UNIVERSITY OF CAMBRIDGE

MRC | Cognition and Brain Sciences Unit

Using Linux, the CLI, and SLURM on the CBU Cluster

November Edition

Version 202611

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
  - needed to use SLURM and compute nodes
- **A Connection to the Cluster**
  - attendees should already have determined that they can connect to the cluster using ssh, x2go, MobaXterm, vnc, or something similar
  - on personal machines, this will require a vpn connection
  - 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

Prerequisites

- **Checking your group membership**
  - log in to a login node of your choice
  - get command line access
    - directly with ssh, etc
    - open a terminal if using x2go, vnc, etc
  - check your group membership with this command
    - id -Gn
    - look for 'cbsu-linux-imagers'
  - if you try to use SLURM without membership you will get an error
    - Batch job submission failed: Invalid account or account/partition combination specified

Advantages of the Cluster

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

- **Computing Cluster**
  - High-Performance Computing – good for analysis
    - Speed
    - Batch processing via SLURM for even more throughput

- **Many Processing Packages and Languages Available**
  - Matlab, fsl, R, python, conda, etc
  - multiple versions of many packages are available

What the Cluster Is ...

The cluster is a group of powerful computing machines intended primarily to fulfil 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 (about which more later)
- Documentation is on the intranet under Compute Cluster (2024)
- 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
  - visualisation tools are available, with various degrees of compatibility
  - requires client on local machine

Ways and Means

- **Command Line Interface (CLI) via terminal, direct ssh, or MobaXterm**
  - Similar to Windows/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
  - Can use visualisation tools if using MobaXterm

Core Concepts – the shell

- **shell**
  - command line interface (CLI) to the operating system
  - provides features like command editing/completion
  - has scripting ability
    - limited programming syntax
    - variables
  - by default users are given tcsh
    - other shells are available, eg bash

Core Concepts – linux CLI syntax

- linux is case sensitive – Home, HOME, and home are different
- spaces matter – spaces are used to separate arguments on the command line
  - avoid spaces in names of files and directories
- filename extensions are a convention, not a requirement – it's good practice to name your python scripts with .py and so on, but sometimes people don't.

Core Concepts – linux CLI syntax

- The CLI is used by entering commands in the terminal window
- again, spaces matter! They delineate arguments to commands (and the command itself)
  - (This is why paths and filenames with spaces in them are annoying)
- The return key completes the command line and tells the OS to process it

  eg: ls -l /etc

  'ls' is the command
  -l is an option flag
  /etc is an argument

  when return is pressed, the command executes

Core Concepts – the environment

- **environment**
  - the environment in which your terminal/shell session operates
  - includes environment variables, which affect how your session interacts with the shell and the operating system
    - PATH is a very important environment variable
    - there are many others
  - controlling and manipulating your environment is key. Tools to help include:
    - conda
    - modules
    - python venv

Core Concepts – the environment

- **environment**
  - the ‘env’ command shows all the variables which are set and which define your environment
  - ‘echo $<variablename>’ will echo the value of the variable to the terminal

```
[zz01@login-k01 ~]$ echo $SHELL
/bin/tcsh
```

PATH

- **Managing $PATH – the PATH environment variable**
  - The $PATH is a list of directories which the OS will search looking for a command if no fully qualified path is given
    - the directories will be searched in order
  - The " which" command will tell you which command it finds
    - "which mkdir" returns “/usr/bin/mkdir" since “/usr/bin" is in $PATH
    - "command not found " says 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:
    https://intranet.mrc-cbu.cam.ac.uk/home/cbu-cluster-2024-managing-environments/

CLI Command editor

Command editor

- up and down arrows
- tab completion
- command line is editable
- Break: Ctrl+C
  - terminate running program and return to prompt

Getting Started

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 name ↑ Your home directory!

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 might not have much in it
  - the parent directory does, try '*ls ..*'

Cluster - automounts

- 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

Shared Spaces

Many groups have a shared space in /imaging or /group

- *ls /imaging*
- *ls /group*
  - if no one has recently accessed a group directory it won't be visible
    - remember automounting
  - 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 (or put in a ticket and cc them)

Special Characters

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 i/o 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]*
  - ; can be used to put multiple commands on a single line
    - cd /imaging; ls -d

Gotchas

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)
  - eg. an executable does not require .exe, a python script does not require .py, a jpeg does not require .jpg or .jpeg
  - 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

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
  - faster 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.

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, and use mkdir to manually create the directory.
    - Or you could script it up.
      - 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 got some participant data in many files, and you want to modify the data.
      - the directory CBUIT123456_MR123456 roughly mimics the format of our mri data
    - you could manually work on each of the 24 files
      - but what if there are 2000?

Why Bother?

Speed and Efficiency

- **Example (cont.)**
  - a script like Process.csh is a better solution.
    - it generates a list of files to process
    - then iteratively processes them

Have a go …

Try this:

- Have a look at some of the ‘data’ files in the directory – they all contain the same data
- then run Process.csh and see what happens.

Remember:

- Linux is case-sensitive
- a command must be in your $PATH or be fully qualified. ("." and "~" are handy for this)
- for many scripts you can also run the shell/interpreter with the file as an argument, eg
  - tcsh <filename>

A Few (More) 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 SLURM
  - copy the class files to your directory using ‘cp’ and a wildcard for the filenames
    - cp /imaging/training/SLURM/* SLURM
  - go into your new working directory
    - cd SLURM
      - or
    - cd ~/SLURM
    - ls

SLURM

- **SLURM -** Simple Linux Utility for Resource Management
  - ‘batch jobs’ are created on a login node and submitted to compute nodes
    - sbatch
    - srun
    - https://intranet.mrc-cbu.cam.ac.uk/home/cbu-cluster-using-the-cluster-2024/#intranet-sp-4
  - compute nodes are non-interactive! jobs are submitted, run, and report results in log files
  - the slurm commands take a wide variety of options to optimise your throughput. There are defaults which may or may not be suitable.

SLURM

- **A trivial example, WhereAmI.sh**

```
#!/bin/sh
#SBATCH -J=test1
#SBATCH -o test1.out
#SBATCH -t 00:20:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=8192
echo `hostname`
```

SLURM

- **A trivial example, explained**

  #!/bin/sh

  this line tells the script which command to execute, '/bin/sh'

  specifically, this is going to run a shell session using that command. If you want, you could use bash or tcsh, by specifying those. Remember, since the shell hasn't started, there is no PATH set, so you must use the fully qualified name.

  - /usr/bin/bash
  - /usr/bin/tcsh

SLURM

- **A trivial example, explained**

  in shell scripts, lines which begin with # are comments so the following lines are comments

  ```
  #SBATCH -J=test1
  #SBATCH -o test1.out
  #SBATCH -t 00:20:00
  #SBATCH --ntasks=1
  #SBATCH --cpus-per-task=1
  #SBATCH --mem-per-cpu=8192
  ```

  the sbatch command interprets those comments as arguments, providing an alternate way to customise jobs.

SLURM

- **A trivial example, explained**

  sbatch -J test1 -o test1.out -t 00:20:00 --ntasks=1 --cpus-per-task=1 --mem-per-cpu=8192 WhereAmI.sh

  Does the same as

  sbatch WhereAmI.sh

  with the arguments set in the file

  ```
  #SBATCH -J test1
  #SBATCH -o test1.out
  #SBATCH -t 00:20:00
  #SBATCH --ntasks=1
  #SBATCH --cpus-per-task=1
  #SBATCH --mem-per-cpu=8192
  ```

SLURM

- **A trivial example, explained**

  echo `hostname`

  This is the script itself, the command executed on the compute node.

  Try running it

  sbatch WhereAmI.sh

  and then look at the output

  more test1.out

SLURM

- **A less trivial example**

  This example is intended to demonstrate setting up a job in more realistic detail.

  The files are LessTrivial.sh and LessTrivial.py

  (LessTrivial.py is still pretty trivial code.)

SLURM

LessTrivial.sh

- #!/usr/bin/bash
  - bash is used rather than sh
- #SBATCH -o LT.%j.out
  - %j is interpreted by sbatch as the job number
  - a unique output file, so rerunning the code won't overwrite it
- the other options are the same as WhereAmI.sh

SLURM

LessTrivial.sh

- shell commands can be used
  - cd ~/SLURM
    - move to the working directory
- LessTrivial.py is python code, so we need a python environment
  - module load conda
  - conda activate mne1.0.3
- then the code can be run
  - because the environment has been set up, we don't need fully qualified path names
  - PATH has been modified by both module and conda

SLURM

LessTrivial.sh

- submit the job
  - sbatch LessTrivial.sh
- look at the ouput with more, less, or an editor

- If you move quickly enough, you could also try
  - scontrol show job <jobnumber>
  - using the jobnumber in your logfile
  - eg: log file is LT.3096927.out, then
    - scontrol show job 3096927

SLURM

```
JobId=1234567 JobName=LessTrivial
   UserId=zz99(9999) GroupId=staff(50) MCS_label=N/A
   Priority=1 Nice=0 Account=cbu QOS=normal
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:01 TimeLimit=00:20:00 TimeMin=N/A
   SubmitTime=2026-02-12T13:33:14 EligibleTime=2026-02-12T13:33:14
   AccrueTime=2026-02-12T13:33:14
   StartTime=2026-02-12T13:33:15 EndTime=2026-02-12T13:33:16 Deadline=N/A
   PreemptEligibleTime=2026-02-12T13:33:15 PreemptTime=None
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2026-02-12T13:33:15 Scheduler=Main
   Partition=Main AllocNode:Sid=login-k01.mrc-cbu.cam.ac.uk:1376656
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=node-j03
   BatchHost=node-j03
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   ReqTRES=cpu=1,mem=8G,node=1,billing=1
   AllocTRES=cpu=1,mem=8G,node=1,billing=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryCPU=8G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/zz99/SLURM/LessTrivial.sh
   WorkDir=/home/zz99/SLURM
   StdErr=/home/zz99/SLURM/LT.1234567.out
   StdIn=/dev/null
   StdOut=/home/zz99/SLURM/LT.1234567.out
   Power=
   TresPerTask=cpu:1
```

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)