy, move and remove

- **$ cp <source> <destination>** creates a copy of the object specified as source at the location specified by destination
  - -r – recursive, copies all files in subdirectories
  - -i – interactive, prompts before overwriting
  - -u – update, only copies when the source file is newer than the destination
- **$ mv <source> <destination>** moves an object from the source to the destination. Also how you rename files in Linux (you move the file into its new name)
- **$ rm <target1> <target2>** removes the files specified as target1 and target 2
  - Can use the same -i and -r flags as cp
  - Beware, rm is potentially dangerous. Deleted files cannot be recovered

cd, mkdir, rmdir - Directories

- **$ cd <directory>** changes the directory to the specified directory
  - Each directory contains two “special” directories, . and ..
  - . refers to the current directory
  - .. refers to the current directory’s parent directory
  - Can be either relative or absolute
- **$ mkdir <directory>** makes a new directory in the current directory with the name specified
- **$ rmdir <directory>** removes the *empty* directory specified
  - You cannot remove a directory with files in it unless you specify the -r flag, which also removes its contents

Useful Command Line Tips

- **$ pwd** – Print Working Directory. This command prints the full path of the current directory.
- . and .. refer to the current and parent directories respectively
- Tab key – Autocompletes file and directory names. Double tapping it prints a list of files and directories at the current path. Useful for navigating through file structures.
- Up/Down arrow keys – Allow you to look back through your command history.
- Ctrl+R – Know you've done something before but can't remember the full command? Press Ctrl-R and start typing part of it and let Linux find it for you!
- Ctrl+C – Break. Stop whatever command you're currently running, or if you're not, cancel the command and move onto a new command line.
- Middle click – Paste whatever you have highlighted.

Getting more help

- Linux has an extensive help system called the manual, which can be looked at using the man command
  - **$ man <command>** will open the manual page for the command specified
  - **$ man ls**
  - **$ man cd**
- Man pages list command options, syntax and some examples of use
- Whatis command gives a brief description of a command
  - **$ whatis ls**
- Google!

Exercises

- List files in your current directory
- List all files in your current directory (including hidden)
- Display the full path of the current directory
- Make a new directory called linux01
- Move into the new directory
- Use the *touch* command to create a new file called myFile
- Create a copy of myFile called myFile2
- Rename myFile2 to oldFile
- Delete oldFile
- Use the man command to work out what *ls -p* does
- What are the permissions of myFile?

Using the Shell

excellence with impact

Shell

- The shell is a program that interprets commands and acts as an interface between the user and the kernel
- In practice, we something to interact with the shell, called a terminal
- The terminal allows us to enter commands into the shell, which can be either a file, or instructions internal to it

Shell vs Terminal

- Often used interchangeably, even though they’re not!
- The shell is the program that interprets your commands provided through the terminal
- A terminal can be physical or emulated
- The Terminal application in Ubuntu is an example of a terminal emulator

PATH

- The shell retains an internal list of directories to search for binaries, stored in the PATH environment variable
- To view the directories in PATH, you can use
  $ echo $PATH
- To execute a binary not on the path, you have to specify the path e.g.
  $ ./runme

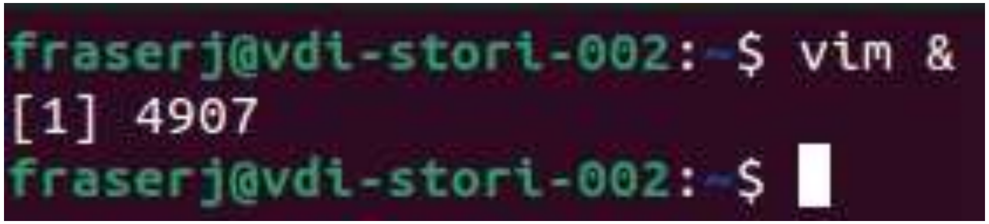
excellence with impact

Processes

- When you enter a command into a terminal, the shell creates a job, which executes the command
- Processes can exist in one of four states:
  - Foreground – The default state. All jobs run in the foreground unless specified otherwise
  - Background – The job runs, but the shell remains interactive
  - Stopped – A stopped job can be resumed in a different state
  - Terminated – A terminated job cannot be resumed

Starting a background job

- Running jobs in the background allows the user to continue using the parent shell

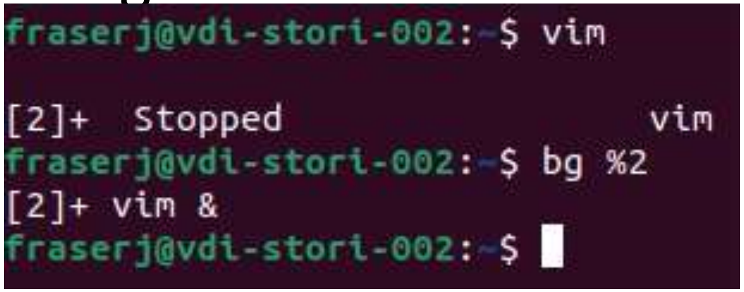


```
fraserj@vdi-stori-002:~$ vim &
[1] 4907
fraserj@vdi-stori-002:~$ 
```

- The shell returns a job id(1) and a process id (4907)
- To obtain a list of currently running jobs, you can use the **$ jobs** command

Stopping a foreground process

- You can stop a job by pressing Ctrl+Z whilst the job is running in the foreground



```
fraserj@vdi-stori-002:~$ vim

[2]+  Stopped                 vim
fraserj@vdi-stori-002:~$ bg %2
[2]+ vim &
fraserj@vdi-stori-002:~$ 
```

- This will suspend the job
- You can then start it in the background by using the **$ bg %<job number>** command
- You can restart it in the foreground by using the **$ fg %<job number>** command

Terminating a process

- You can terminate a job that is running in the foreground by using Ctrl+C
- You can terminate a job that is running in the background using the **$ kill** command
- You specify either the job number or the process id:
  - **$ kill %1** terminates the job 1
  - **$ kill 2894** terminates the process 2894
- These will immediately terminate the process, discarding any results or current processing

Process management

- The operating system maintains a list of all processes called the processes table
- Each process has a large amount of information related to it such as:
  - Process ID (PID)
  - Owner
  - Start time
  - Execution priority
- The command for viewing the processes table is
  $ top
- You can view a summary of just your processes by using the **$ ps** command

Exercises

- Start vi in the background
- Terminate vi whilst it is still a background process
- Find the PID of the current shell
- List all processes currently running as root
- Start vi in the background again. Use the processes table to determine how much memory is being used by vi.
- Terminate the vi from the processes table
- Quit the processes table

Interacting with Files

Viewing Files

- There are numerous tools for looking at the content of a file
  - **$ cat** concatenates files and prints them to screen
  - **$ more** allows you to view the contents of a file bit by bit
  - **$ less** is the same as more, but with better navigation and more functionality
  - **$ head** prints the first 10 lines of a file to screen
  - **$ tail** prints the last 10 lines of a file to screen
  - **$ nl** prints the file to the screen with line numbers added

STDIN, STDOUT and STDERR

- These refer to input and output streams
- STDIN is where a command takes input from that it requires
  - Usually, this is the current terminal and you type the input and hit return
- STDOUT is where the command sends its output to
  - Usually, this prints out to the screen
- STDERR is where a command sends its error output to
  - Usually, this isn't displayed at all or is sent to the screen
- They can be redirected using their file descriptors 0, 1 and 2 respectively

Redirection

- We can use redirection to direct the output of a command to a file rather than STDOUT:
  - **$ ls > listing.txt**
- This puts the results of the current directory listing into the file called listing.txt, overwriting any existing content, and creating it if necessary
- Similarly, for commands that require inputs, you can specify an input file rather than from STDIN:
  - **$ wc -l < listing.txt**
- This counts the number of lines in listing.txt and prints the number to screen

More Redirection

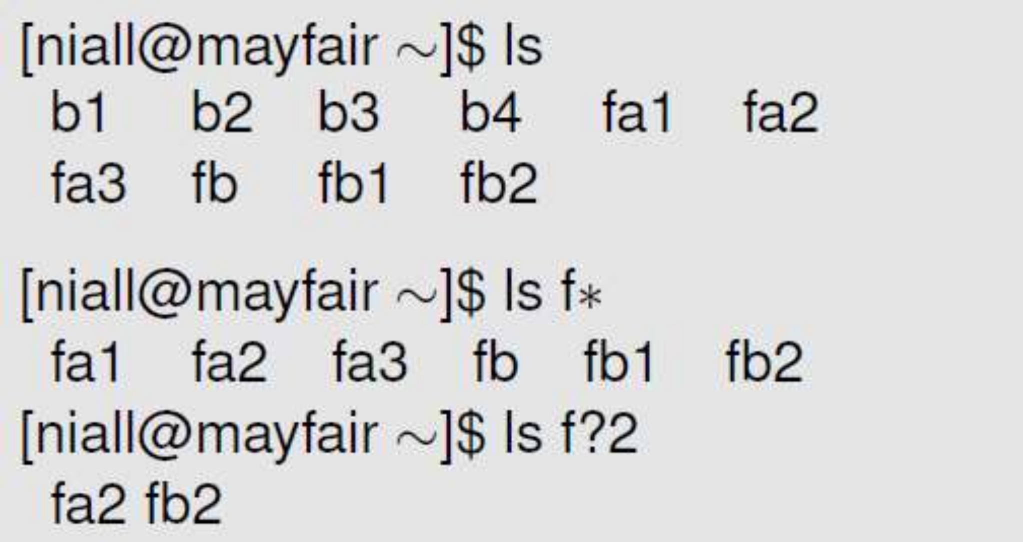
- Sometimes, we'll want to add content to the end of a file, rather than overwriting it
- This is done through the append redirection:
  - **$ ls >> listing.txt**
- Sometimes, you might want to not see any output at all.
- This is done by redirecting STDOUT and STDERR to /dev/null:
  - **$ls > /dev/null 2>&1**

Special Characters

- Like most operating systems, Linux has a number of special characters, the majority of which offer additional powerful functionality within a shell
- ; $ % > < ! ~ [ ] ( ) | / ‘ “ *
- **Don’t use these characters in filenames**
  - – We’ve already used < > & and $

Wildcards

- A wildcard character represents “any other character”
  - * represents zero or more characters
  - ? Represents any single character

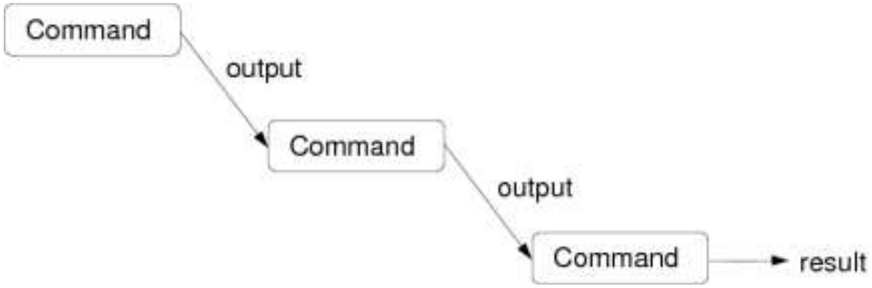


```
[niall@mayfair ~]$ ls
 b1    b2   b3    b4    fa1   fa2
 fa3   fb   fb1   fb2

[niall@mayfair ~]$ ls f∗
 fa1   fa2   fa3   fb   fb1   fb2
[niall@mayfair ~]$ ls f?2
 fa2 fb2
```

Pipelines

- A pipeline is a mechanism for feeding the output of one command into the input for another. The symbol for this is called a pipe |



- The | symbol is placed between commands
- **$ ls /dev | less**

Filters

- Filters are commands that read an input from a pipeline, process it and produce an output
- Some simple filters include:
  - sort (reorder the output)
  - tail (extract lines from the end of the output)
  - more (pass through pages of the input one at a time)
- Sort the contents of a file and save the results in a new file:
  - **$ cat file.dat | sort -n > newfile.dat**

excellence with impact

Exercises

- Run the following command:

  -
    ```
    $ wget www.lancs.ac.uk/~fraserj/stor601/cathedral-bazaar.txt
    ```
- View the contents of the downloaded file
- View the last 15 lines of the file using tail
- Run the previous command but put the output in a txt file
- List all files and permissions in current directory and **append** to the previously created txt file
- Run the following command

  -
    ```
    $ wget www.lancs.ac.uk/~fraserj/stor601/names.txt
    ```
- View the file using cat
- Sort into alphabetical order
- Sort into reverse order, remove duplicates and write last 10 lines to new_names.txt
- List all files in /bin (including hidden) and sort by size. (hint: -k -n flags)

Archiving and Aliasing

Archiving

- Sometimes, it’s useful to be able to collect a group of files into a single file (e.g. backup, file transfer etc.)
- Linux has a utility called tar (Tape Archiver) to do this:
- **$ tar cvf tarname.tar <file1> <file2>**
- Options here are:
  - c – Create new archive
  - v – Verbose mode. Shows more information in the output
  - f – Use the filename specified in the first argument

Compression

- tar can also be used to compress files and directories
- tar can compress files using gzip or bzip2
- gzip – Append z to the options
- bzip2 – Append j to the options
- **$ tar zxvf filename.tar.gz**
  - Extract from gzip file into the current directory
- **$ tar jcvf filename.tar.bz2 *.txt**
  - Compress all files with a .txt extension into a new bzip2 file

Environment Variables

- Environment variables store information about the Linux environment. They have two parts:
  - Variable name
  - Variable Value
- To view all the current environment variables, you can use the **$ env** command (long list!)
- You can see the value of a particular variable by using **$ echo $<NAME>** as we did with the PATH variable earlier
- You can create your own variables:
  - **$ myvar='variable value'**

excellence with impact

Aliasing

- Shells provide a mechanism to create new commands using a system called aliasing
- A command alias allows you to substitute a long command for a short one, or a series of commands as a single one:
  - **$ alias alias-name=‘command [options]’**
  - **$ alias rm=‘rm -i’**
  - **$ alias home=‘cd ~;ls’**
- Aliases are only valid for the current shell session
- You can make them permanent by adding them into a shell configuration

Initialising the Shell

- You can configure the shell to set up your working environment every time you connect to it
- Three major configuration files for bash:
  - /etc/bash.bashrc – System wide configuration, applies to all users
  - ~/.bash_profile – Personal profile that is processed whenever you launch a login shell
  - ~/.bashrc – Personal profile that is processed every time you launch a non-login shell

Command History

- You can view the full list of commands you’ve previously executed by using the **$ history** command
- There are a number of history shortcuts
- Be careful. Using **$!r** for example will execute the last command starting with r. This could be **$ rsync** (more on this tomorrow!) or possibly **$ rm -rf / --no-preserve-root** (**DO NOT RUN THIS ON A SYSTEM YOU DON’T INTEND TO BREAK!**)

Exercises

- View the contents of the variables SHELL, USER, LANG
- Create a variable called ‘myage’ and give it a value
- Print the value of your variable to screen
- Create a backup of everything in your home directory using tar
- Delete that archive, create it again using gzip compression
- View all the files in the archive (hint: -t flag)
- Extract everything in the archive to /tmp/restored
- Create an alias called home that takes you to your home directory and automatically lists all files
- Make the alias permanent
- View all your command history, one page at a time