Introduction to Linux and High Performance Computing

John Zaitseff
September 2018
Outline of this course

- Computer architecture: laptops/desktops, workstations, servers, cloud and HPC
- Available HPC facilities: getting an account, creating a project
- Connecting to a server, cloud and/or HPC system
- The Linux command line and the Bash shell
- Working with directories and files
- Redirecting standard input, output and error
- Creating, editing and running script files
- Submitting jobs to a HPC cluster, controlling jobs, querying job status

This is your course, so ask questions!
What is High Performance Computing?

“High performance computing (HPC) is the use of large-scale, off-site computers and parallel processing techniques for solving complex computational problems… HPC is typically used for solving advanced problems and performing research activities through computer modelling, simulation and analysis…”

— Intersect Australia
http://www.intersect.org.au/time/supercomputing

Image credit: Oak Ridge National Laboratory Leadership Computing Facility
Computer architecture: desktops, laptops...

Typical standard PC architecture:

• One processor (CPU)
• DRAM memory
• One graphics processor (GPU)
• Storage: hard drive(s), SSD(s)
• Keyboard
• Display screen: LCD
• Network: GbE
• Other peripherals, power supply, cooling
Computer architecture: workstations

Typical workstation architecture:
• One or two processors (CPU)
• DRAM memory (with ECC)
• One or more GPUs
• Storage: hard drives, SSDs
• Keyboard
• Display screen: LCD
• Network: GbE, 10GbE
• Other peripherals, power supply, cooling
Computer architecture: servers

Typical server architecture:

• One to four processors (CPU)
• DRAM memory (with ECC)
• One or more GPUs (optional)
• Storage: hard drives, SSDs
• Network: GbE, 10GbE
• Power supply, cooling
• Access is almost always via network ports using TCP/IP Internet protocols
Computer architecture: cloud servers

Typical cloud server architecture:

• Standard server architecture

• **Hypervisor software** creates the illusion of multiple individual (virtual) servers

• Virtual servers are usually independent, non-cooperating

• Allows for virtual server migration

• Excellent for interactive processes

• Not “bare metal”: run ~10-15% slower than physical hardware
Computer architecture: HPC

Massively Parallel Distributed Computational Clusters

- Many individual cooperating servers ("nodes"): dozens to tens of thousands
- Multiple processors per node: between 8 and 64 cores
- Interconnected by fast networks: 10Gb, 56Gb, 100Gb+
- Fast networks optimised for interprocess communications, often MPI (Message Passing Interface) using InfiniBand or Omni-Path using fat-tree networks
- Almost without exception run Linux, often CentOS 6.x or 7.x
Computer architecture: simple HPC

- Compute nodes 1, 2, …, n
- Internal network switch
- Storage nodes
- Head node
- Login node(s)
- Data Mover node(s)
- Disks
- SSH
Computer architecture: more complex HPC

- Compute nodes 1, 2, …, n
- Internal network switches (e.g., for MPI, storage)
- Storage nodes
- Disks
- Head node
- Login nodes
- Data Mover nodes
- Admin node(s)
- SSH
The Katana cluster: katana.unsw.edu.au

For staff and students at UNSW Sydney:

- 190 × Dell blade server nodes (various models)
  - Head/login node: katana
  - Compute nodes: kc01b01 to kc12b16

- 3036 × Intel Xeon processor cores (various models)
  - Two physical processors per node
  - Six to fourteen CPU cores per processor

- 27.75 TB of main memory (24–1024 GB per node)

- 900 TB of storage: 200 × 3–6 TB drives

- 10Gb and 1Gb Ethernet network interconnect

- Currently uses a “buy-in” scheme: ~$11k per node

https://www.hpc.science.unsw.edu.au/
The Raijin cluster: raijin.nci.org.au

For researchers across Australia (national facilities):

- 4452 × Fujitsu Primergy and Lenovo NeXtScale blade server nodes
- 84,656 × Intel Xeon Sandy Bridge 2.6GHz and Broadwell 2.6GHz processor cores
- 120 × NVIDIA Tesla K80 and 8 × Tesla P100 GPU coprocessors
- 32 × Intel Xeon Phi processors
- 309 TB of main memory
- 45 PB of storage and growing…
- 56Gb/100Gb Infiniband fat-tree network

Why learn Linux?

• To use High Performance Computing, you need to know how to use Linux

• Every single Top500 HPC system in the world uses Linux (see https://www.top500.org/). So does almost every other HPC system in the world—as well as cloud, workstations…

Why? “Linux is efficient, well-understood, battle-tested. It works and it’s free.”
— Steve R. Hastings, Why is Linux the preferred OS for supercomputers?

• Scalable: from mobile phones to the Sunway TaihuLight HPC system with 10,649,600 processor cores… and everything in-between

• Free Software / Open Source: full source code provided with permission to modify and redistribute (you can fix it yourself)

• Based on the principles of Unix: in use since 1969, encouraging minimalist, modular, extensible software development
“But Linux is hard!”

• Desktops/laptops with Linux do have nice graphical user interfaces (KDE, Gnome, …)

• HPC systems use the Linux command line

Why? Scriptable: the ability to automate tasks

The UNIX software development philosophy (Peter H. Salus, A Quarter-Century of Unix, 1994):

1. Write programs that do one thing and do it well.
2. Write programs to work together.
3. Write programs to handle text streams, because that is a universal interface.

Analogy: Linux provides you with the tools you need to build a house, skyscraper, shack…
Connecting to a HPC system

Use the **Secure Shell** protocol (SSH):

- Under Linux or Mac OS X:
  - Open a terminal and type: `ssh username@hostname`
    (for example, `ssh jlg777@raijin.nci.org.au`)

- Under Windows:
  - Use **PuTTY** (Start » All Programs » PuTTY » PuTTY), can be downloaded from [https://www.putty.org/](https://www.putty.org/) (or search for “PuTTY download”)
  - Start PuTTY, select Window » Appearance on left-hand side, change the font to Consolas, Regular, size 16
  - Can also use **MobaXterm** ([https://mobaxterm.mobatek.net/](https://mobaxterm.mobatek.net/)) but check licensing
  - Can also install **Cygwin**: “that Linux feeling on Windows” ([https://www.cygwin.com/](https://www.cygwin.com/))
Connecting to a HPC system

Try it now:

• Start PuTTY (Start » All Programs » PuTTY » PuTTY)
• Specify Host Name as raijin.nci.org.au
• Select Window » Appearance on left-hand side, click Change, change the font to Consolas, Regular, size 16, click OK
• Click Open
• At the “login as:” prompt, enter jxx777 (replace xx, using login provided at registration), press ENTER, then enter the password (nothing will be shown) and press ENTER
• You will get a command line prompt: something like jlg777@raijin1:~ $ 
• To exit, type exit and press ENTER
Typing in commands

• Use the keyboard to enter commands

• Commands consist of:
  – the *program name* (which command to run)
  – command line *arguments* (optionally in quotes)
    each of which must be separated by one or more *spaces*

• Commands and arguments are *case-sensitive*!

Examples:

```plaintext
ls /apps  — command “ls”, argument “/apps”
~jjz561/bin/cmdline arg1 arg2  — command “~jjz561/bin/cmdline”, 2 arguments
~jjz561/bin/cmdline arg1 arg2 "arg3 with spaces"  — command with 3 args
```
Command line options

• Many commands (programs) have optional command line options
• By convention, command line options appear as the first argument(s)
• Two forms of options: long options and short-form options
• Long options start with two hyphens, “--”, followed by a word
• Short-form options start with one hyphen, “-”, followed by one letter or digit
• By convention, short-form options can be combined, usually in any order: options in “ls -a -l -F” can be combined as “ls -alF” or “ls -laF” or...
• Most (but not all!) short-form options have a corresponding long option: “ls -a” is the same as “ls --all”, but “ls -l” is “ls --format=long”
• Some options have arguments, some of which may be optional: “tail -n 20 myfile” or “tail --lines=20 myfile”
• Many, many inconsistencies after almost 50 years of Unix history!
Getting help

How to remember all the command line options and parameters to commands? Don’t try!

• For a brief summary of command line options, try “command --help”
• For some (Bash shell built-in) commands, try “help command”
• For a full explanation, try “man command”
• For some commands, try “pinfo command”
• To quit the man or pinfo commands, press “q” (the Q key, no need to press ENTER)
• To search for a keyword in the Unix manual: “man -k keyword”
• Conventions: [] indicate optional arguments, italics indicate replaceable parameters
• Remember, “Google is your friend!” 😊
Some simple commands with help

Try it now:

```plaintext
cd ~jjz561/src/trader-7.12  # Change directory to ~jjz561/src/trader-7.12
ls                         # List the contents of the directory
cd src; ls                 # Multiple commands on one line, separated by “;”
pwd                        # Comments start with “#”, no need to type them in!

ls --help                   # Over five pages of summary information!
cd --help                   # Does this work?
help cd                     # But this does...
man ls                      # SPACE or PGDN to go to the next page, “q” to quit
pinfo coreutils            # Remember: “q” to quit

ls -a -l                    # “-a”: also list files starting with “.”; “-l”: list using a more detailed format
ls -al                      # Combining command line options…
ls --all -l                 # Mixing long and short-form options
```

20
Directories and files: *paths and pathnames*

- Files and directories are organised into a single hierarchical *tree* structure.
- The top of the tree is called the *root* directory (*root*), and is denoted as `/` (slash).
- Directories are containers (or folders) for files and directories.

**Example:** (partial tree only)
Absolute pathnames

• Any file or directory can be uniquely represented as an absolute pathname:
  – gives the full name of the file or directory
  – starts with the root “/” and lists each directory along the way
  – has a “/” to separate each path (or pathname) component

Example:

Directory /apps/matlab/R2018a
Relative pathnames

• When a program (command) is running, it is called a process
• Every process has a current working directory or current directory (“the directory I am currently in”)
• When you log in, the system sets your current working directory to your home directory, something like /home/z9693022 or /home/561/jjz561 (highly system dependent)
• Any process can change its current working directory (“cd directory”) at any time
• A relative pathname points to a path relative to the current directory
  – does not start with “/”
  – path components are still separated with slashes “/”
• Current directory is denoted by “.” (dot)
• The directory above the current one (parent directory) is denoted by “..” (dot-dot)
• Relative pathnames often just contain a filename with no directories (i.e., no slashes “/”)
Examples of relative pathnames

- Assume current directory is `/home/561/jjz561/src/trader-7.12`:
  
  - README → `/home/561/jjz561/src/trader-7.12/README`
  - src/trader.c → `/home/561/jjz561/src/trader-7.12/src/trader.c`
  - src/../../README → `/home/561/jjz561/src/trader-7.12/README`
  - ./README → `/home/561/jjz561/src/trader-7.12/README`
Important directories

• Home directory (system dependent): on Raijin, something like /home/561/jjz561

• Binary directories for utility programs:
  - /bin — for essential utilities
  - /usr/bin — for other utilities and some applications
  - /opt/bin or /usr/local/bin — for local utilities and applications
  - /home/num/user/bin — for your own utilities

• On Raijin, scratch directory for temporary files: /short/proj/user

• On Raijin, applications: /apps

• On Raijin, module files: /apps/Modules/modulefiles

Note synonyms: path, pathname, filename
More with pathnames

• To change directories: “cd  dir”
• To change to your home directory: “cd ~” or “cd” (by itself)
• To get current working directory: “pwd”
• To list files in a directory: “ls”
• In full, using Unix conventions: “ls [options] [pathname ...]”
• Some options for ls:
  – “-a” for all files, including those starting with “.”
  – “-l” (lowercase letter L) for long (detailed) listing
• To show the directory tree structure: “tree”, “tree -d” (show directories only)
• To view a file page by page: “less filename”, “q” to quit, “h” for help
Playing with pathnames

Try it now:

```
cd ~jjz561/src/trader-7.12  # Change directory to ~jjz561/src/trader-7.12
pwd  # Should show "/home/561/jjz561/src/trader-7.12"
ls  # List the contents of the directory
ls -al  # List the contents of the directory (all files, long format)
tree -d .  # Show the directory tree structure starting from ".
```

```
ls -l README  # Look at the listing details for README
ls -l src/README  # Is it the same as src/README?
cd src  # Now change to src subdirectory
pwd  # Should show "/home/561/jjz561/src/trader-7.12/src"
ls -l README  # Are the details the same as the previous "ls -l" line?
ls -l ../README  # And which README are we referring to now?
cd ..  # Now change to the parent directory
pwd  # Should show "/home/561/jjz561/src/trader-7.12" again
```
The Bourne Again (Bash) shell

- Official manual page entry ("man bash"): Bash is an sh-compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the Korn and C shells (ksh and csh).

  Bash is intended to be a conformant implementation of the Shell and Utilities portion of the IEEE POSIX specification (IEEE Standard 1003.1). Bash can be configured to be POSIX-conformant by default.

- Interprets your typed commands and executes them
- Just another Linux program: nothing special about it!
- By default, started by the system when you log in
- You can then start another shell, if you like (e.g., ksh, tcsh, even python)
- You can start a subshell by running "bash"
- To exit a subshell (or the main shell): "exit"
Some features of Bash

• Powerful command line facilities (shortcuts) to make life easier for you:
  – Tab completion (press the TAB key to complete commands and pathnames, TAB TAB to list all possibilities)
  – Command line editing: try ↑ (Up-Arrow) to recall previous commands, CTRL-R (C-R or ^R) to search for previous commands, ← and → to move along current command line

• A full programming and scripting language:
  – Variables and arrays
  – Loops (for; while; until), control statements (if … then … else; case)
  – Functions and coprocesses
  – Text processing (“expansion” and “parameter substitution”)
  – Simple arithmetic calculations
  – Input/output redirection (e.g., redirect output to different files)
  – Much, much more! (The man page runs to over 5,900 lines)
File and directory patterns

- The Bash shell *interprets* certain characters in the command line by replacing them with matching pathnames.
- Called *pathname expansion*, *pattern matching*, *wildcards* or *globbing*.
- This globbing is a feature of the Bash shell, *not* the operating system itself.
- At the start of a filename: “~” is replaced with your home directory, “~user” is replaced with the home directory of user user.
- For existing pathnames: “*” matches any string, “?” matches any single character, “[abc]” matches any one of the enclosed characters (in this case, “a”, “b” or “c”).
- Glob patterns “*”, “?” and “[...]” only match existing pathnames.
- Even for pathnames that do not exist: “{alt1,alt2,...}” lists alternatives, “{n..m}” lists all numbers between n and m, “{n..m..s}” from n to m in steps of s.
  - Technically called *brace expansion*.
Playing with pathname expansion

Try it now:

```
cd ~jjz561/src/trader-7.12/src
alias z=~jjz561/bin/cmdline  # Make a temporary shortcut “z” to the cmdline script
z arg1 arg2  # Show how arguments arg1 and arg2 are passed to programs
z arg1 "arg2 with space"  # Bash handles the quoting characters, too
z ~  # Show how Bash expands “~”
z ~jjz561  # … and for user jjz561’s home directory
z *c  # Show how Bash expands “*c”: all filenames ending in “c”
z ?????.c  # … all filenames six characters long (4 + “.c”) ending in “.c”
z M*m  # … all filenames starting with “M” and ending with “m”
z [it]*  # … all filenames starting with either “i” or “t”
z ../lib/uni*  # … all filenames in ../lib starting with “uni”
z ../*//*.c  # What does this do?
```
Playing with brace expansion

Try it now:

```bash
cd ~/jjz561/src/trader-7.12/src
alias z=~/jjz561/bin/cmdline
```

Make a temporary shortcut “z” to the `cmdline` script

```bash
ls test-*
# “No such file or directory”
z test-*
# What is passed as argument 1?
z test-{one,two,three}
# What three arguments does Bash expand this to?
z somedir/{one,two,three}
# … and this?
```

```bash
z test-{1..100}
# Expand to “test-1”, “test-2”, …, “test-100”
z test-{001..100}
# … with zero-padding
z test-{1..100..3}
# … by steps of three
z test-{100..1..-3}
# … by steps of negative three
```
Naming files and directories

• Linux allows any characters in filenames except “/” and the NUL byte
• You may create filenames with “weird” characters in them:
  – spaces and tabs
  – starting with “-”: conflicts with command line options
  – question marks “?”, asterisks “*”, brackets and braces
  – other characters with special meanings: “!”, “$”, “&”, “#”, “"”, etc.
• Just because you can does not mean you should!
• To match such files: use the glob characters “*” and “?”
• Linux file systems are case-sensitive: README.TXT is different from readme.txt, which is different from Readme.txt and ReadMe.txt!
• File type suffixes (e.g., “.txt”) are optional but recommended
• Filenames starting with “.” are usually hidden from globs and ls output

Recommendation: Use “a” to “z”, “A” to “Z”, “0” to “9”, “-”, “_” and “.” only.
Managing directories

- To create a directory: “`mkdir dir ...`”
- To create intermediate directories as well: “`mkdir -p dir ...`”
- To remove an empty directory: “`rmdir dir ...`”

Try it now:

```
cd; ls  # Change to your home directory and list its contents (should be empty)
mkdir test1  # Create the directory test1
cd test1  # ... and change to it
mkdir sub{1,2,3}  # What does this do?
mkdir ../test2  # Where is the directory test2 created?
cd ../test2  # Change to it
mkdir sub{04..10}  # How to make lots of subdirectories in one go!
cd ~  # Go back to the home directory
tree -d  # What does the directory tree structure look like?
```
Managing files

- To output one or more file’s contents: “cat filename ...”
- To view one or more files page by page: “less filename ...”
- To copy one file: “cp source destination”
- To copy one or more files to a directory: “cp filename ... dir”
- To preserve the “last modified” time-stamp: “cp -p”
- To copy recursively: “cp -pr source destination”
- To move one or more files to a different directory: “mv filename ... dir”
- To rename a file or directory: “mv oldname newname”
- To remove files: “rm filename ...”

**Recommendation:** use “ls filename ...” before rm or mv: what happens if you accidentally type “rm *”? or “rm * .c”? (note the space!)
Managing files and directories

- To copy whole directory trees: “cp -pr filename ... destination”
- To copy to and from another Linux or Mac OS X system (e.g., from Raijin to Katana), use Secure Copy: `scp [-p -r] source ... destination`
  - Either source or destination (but not both) can contain a remote system identifier followed by a colon: “`[user@]hostname:`”
  - Powerful command but tricky! Note the trailing “/” on the directory arguments

Examples: (remember, don’t type in the examples!)
```
cp -pr ~jjz561/src/trader-7.12 .
scp -p ~/file1.txt jjz561@rajin.nci.org.au:file2.txt
scp -p john@zap.org.au:src/README .
rsync -vauSH --delete ~/src/ jjz561@rajin.nci.org.au:~/src-unsw/
```
Playing with pathname expansion

Try it now:

```bash
cd ~; mkdir src; cd src

cp -pr ~jjz561/src/trader-7.12 .
# Copy directories recursively to “.” (current directory)
cd trader-7.12
# Change to the newly copied directory
cat build-aux/bootstrap
# Display the contents of this file
ls */*.c
# List all files matching “*/*.c”
rm */*.c
# … and then remove them!
ls */*.c
# What happens now?

mv README my-new-filename
# Rename the README file
cp INSTALL new
# Make a copy of INSTALL and call it “new”
ls -l INSTALL new
# What is the difference between the listings?
cp -p INSTALL same
# Copy INSTALL, preserving time-stamps
ls -l INSTALL same
# Verify the two files have the same date and time
```
Transferring files to the outside world

• To copy files to another Linux or Mac OS X system: use “scp” or “rsync”
  – same as within a HPC/Linux system

• To copy files to and from a Windows machine: use WinSCP (Start » All Programs » WinSCP » WinSCP), FileZilla, or “scp” or “rsync” under Cygwin
  – WinSCP may be downloaded from [https://winscp.net/eng/index.php](https://winscp.net/eng/index.php)
  – FileZilla may be downloaded from [https://filezilla-project.org/](https://filezilla-project.org/)
  – both of these programs use a “drag-and-drop” graphical interface
  – the MobaXterm client ([https://mobaxterm.mobatek.net/](https://mobaxterm.mobatek.net/)) has a built-in Secure Copy interface as well
More Linux commands

• What machine am I on? “hostname”
• What is the date and time? “date”
• Who is logged in? “who”
• But who is user jjz561? “finger [username ...]”
• What is the user name for someone? “finger part-of-name”
• What files contains a particular string? “grep 'pattern' filename ...”
• What is the difference between two files? “diff [-u] file1 file2”
• How do I rename multiple files at once? “rename” or “prename”
• Where is a file named filename? “find dir ... -name filename”
• How big is a file or directory? “du -h [filename ...]”
• How much space is available in a directory? “df -h [dir ...]”
• How much disk quota do I have? “lquota” or (on other systems) “quota” or “quota -s”
  – On Raijin: quota for your home directory is 2.0 GB
Everything is a file

• Every process (running program) can read from or write to any file
  – process must have appropriate read or write permissions!
  – data files, configuration files, pathnames passed on the command line, …
• Three files are automatically opened for each process:
  – standard input (stdin)
  – standard output (stdout)
  – standard error (stderr)

In Unix, everything is a file!

• Keyboard and screen are represented by the file /dev/tty; use CTRL-D to signify the end of input
• Some other special files: /dev/null (an empty file), /dev/zero (an infinite number of binary zeros—will use up your disk quota in a hurry!)
Redirecting input and output

- Standard input, standard output and standard error can be redirected to/from a file or even piped to another program.
- To redirect output to `filename`, use “>filename”
- To append output to `filename`, use “>>filename”
- To redirect input from `filename`, use “<filename”
- To connect the output from one program to the input of another (a pipe), use “program1 | program2”
- To redirect output to `filename` and the screen, use “| tee filename”
- Multiple pipes are allowed: “program1 | program2 | ... | programn”
- Output of a process can be substituted into a command line: “$(commandline)”
- Many Unix programs are designed to be used in this way, as filters
Playing with file redirection

Try it now:

```bash
cd ~jjz561/src/trader-7.12

ls > ~/dir-list1  # Redirect the output of ls to ~/dir-list1
cat ~/dir-list1   # Show what is in that file
ls src >> ~/dir-list1  # Append the output of “ls src” to ~/dir-list1
cat ~/dir-list1   # What does the file contain now?
wc -l < ~/dir-list1 # Run “wc -l” (count lines in a file), but use ~/dir-list1 instead
#   of /dev/tty (the keyboard), the default stdin file

cat ~/dir-list1 | wc -l  # Use a pipe from cat to wc (output of cat becomes input of wc)

ls -l /bin | grep Oct  # How many files were last modified in October?
ls -l /bin | grep Oct | sort -nk5  # … sorted by the file size (5th field)
```
Simple scripting

- Shell scripts are just files containing a list of commands to be executed
- First line (“magic identifier”) must be “#!/bin/bash”
- Comments are introduced with “#”
- The script file must be made executable: “chmod a+x filename”

Variables:

- To set a variable, use “varname=value” (no spaces!)
- To use a variable, use “$varname” or “${varname}”
- Variable names start with a letter, may contain letters, numbers and “_”
- Variable names are case-sensitive (as with most things Unix)
Simple scripting, continued

For loops:

```bash
for varname in list ...; do
  process using ${varname}
done
```

**Control statements** (multiple “elif” allowed; “elif” and “else” clauses are optional):

```bash
if [ comparison ]; then # Use literal “[” and “]” characters
  if-true statements
elif [ second-comparison ]; then
  if-second-true statements
else
  if-false statements
fi
```
Simple scripting, continued

While loops:

while [ comparison ]; do
  while-true statements
done

Until loops:

until [ comparison ]; do
  while-false statements
done

Examples of comparisons:

• string1 = string2 — strings string1 and string2 are equal
• number1 -lt number2 — number1 is less than number2
• file1 -nt file2 — file1 (e.g., a data file) is newer than file2 (e.g., output file)
  
  – See the manual page for test ("man test") for more information
Simple scripting, continued

Functions:

```bash
funcname () {
    body of function, parameters are accessed using $1, $2, …
}
```

– Called using “`funcname arg1 arg2 …`” within the script

• Many, many other programming features available!
• Read the reference and manual pages: “`pinfo bash`”; “`man bash`”
• Some books:
Editing files under Linux

• Use an *editor* to edit text files

• Many choices, leading to “religious wars”!

• Some options: GNU Emacs, Vim, Nano

• Nano is very simple to use: “nano filename”
  – CTRL-X to exit (you will be asked to save any changes on the bottom of the screen)

• GNU Emacs and Vim are highly customisable and programmable
  – For example, see the file `~jjz561/.emacs` on Raijin
Creating your first script

Try it now:

```
mkdir ~/ex1; cd ~/ex1  # Create the ~/ex1 directory and change into it
nano ./script1         # Start the Nano text editor with the file script1
```

Enter the following text:

```
#!/bin/bash
echo "I am user $(whoami), running on $(hostname)"
echo "Dates and times:"
date          # Print the date and time
sleep 30      # Do nothing for 30 seconds
date          # Do it again
```

Press CTRL-X to save the file and exit the editor (follow the prompts on the bottom of the screen), then:

```
chmod a+x ./script1  # Make script1 executable
./script1            # Execute the script! (Note the use of "./")
```
A script with loops

Try it now:

```
mkdir ~/ex2; cd ~/ex2  # Create and change to ~/ex2
cp -p ~jjz561/doc/hpc-tutorial/examples/make-matlab-scripts .  # Don't forget the trailing "!"
less ./make-matlab-scripts  # Examine the make-matlab-scripts script
./make-matlab-scripts  # Run the make-matlab-scripts script
```

Answer the following questions:

1. What does the `make-matlab-scripts` do?
2. How does it do it?
3. What files are generated by the script? Hint: use the `ls` command
4. What type of files are they? (Data files, programs, input files, …)
Applications on the cluster

• Applications are managed using the *module system*
• On Raijin, applications are stored in `/apps`
• On Raijin, module files are stored in `/apps/Modules/modulefiles`
• Module files set shell environment variables such as `PATH`
• `PATH` controls where applications are searched (the `search path`)
• To see available applications: “`module avail [application]`”
• To see currently loaded applications: “`module list`”
• To load an application: “`module load application[/version]`”
• To unload an application: “`module unload application[/version]`”
Seeing the applications

Try it now:

module avail  # What applications are available?
module list  # What applications are currently loaded?

```
echo $PATH
```
# See the current value of the PATH variable
```
module load matlab/R2018a
```
# Set the PATH to include Matlab R2018a
```
echo $PATH
```
# What does PATH look like now?
```
module unload matlab/R2018a
```
# We don’t want to use Matlab R2018a any more…
```
echo $PATH
```
# PATH no longer contains the Matlab directory
HPC architecture revisited

- Compute nodes 1, 2, …, \( n \)
- Internal network switch
- Storage nodes
- Disks
- Head node
- Login node(s)
- Data Mover node(s)
- SSH

We've been running jobs (scripts, programs) on a login node: a bad idea!
Submitting jobs to the cluster

• To submit a job to the cluster compute nodes:
  – Create a shell script file as per normal
  – Add #PBS directives as required directly after “#!/bin/bash”
    (These look like shell comments, but are interpreted by the PBS scheduler)
  – Add “cd $PBS_O_WORKDIR” after the #PBS directives, or use “#PBS -l wd”
  – Execute “qsub ./scriptfile”
  – Wait for the job to run, checking its status as required

• Warning: If you have not submitted a job using qsub (or equivalents such as sbatch on other systems), you are almost certainly running your job on a login node!

• Running jobs on login nodes bypasses the power of the HPC cluster
Common PBS directives

- Some common #PBS directives on Raijin (see https://opus.nci.org.au/ and “man qsub” for full details); many options have reasonable defaults:

  - #PBS -N scriptname — Set a name for the script
  - #PBS -P project — Charge resources from this project
  - #PBS -q queuename — Which queue to submit to
  - #PBS -l ncpus=n — Request n processor cores in total
  - #PBS -l ngpus=n — Request n GPUs
  - #PBS -l walltime=hh:mm:ss — How much time is required for running the job
  - #PBS -l mem=sizeMB — How much memory is required (in MB)
  - #PBS -l software=licname — Use software licence licname
  - #PBS -M email — Send notifications to the email address
  - #PBS -m abe — What notifications to send by email
  - #PBS -l wd — Run from the same directory as submission
Checking your job status

• Submit your jobs using “qsub”
  – You will be given a job identifier: save this somewhere

• Check job and queue status: “qstat [jobid]”

• Check your project’s usage this quarter: “nci_account [-P project] [-v] [-vv]”

• Many systems have an overall system status page (e.g., using Ganglia)
  – On Raijin, the live status page is http://nci.org.au/user-support/current-job-details/
  – Can also view software licences currently being used: http://nci.org.au/user-support/getting-help/license-status/

Try it now: view the Raijin live status page and the Raijin software licence status page
Managing your jobs

- To see jobs belonging to you: “qstat -u $USER”
- To delete a queued job (whether running or not): “qdel jobid ...”
- To modify the resources of a job in the queue: “qalter options jobid ...”
- To move the job to another queue: “qmove newqueue jobid ...”
- To place a job on hold: “qhold jobid ...”
- To release a job currently on hold: “qrls jobid ...”
- To rerun a job (kill it and then restart it): “qrerun jobid ...”
- To show the current standard output (stdout) of a job: “qcat jobid ...”
Your first HPC job!

Try it now:

```
mkdir ~/ex3; cd ~/ex3             # Create and change to ~/ex3
cp ../ex1/script1 job1          # Copy script1 into job1
nano ./job1                      # Start the Nano text editor with the file job1
```

Enter the following text directly after the "#!/bin/bash" line:

```
#PBS -q express
#PBS -M yourEmailAddress
#PBS -m abe
#PBS -l walltime=00:05:00
#PBS -l mem=1GB
#PBS -l ncpus=1
#PBS -l wd
```

Press CTRL-X to save the file and exit the editor (follow the prompts on the bottom of the screen), then:

```
qsub ./job1                      # Submit the job to the cluster
qstat -u $USER                  # Check the queue status (you may need to run this more than once)
```
Did my job finish successfully?

- If your job script contains the “#PBS -M email” directive, you will receive an email once your job starts and a second email once it finishes.
- Check `Exit_status` in the second email: it should be zero for a successful job.

**Example completion email:**

```plaintext
PBS Job Id: 549525.r-man2
Job Name: job1
Execution terminated
Exit_status=0
resources_used.cpupercent=0
resources_used.cput=00:00:00
resources_used.mem=0kb
resources_used.ncpus=1
resources_used.vmem=5048kb
resources_used.walltime=00:00:31
```

— Successful job!
— 31 seconds out of 5 mins requested
Where did my output go?

- PBS automatically redirect standard input, standard output and standard error:
  - standard input from /dev/null
  - standard output to script.ojobid
  - standard error to script.ejobid (should be empty for successful runs)

Try it now:
```bash
cd ~/ex3; ls          # What files are present?
less job1.e*          # View the error output (should be empty); remember: “q” to quit less
less job1.o*          # View the standard output
```

Answer the following questions:
1. What difference is there between the output of job1 and ../ex1/script1? Hint: “running on …”
2. What else appears in the standard output file?
3. How could you use this information for future runs of this job?
Running interactive jobs

• **Remember:** Running jobs on login nodes bypasses the power of the HPC cluster
• But running interactively is useful for debugging!
• Solution: Start an *interactive job*
  – Replace the script name with “-I”
  – For programs with a graphical user interface, use “-I -X” if you have an X11 server
  – Specify all **#PBS** directives as command line options to “qsub”:

  ```bash
  #PBS -P project → “qsub ... -P project ...”
  #PBS -q queuename → “qsub ... -q queuename ...”
  #PBS -l walltime=hh:mm:ss → “qsub ... -l walltime=hh:mm:ss ...”
  #PBS -l mem=sizeMB → “qsub ... -l mem=sizeMB ...”
  ...
  ```
Running interactively

Try it now:

```
  cd ~/ex1
  hostname        # Where am I running? raijin1–rajin6 are login nodes
  qsub -q express -l walltime=0:10:00 -l mem=4GB -l ncpus=1 -l wd -I
  # Request an interactive job (you may need to wait)
```

Once a command line prompt appears:

```
  hostname        # Where am I running now? rNNN is a compute node
  ./script1       # Run ./script1, but now on a compute node
  exit            # Finish the interactive job and return to the login node
```
The queues on Raijin

<table>
<thead>
<tr>
<th>Queue</th>
<th>CPUs + GPUs</th>
<th>Memory</th>
<th>Charge Rate</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>2×8</td>
<td>32/64/128 GB</td>
<td>1.0</td>
<td>Intel Xeon Sandy Bridge nodes</td>
</tr>
<tr>
<td>express</td>
<td>2×8</td>
<td>32/64/128 GB</td>
<td>3.0</td>
<td>Sandy Bridge; high-priority queue</td>
</tr>
<tr>
<td>normalbw</td>
<td>2×14</td>
<td>128/256 GB</td>
<td>1.25</td>
<td>Intel Xeon Broadwell nodes</td>
</tr>
<tr>
<td>expressbw</td>
<td>2×14</td>
<td>128/256 GB</td>
<td>3.75</td>
<td>Broadwell; high-priority queue</td>
</tr>
<tr>
<td>gpu</td>
<td>2×12 + 4 × K80</td>
<td>256 GB</td>
<td>3.0</td>
<td>Nvidia Tesla K80; must use multiples of 6 cores</td>
</tr>
<tr>
<td>gpupascal</td>
<td>2×12 + 4 × P100</td>
<td>128 GB</td>
<td>4.0</td>
<td>Nvidia Tesla P100; must use multiples of 6 cores</td>
</tr>
<tr>
<td>knl</td>
<td>1×64</td>
<td>192 GB</td>
<td>0.25</td>
<td>Intel Xeon Phi 7230; must use all 64 cores</td>
</tr>
<tr>
<td>hugemem</td>
<td>2×14</td>
<td>1024 GB</td>
<td>1.25</td>
<td>Broadwell; must use multiples of 7 cores</td>
</tr>
<tr>
<td>megamem</td>
<td>4×8</td>
<td>3072 GB</td>
<td>1.25</td>
<td>Broadwell; must use multiples of 32 cores</td>
</tr>
<tr>
<td>copyq</td>
<td>2×8</td>
<td>32 GB</td>
<td>1.0</td>
<td>For file copying; must use one core only</td>
</tr>
</tbody>
</table>
Where to from here?

• Manage your resources wisely: use “nci_account [-P project] [-v] [-vv]”

Try it now:

nci_account
nci_account -P w47

# Usage for my (default) project
# A rather large project, well used!

• Read the documentation for your HPC system. On Raijin:
Conclusion

You have begun your journey to using Linux and High Performance Computing effectively. Well done!

John Zaitseff <J.Zaitseff@unsw.edu.au>

Please fill out the following two-minute survey:

https://goo.gl/forms/vdZl1XlHfXXebuFy1

Keep in contact:

https://research.unsw.edu.au/research-technology-services
restech@unsw.edu.au