UNIVERSITY OF CAMBRIDGE

MRC | Cognition and Brain Sciences Unit

Introduction to Computing at the CBU
Linux, The Cluster, and the CLI
Part 1

Prerequisites

- **A CBU IT account**
  - needed to access any CBU computing resources
  - including access the cluster
- **Membership in the 'Imagers' group**
  - to give your account the permissions needed to access some cluster resources
- **A Connection to the Cluster**
  - attendees should already have determined that they can connect to the cluster using ssh, x2go, or vnc
  - attendees should be connected at the start of the class

- **If you lack the prerequisites**
  - you may still follow along but will be unable to do the 'practical' exercises

Motivation – Why do you need this?

- **Shared CBSU environment**
  - Central storage: images, scripts, documents, etc.
    - Backup (automatic)

- **Computing Cluster**
  - High-Performance Computing – good for analysis
    - Speed (especially once you learn a few “tricks”)
    - Batch processing for even more throughput

What the Cluster Is ...

The cluster is a group of powerful computing machines intended primarily to fulfill computing needs which are too heavy for desktop machines. The parts of the cluster are called nodes:

- login nodes are available for users to directly log on to, in order to test code, visualize data, and so on.
- compute nodes do not allow users to log on, they are used for batch submission via SLURM (which is beyond the scope of this course)
  - Documentation is on the intranet:

  http://intranet.mrc-cbu.cam.ac.uk/intranet_category/compute-cluster-2019/
- quick status check of the cluster (good for picking a login node):

  http://master02.mrc-cbu.cam.ac.uk/

Ways and Means

- **Graphical Desktop Environment via X2GO (or VNC)**
  - Similar to other desktop OS GUIs (Windows, Mac)
  - Menu-driven
  - Can be slow, especially when operating on many files
  - Available on cluster login nodes, but not on compute nodes

- **Command Line Interface (CLI)**
  - Via ‘terminal’
  - Similar to DOS command prompt
  - Available natively in Mac OS X (Mac OS X sits on top of a BSD Unix)
  - Fast, once you learn how it works
  - Commands may be used for scripts on compute (or login) nodes

Getting Started

Connected, Now What?

- **Some Linux Conventions and Things to Know**
  - linux is case sensitive – Home, HOME, and home are different
  - Directory is a hierarchical tree starting at root (= /)
    - /
      - /home
        - /home/zz99
      - /var
        - /var/log
    - . is the current directory
    - .. is the parent directory
    - ~ is the logged in user's home directory

```
[zz99@login-x59 ~]$
```

username machine ↑ Your home directory!
name

Getting Started

Connected, Now What?

- **Useful commands to try**
  - *pwd* – print working directory (where am I?)
    - note the full path starting with /
  - *cd* – change directory. Give it an argument after a space
    - *cd ..* (change to parent directory) then try pwd again
    - *cd ~* (change to home directory) then pwd
  - *ls* – list the contents of a directory, your home directory doesn't have much in it
    - the parent directory does, try '*ls ..*'

What the Cluster Is, Part 2 ...

- All nodes utilize shared filesystems – home spaces and imaging spaces.
  - Shared filesystems are automounted – which means they are mounted 'on demand' when they are accessed. Before they are explicitly referred to they are 'invisible.'
  - viewing the contents of a parent directory will not show the possible mountable subdirectories, only the ones currently mounted
  - a file browser will not show unmounted directories either
  - linux 'ls' or 'cd' commands using the desired directory will force the mount
  - a filebrowser will need the directory explicitly entered

  NB: home directories are also automounted. When you login, you are put in that directory, and so the mount happens when you login

Getting Started

Connected, Now What?

- **Many groups have a shared space in /imaging or /group**
  - *ls /imaging*
  - *ls /group*
    - if no one has accessed a group directory it won't be visible
    - if you have permissions to access a group directory you can cd or ls on the full path to mount it
    - if you don't have the right permissions, it will mount, but give you a permission denied error
    - if you need access to a group directory, ask your supervisor/manager to contact IT to request access for you

Getting Started

Connected, Now What?

- **Some projects also have a shared space in /imaging/projects**
  - *ls /imaging/projects*
    - project directories are automounted
    - if you need access to a project directory, ask your supervisor/manager to contact IT to request access for you

- **Training materials can sometimes be found in /imaging/training**

Getting Started

Connected, Now What?

- **More useful stuff**
  - most commands take options/flags
    - -X - a single dash and a single letter for short form
    - --Xpect – a double dash and a word for long form
  - some examples:
    - *ls -F*
      - note the trailing "/" in the output which indicates a directory
    - *cd /imaging/training/LinuxClass*
    - *ls*
    - *cd FixFiles*
      - this is a relative path, changing into a directory relative to the current directory
      - tab completion - type *cd Fix* and then hit the tab key, and it should autocomplete

Getting Started

Connected, Now What?

- **Special characters, including wildcards**
  - several characters have special meaning to the operating system and should be avoided in things like file and directory names:
    - <> are used for io redirection
    - $ is used for variables, & is used to control backgrounding processes
    - * is the wildcard character, it is used in commands to represent any arbitrary string of characters. Try:
      - ls -d /etc
      - ls -d /etc/b*
      - ls -d /etc/[c-d]*

Getting Started

Connected, Now What?

- **Gotchas for those familiar with other Operating Systems**
  - whitespace is used to separate commands and arguments.
    - `ls My Directory` – this has two arguments My and Directory
    - `ls "My Directory"` – has been quoted and is now a single argument
    - General style is to use capitalisation, hyphens, or underscores rather than spaces: MyDirectory, My_Directory, my-directory, my_directory
  - filename extensions are optional and mostly irrelevant (but useful)
    - an executable does not require .exe, a python script does not require .py, a jpeg does not require .jpg or .jpeg, etc.
    - However! Using meaningful extensions improves clarity
  - the dot/full stop "." has no particular meaning to the operating system
      - gcc.4.7.11.tar.gz is a perfectly valid filename

Getting Started

Connected, Now What?

- **Things you can do and learn with ls**
  - *ls -al*
    - multiple flags only need a single leading -
    - -a = all files, -l = long listing
    - the command '*man ls*' will show the manual page for ls. 'man' works for (almost) any command

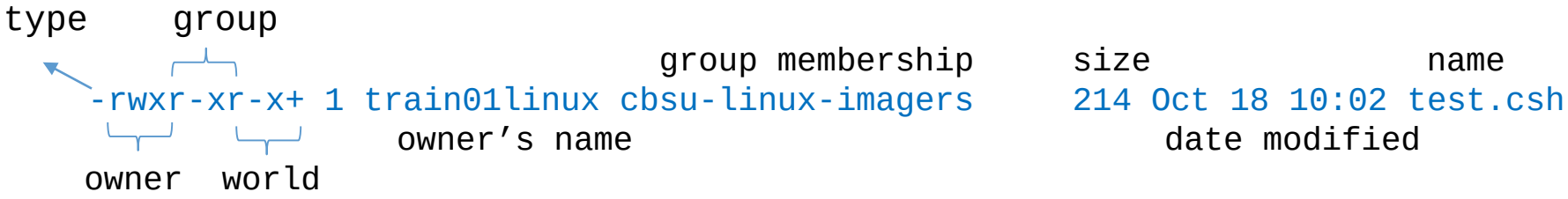
```
drwxr-sr-x+ 2 train01linux cbsu-linux-imagers      12 Oct 18 10:02 .
drwxr-s---+ 3 train01linux cbsu-linux-imagers       3 Oct 18 10:02 ..
-rwx------+ 1 train01linux cbsu-linux-imagers  385158 Oct 18 10:02 dicom.dcm
```

Notice the two special directories (. and ..) are listed, thanks to -a. As for the rest …

Getting Started

Linux Filenames, Users, Groups, and Permissions

- Every user has a username and one or more group memberships
- Files can have arbitrary names – extensions are not required
- Files have an owner and a single group membership
- Files (and directories) have three sets of permissions
  - owner, group, other/world
- Files (and directories) have three types of permissions
  - read, write, execute



- NB: files may also have access control lists (ACLs) but those are beyond the scope of this class

Shells, Environments, and More

- **Shell**
  - Encapsulated ("in a shell") environment
    - environment variables – *env* command
      - 'pipe' the output of one command to the input of another
        - *env | more*
      - redirect the output to a file
        - *env > YourFileHere*
        - or append it with >>
      - look at output with an editor or
        - *cat FileName* (and if it is more than one page ...)
    - *echo $OS*
    - *echo $PATH*
    - *echo $SHELL*

Shells, Environments, and More

Command editor

- up and down arrows
- tab completion
- command line is editable
- !<part command> - execute most recent matching command from history
  - *ls -alt; cd .. ; !ls*
- Break: Ctrl+C
  - terminate running program and return to prompt

Shells, Environments, and More

- **Variables**
  - environment variables propagate – regular variables don't
    - *set MYVAR="LocalVar"*
    - *echo $MYVAR*
    - *tcsh*
    - *echo $MYVAR*
    - *exit*
    - *unset $MYVAR*
    - *setenv MYVAR "EnvVar"*
    - *echo $MYVAR*
    - *tcsh*
    - *echo $MYVAR*

Shells, Environments, and More

Aliases:

- short-cuts to command/script/program
- *alias* (with no arguments to show existing aliases)
- *alias <alias> <aliased command>*
  - *alias cdi 'cd /imaging/training/LinuxClass'*
  - *cd ~; ls ; cdi; ls*
- *unalias <alias>*
  - *unalias cdi; cdi*

Unix Primer – Make your life easier!

- **File browser - nautilus**
  - Create Bookmarks
  - Once a browser, always a browser

- **Symbolic links**
  - References to folders/files – not a copy
  - Create: *ln -s <original> <link>*
  - Delete: *rm <link>, unlink <link>*

Unix Primer – Useful Commands

Help/Info	*man, -h, --help*	help MANual
	man -k	find similar or related commands
	pwd	Present(print) Working Directory
	echo	display text or variable
Listing	*ls*	LiSt directory contents
	cat	conCATenate file to display
	top	show processes
	id -G -n <username>	list group memberships
Searching	*locate*	find files
	which	which executable will a command use
Navigate	*ssh*	connect to a linux machine
	cd	Change Directory

Unix Primer – Useful Commands

Manipulate	*mkdir*	MaKe Directory
	cp (-r)	CoPy (-r for recursively)
	mv	MoVe/rename
	rm (-r)	ReMove/delete irreversibly (and -r)
	chown (-R)	CHange OWNer (-R for recursively)
	chmod (-R)	CHange MODe/permissions (and -R)
	vncserver	create/kill VNC session
	dos2unix, unix2dos	convert text file (EOL)
	wget <URL>	download
	tar	Tape ARchive – -c create tar archive file -x extract from tar file -z compress('zip') or uncompress

Unix Primer – Useful Commands (toolkit)

grep <filename>	Print lines with pattern fit
grep -c pttrn <filename>	Print number of lines with pattern fit
awk '{print $2}' <filename>	Print second entry only (of each line)
sed 's/ptrn1/ptrn2/' <filename>	Print content with ptrn1 replaced with ptrn2 (first instance only)
sed 's/ptrn1/ptrn2/g' <filename>	Print content with ptrn1 replaced with ptrn2 (globally)
printf "format" inpt	Formatted print (~ MATLAB)
echo "statement" | bc	Calculator
seq n	Sequence of integer up to n (useful for loops)

Further Information

- **Linux:**
  - Basics:
    - http://imaging.mrc-cbu.cam.ac.uk/methods/unixsurvivalguide
    - http://www.ee.surrey.ac.uk/Teaching/Unix
  - Scripting:
    - http://tldp.org/LDP (both Beginner and Advanced)

UNIVERSITY OF CAMBRIDGE

MRC | Cognition and Brain Sciences Unit

Introduction to Linux at the CBSU
Part 2
Advanced Concepts and Scripting

Previously on Intro to Linux ...

- **Basic Commands**
  - cd
  - pwd
  - ls
  - man
  - (and so on)

Files, Permissions, and Owners, Oh My!

- **Users and Groups**
  - Unique Username and User ID (UID)
    - *whoami; id –un; id -u*
  - May be member of multiple groups
    - *id -gn; id -G*

- **Files and Directories**
  - Single owner and single group
    - altered with *chown*
  - Three sets of permissions and three kinds of permissions
    - owner, group, and world(all)
    - read (r), write (w), execute (x)

Files Permissions and Owners

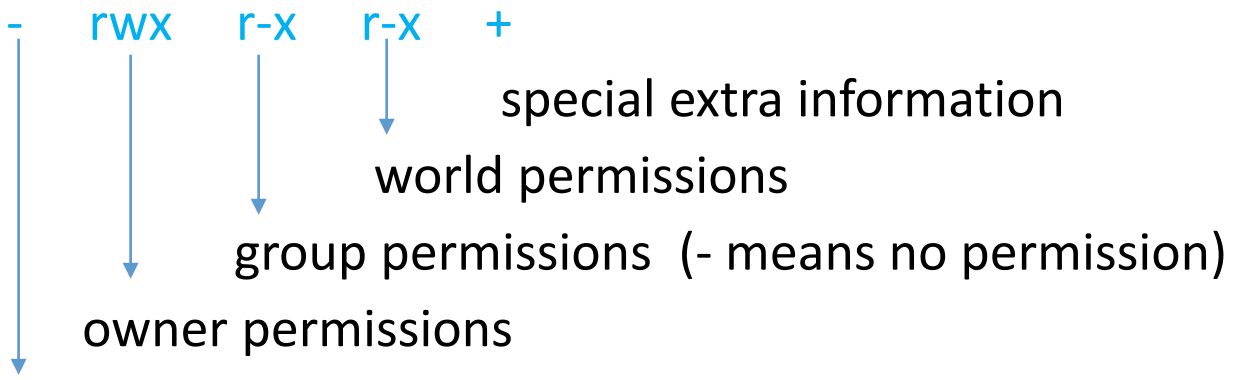
ls –n

		uid	gid		timestamp		name
-rwxr-xr-x+	1	6831	222	214	Oct 25	2016	test.csh
permissions				size			

number of links (usually 1 for files, for directories is a count of subdirectories and current directory)

More complicated permissions and ownerships are possible but beyond the scope of this class. ACLs (Access Control Lists) also allow for finer control, but are beyond the scope of this class.

Files Permissions and Owners



Permissions often referred to as octal numbers

- read = 4, write = 2, execute = 1
- rwx = 7, r-x = 5, rw- = 6
- 755 and 664 are common
- anything world-writable is scary

Programs and Executables

- **Executables must have executable permission set**
  - group execute is sufficient if user is member of group
  - file must also be something which can be executed
    - a script
    - a compiled program

- **The shell must be able to find the code**
    - in the path ($PATH)
    - or given explicit path
      - fully qualified – starting at /
      - relative – eg ./*executable* for current directory

Programs and Executables

- **Managing $PATH**
  - The $PATH is a list of directories which the OS will search in order looking for a command if no fully qualified path is given
  - The "which" command will tell you which command it finds
    - "which mkdir" returns “/usr/bin/mkdir" since “/usr/bin" is in $PATH
    - a response of "command" not found indicates that no command with that name is in $PATH
  - modules and conda can help to manage paths when multiple versions of a command exist, or they are located in non-standard places.
    - Documentation is on the intranet:
      http://intranet.mrc-cbu.cam.ac.uk/home/cluster2019-other/

A Few Preliminaries

- **Working Directories**
  - directories and subdirectories help to organise work and projects
  - create a directory in your home space for this class using ‘mkdir’
    - cd ~
    - mkdir LinuxClass
  - copy the class files to your directory using ‘cp’ with a flag for recursion and a wildcard for the filenames
    - cp -r /imaging/training/LinuxClass/* LinuxClass

Why Bother?

Accuracy and Ease

- **Shell Scripts Can Contain lists of commands**
  - **Example**
    - You want to set up datestamped working directory in your home space. One way is to change directory to your home, figuring out the day and then use mkdir to create the directory.
      - cd ~
      - set DATE = `date +%y%m%d`
      - set WORKDIR = “work”${DATE}
      - mkdir $WORKDIR
    - The file is called ‘makework’ in your LinuxClass directory

Why Bother?

Speed and Efficiency

- **Loops and Iteration**
  - **Example**
    - You've run some code (DataGen.csh) and generated a lot of output files
      - *ls SampleOutputDir*
      - How many files? Hint: use *ls* (with a flag), *wc*, and a pipe.)
    - Now you want to re-run the code with a slight change
      - save old data
      - look for changes
    - Script it up!
      - retain existing files, date-stamped in some way
      - run the code
      - compare the new output with the old
        - Different files?
        - Differences in files with the same name?

Why Bother?

Speed and Efficiency

- **Example (cont.)**
  - Rerun.csh should to the job
    - *cat Rerun.csh | more* (or *more Rerun.csh*)
  - set variables – change them to address changing circumstances
  - loop over files
  - creates a shell with working directories, environment, and so forth
    - Remember environment variables propagate – so your shell environment will propagate down, but …
    - if you are submitting a job remotely to the cluster, the environment will be different, so …
    - good practice is to set whatever you know you need in the script itself.

Have a go …

Try this:

- make a directory in your class directory called ‘SampleOutputDir’
- run Datagen.csh
- have a look in SampleOutputDir
- change the names of a few files, or remove a few, or add new ones (explore the ‘touch’ command), copy one file over another
- run Rerun.csh

Remember:

Linux is case-sensitive

a command must be in your $PATH or be fully qualified. ("." and "~" are handy for this)

Compilation and Interpretation

There are (broadly speaking) two kinds of executable files

- scripts, which usually means interpretable files
  - scripts are evaluated as they are executed
  - single step process
  - sometimes line by line
  - slow but portable
- programs, which usually means compiled files
  - programs are compiled before they are executed
  - two step process – compile and run
  - fast but need to be compiled on the platform where they will be run.

Scripting languages

So many options:

Scripts can be written using only shell commands, or can be written in one of any number of scripting languages such as:

- perl
- python
- php
- even (shudder) PowerShell
- and many more

Each has advantages, disadvantages, proponents, and detractors.

Simple example

Example:

test.csh, test.pl, and test.py all do the same thing.
Each does similar things in different ways internally.

For simple tasks, a simple shell script is good - the commands are just linux commands in a file. (Mostly.)

For more complex tasks, a more fully featured scripting language is often preferable, although it will require some programming skill.

The more you want to manipulate data, the more likely it is that a scripting language is the right choice.

A Mostly Real World Example

There was a directory on our website with pictures of staff with names in the format firstname.lastname.pdf – a new system wanted the format of firstnamelastname.pdf – ie without the middle '.'

It could be done by hand using mv to rename the files (or cp to copy them with the new name), but for a couple of hundred files that is both time consuming and error-prone.

The directory FixFiles in the LinuxClass directory replicates this problem.

A Mostly Real World Example

A python script called ‘genfiles.py’ is used to generate a bunch of dummy filenames, all ending with .dwarf

The simple shell script in the src directory, fixname.sh, does what we want it to do.

Further Information

- **Linux:**
  - Basics:
    - http://imaging.mrc-cbu.cam.ac.uk/methods/unixsurvivalguide
    - http://www.ee.surrey.ac.uk/Teaching/Unix
  - Scripting:
    - http://tldp.org/LDP (both Beginner and Advanced)