UNSW SYDNEY | Australia's Global University

Research Technology Services, DVC Research Infrastructure

Introduction to Linux and High Performance Computing

John Zaitseff
March 2024

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*

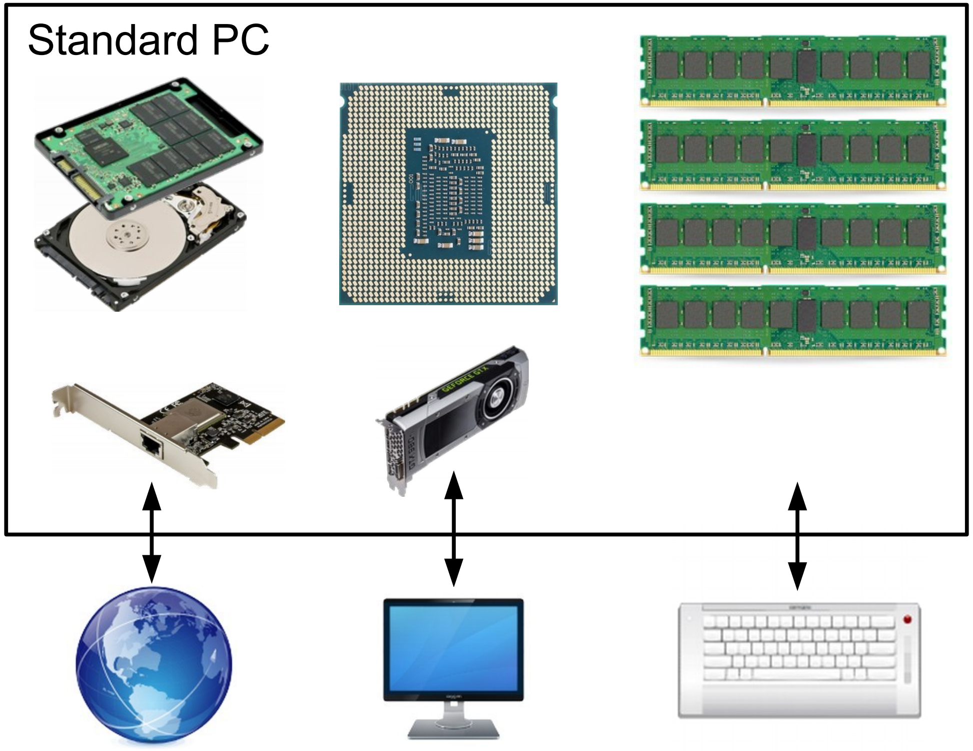
http://www.intersect.org.au/time/supercomputing

Image credit: Oak Ridge National Laboratory Leadership Computing Facility

Computer architecture: desktops, laptops…

Typical standard PC architecture:

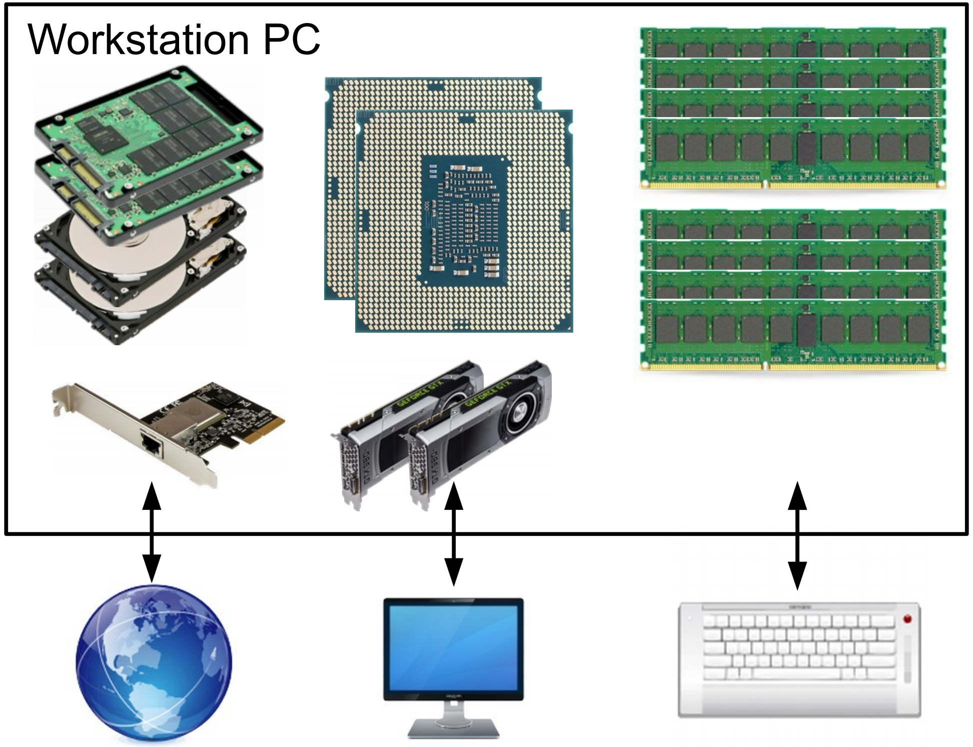
- One processor (CPU)
- DRAM memory
- One graphics processor (GPU)
- Storage: hard drive(s), SSD(s)
- Keyboard
- Display screen: LCD
- Network: GbE
- Other peripherals, power supply, cooling



Computer architecture: workstations

Typical workstation architecture:

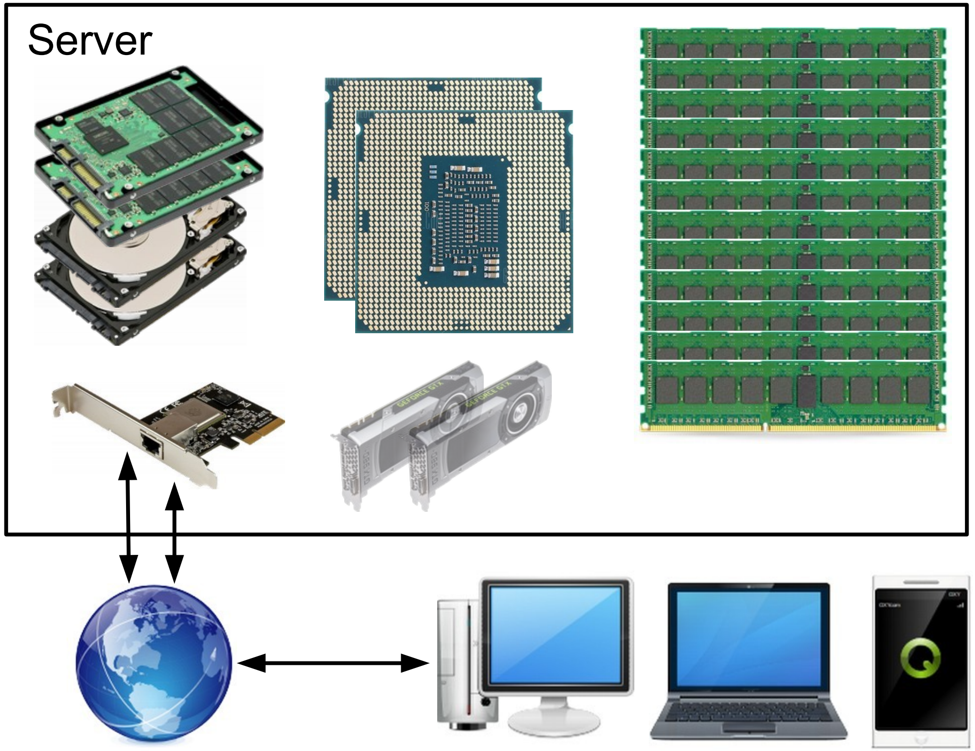
- One or two processors (CPU)
- DRAM memory (with ECC)
- One or more GPUs
- Storage: hard drives, SSDs
- Keyboard
- Display screen: LCD
- Network: GbE, 10GbE
- Other peripherals, power supply, cooling



Computer architecture: servers

Typical server architecture:

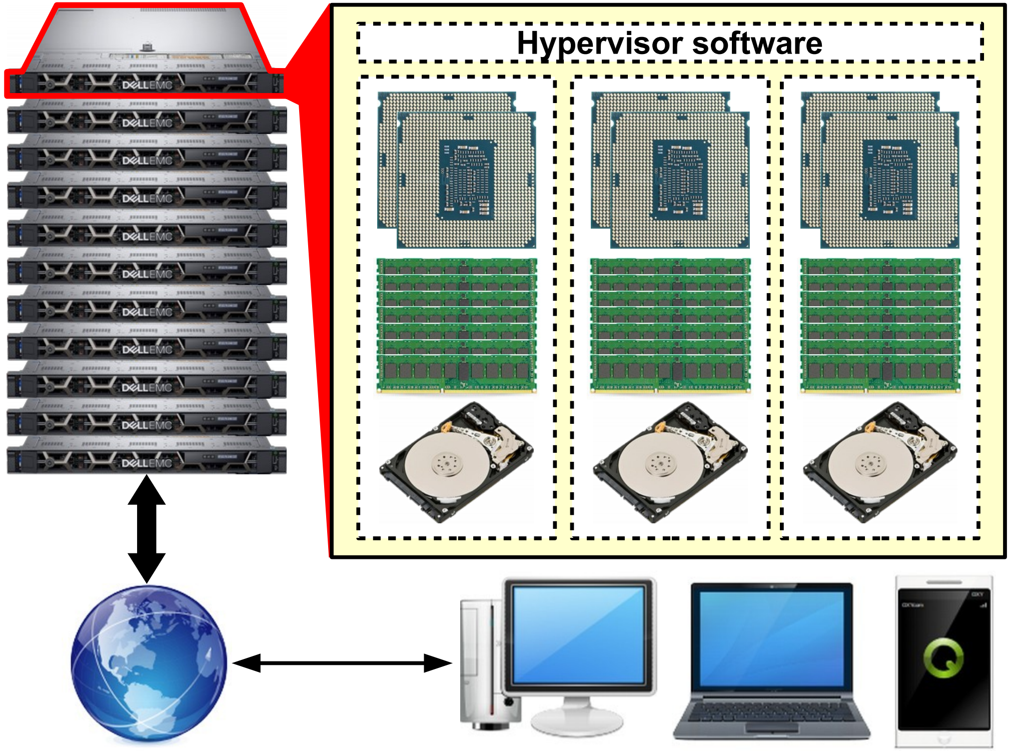
- One to four processors (CPU)
- DRAM memory (with ECC)
- One or more GPUs (optional)
- Storage: hard drives, SSDs
- Network: GbE, 10GbE
- Power supply, cooling
- Access is almost always via network ports using TCP/IP Internet protocols



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



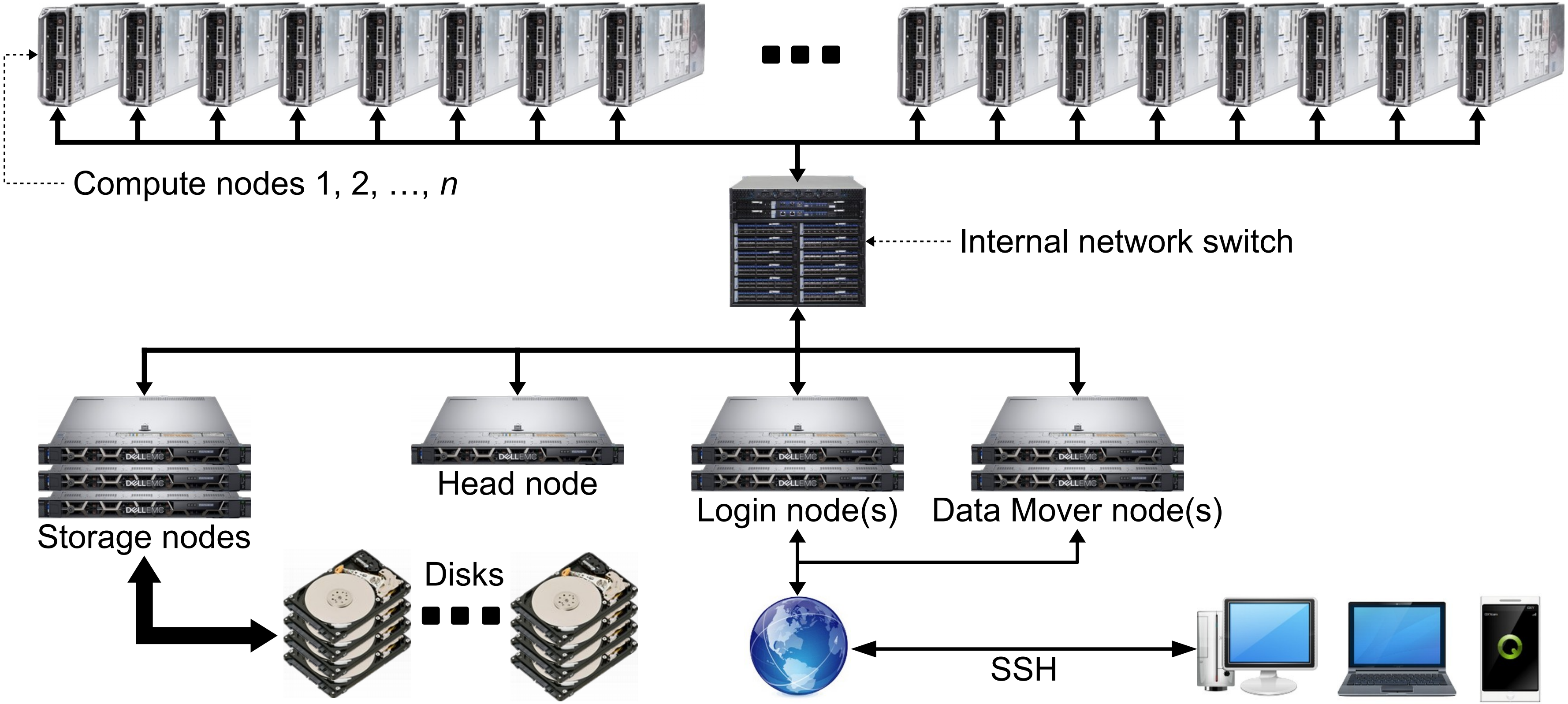
Computer architecture: HPC

Massively Parallel Distributed Computational Clusters

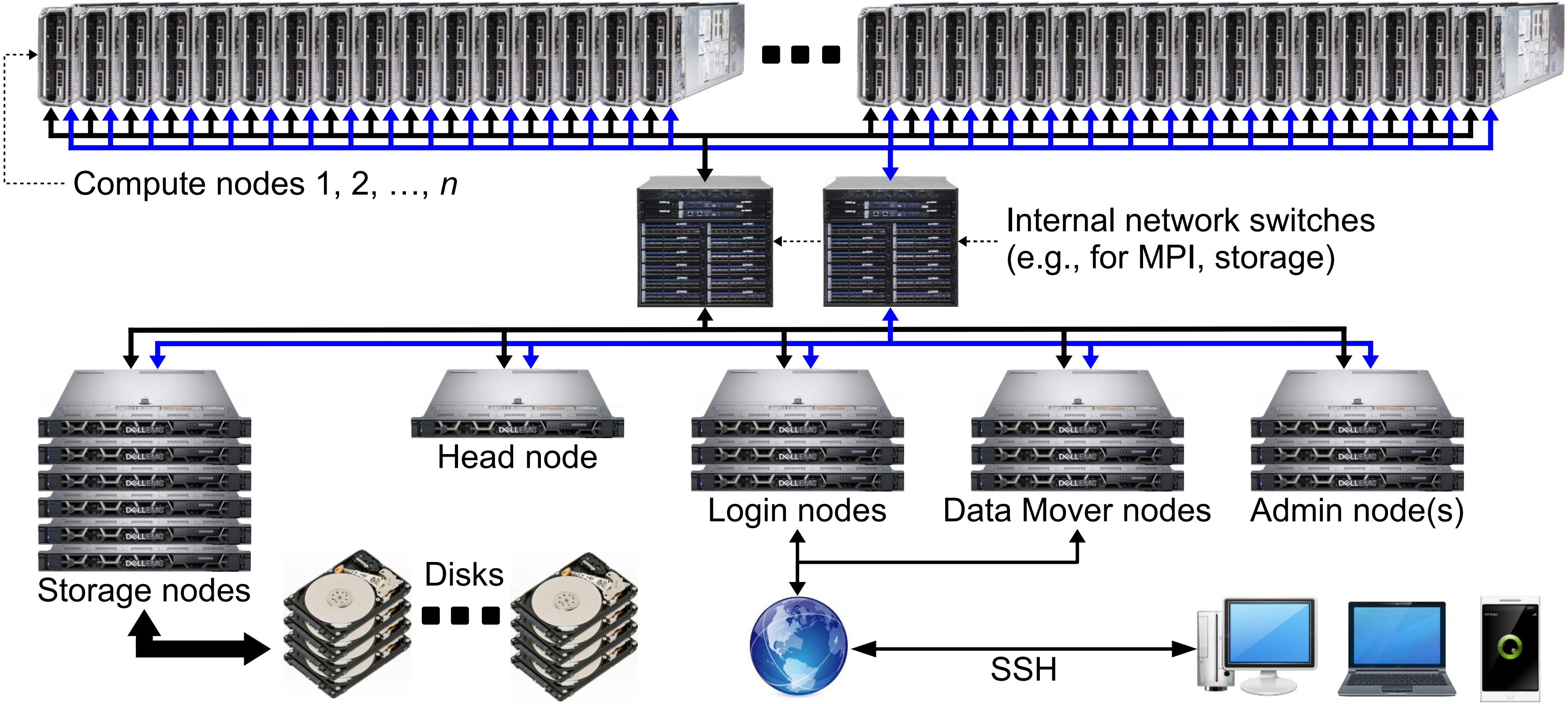
- Many individual cooperating servers (“nodes”): dozens to tens of thousands
- Multiple processors per node: between 8 and 64 cores
- Interconnected by fast networks: 10Gb, 56Gb, 100Gb+
- Fast networks optimised for interprocess communications, often MPI (Message Passing Interface) using InfiniBand using fat-tree or similar networks
- Almost without exception run Linux, often CentOS 7 or later

The old Trentino cluster
Image credit: John Zaitseff, UNSW

Computer architecture: simple HPC



Computer architecture: more complex HPC


Compute nodes 1, 2, …, *n*
Internal network switches (e.g., for MPI, storage)
Head node
Login nodes
Data Mover nodes
Admin node(s)
Storage nodes
Disks
SSH

The Katana cluster: *katana.unsw.edu.au*

For staff and students at UNSW Sydney:

- 168 × Dell, Lenovo and Huawei server nodes (various models)
  - Head/login nodes: *katana* (*katana1*, *katana2* and *katana3*)
  - Compute nodes: *k001* to *k255* (not all nodes present)
- 7060 × Intel Xeon processor cores (various models)
  - Mostly two physical processors per node
  - 16–80 × CPU cores per physical processor
- 54.5 TB of main memory (128–1536 GB per node)
- Over 3 PB of storage (and growing)
- 10Gb Ethernet + 100Gb Infiniband network interconnect
- Currently uses a “buy-in” scheme: ~$20k per node
- Ideal for beginner and intermediate HPC users

https://research.unsw.edu.au/katana

The old Leonardi cluster (similar to Katana)
Image credit: John Zaitseff, UNSW

The Gadi cluster: *gadi.nci.org.au*

For researchers across Australia (national facilities):

- 4997 × compute server nodes
- 260,760 × Intel Xeon Cascade Lake and some older Skylake and Broadwell processor cores
- 50 × compute nodes with 1536 GB of memory
- 7 × compute nodes with 3072 GB of memory
- 692 × NVIDIA Tesla V100 GPU coprocessors
- Over 1275.9 TB of main memory
- Over 68 PB of storage
- 200Gb Infiniband network in Dragonfly+ topology
- High-speed DDN Lustre parallel file system
- Ideal for intermediate and advanced HPC users

https://nci.org.au/our-systems/hpc-systems

Part of the Gadi cluster in Canberra, ACT
Image credit: National Computational Infrastructure

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 Frontier HPC system in the United States with 8,699,904 processor cores (1194 PFlop/s, 22.7 MW)… 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

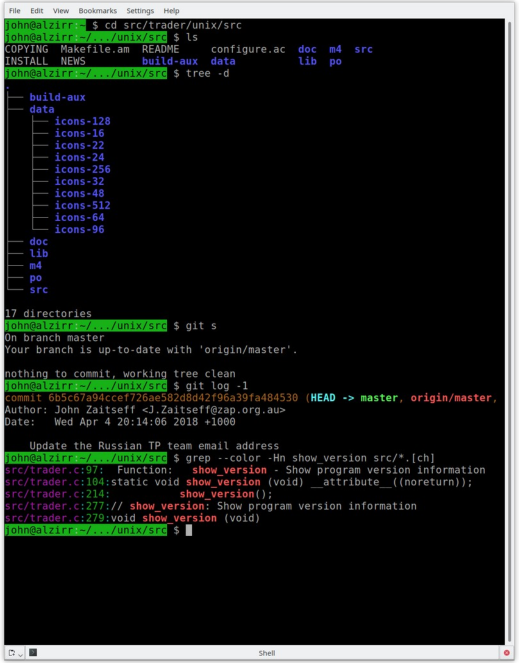
“But Linux is hard!”

- Desktops/laptops with Linux *do* have nice graphical user interfaces (KDE, Gnome, …)
- HPC systems normally 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.



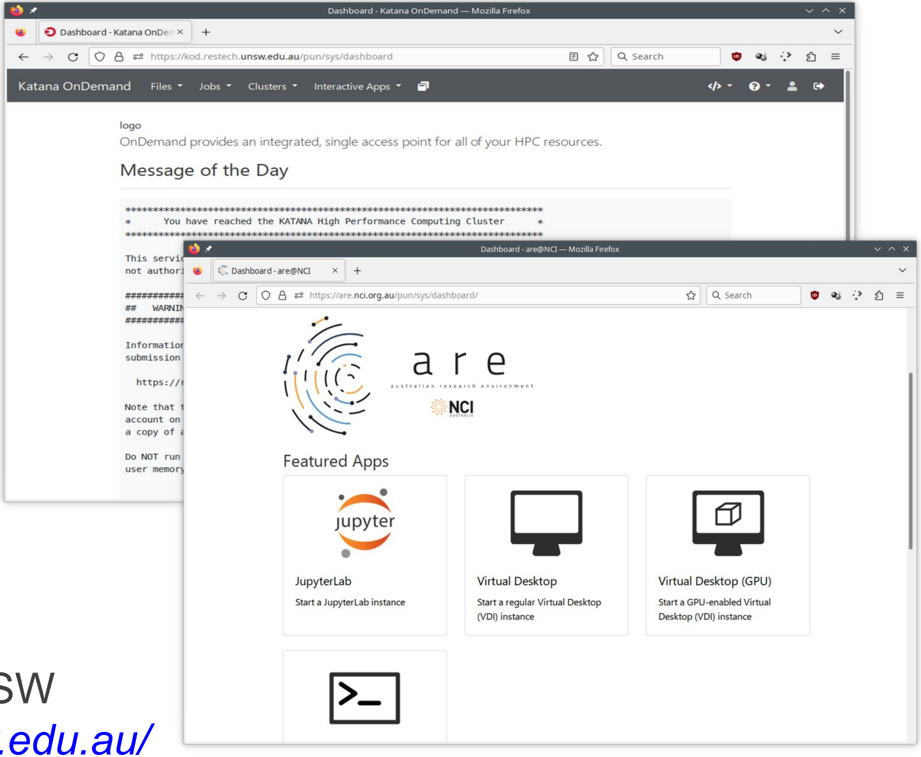
```
john@alzirr:~ $ cd src/trader/unix/src
john@alzirr:~/.../unix/src $ ls
COPYING  Makefile.am  README     configure.ac  doc  m4  src
INSTALL  NEWS         build-aux  data          lib  po
john@alzirr:~/.../unix/src $ tree -d
.
├── build-aux
├── data
│   ├── icons-128
│   ├── icons-16
│   ├── icons-22
│   ├── icons-24
│   ├── icons-256
│   ├── icons-32
│   ├── icons-48
│   ├── icons-512
│   ├── icons-64
│   └── icons-96
├── doc
├── lib
├── m4
├── po
└── src

17 directories
john@alzirr:~/.../unix/src $ git s
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working tree clean
john@alzirr:~/.../unix/src $ git log -1
commit 6b5c67a94ccef726ae582d8d42f96a39fa484530 (HEAD -> master, origin/master,
Author: John Zaitseff <J.Zaitseff@zap.org.au>
Date:   Wed Apr 4 20:14:06 2018 +1000

    Update the Russian TP team email address
john@alzirr:~/.../unix/src $ grep --color -Hn show_version src/*.[ch]
src/trader.c:97:  Function:   show_version - Show program version information
src/trader.c:104:static void show_version (void) __attribute__((noreturn));
src/trader.c:214:           show_version();
src/trader.c:277:// show_version: Show program version information
src/trader.c:279:void show_version (void)
john@alzirr:~/.../unix/src $ 
```

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

An easy-to-use interface for HPC

NCI Australian Research Environment and Katana OnDemand

- For jobs “just a bit bigger” than your desktop or laptop
- For graphical interactive jobs
  - “Quick and dirty” testing
  - Setting up for a longer job (e.g., Ansys/Fluent/CFX meshes)
- Uses your web browser: go to https://are.nci.org.au/ or https://kod.restech.unsw.edu.au/
- Katana OnDemand requires using the UNSW Virtual Private Network at https://vpn.unsw.edu.au/


Dashboard - Katana OnDemand — Mozilla Firefox
Dashboard - Katana OnDen
https://kod.restech.unsw.edu.au/pun/sys/dashboard
Search
Katana OnDemand
Files
Jobs
Clusters
Interactive Apps
logo

OnDemand provides an integrated, single access point for all of your HPC resources.

Message of the Day

```
*****************************************************************************
*      You have reached the KATANA High Performance Computing Cluster      *
*****************************************************************************

This servi
not authori

###########
##   WARNIN
###########

Informatior
submission

  https://r

Note that t
account on
a copy of a

Do NOT run
user memory
```

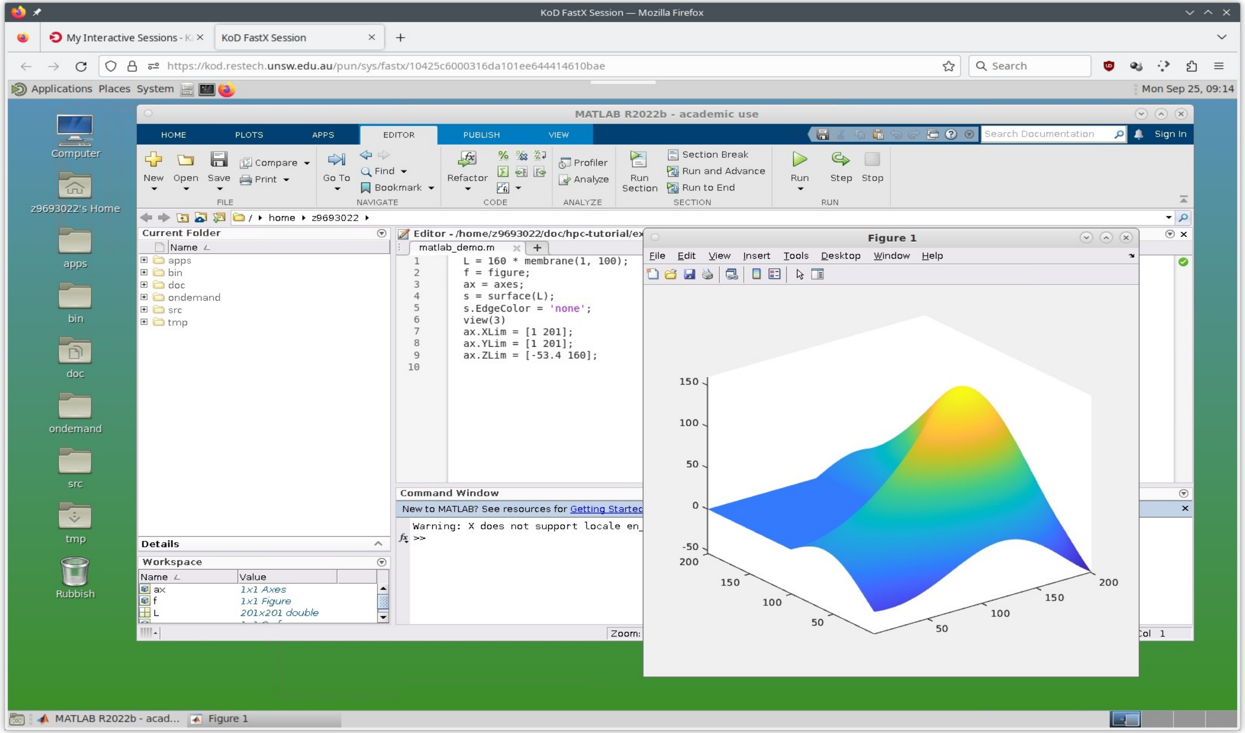
Dashboard - are@NCI — Mozilla Firefox
Dashboard - are@NCI
https://are.nci.org.au/pun/sys/dashboard/
Search
are
australian research environment
NCI
AUSTRALIA
Featured Apps
jupyter
JupyterLab
Start a JupyterLab instance
Virtual Desktop
Start a regular Virtual Desktop (VDI) instance
Virtual Desktop (GPU)
Start a GPU-enabled Virtual Desktop (VDI) instance

An easy-to-use interface on Katana

Available applications

- Ansys Workbench
- COMSOL
- Matlab
- ParaView
- Jupyter Notebook
- RStudio Server
- File browser
- Command line

This list is growing!



KoD FastX Session — Mozilla Firefox

https://kod.restech.unsw.edu.au/pun/sys/fastx/10425c6000316da101ee644414610bae

MATLAB R2022b - academic use

Editor - /home/z9693022/doc/hpc-tutorial/ex

matlab_demo.m

```
L = 160 * membrane(1, 100);
f = figure;
ax = axes;
s = surface(L);
s.EdgeColor = 'none';
view(3)
ax.XLim = [1 201];
ax.YLim = [1 201];
ax.ZLim = [-53.4 160];
```

Command Window

New to MATLAB? See resources for Getting Startec

```
Warning: X does not support locale en_
>>
```

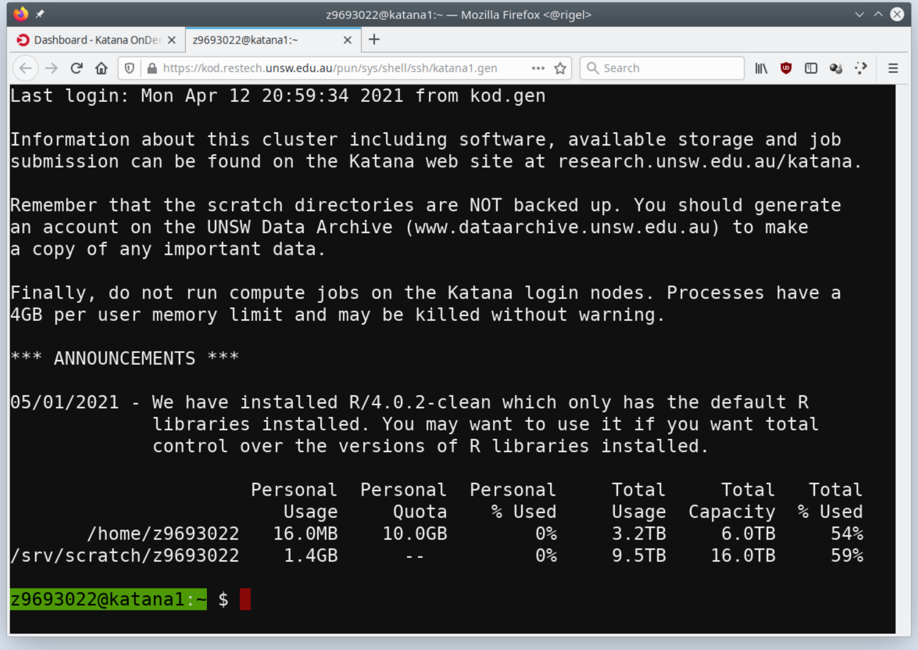
Workspace

Name	Value
ax	1x1 Axes
f	1x1 Figure
L	201x201 double

Using Katana On Demand shell access

Try it now:

- Make sure you are connected to the UNSW VPN (*https://vpn.unsw.edu.au/*)
- Open your web browser to Katana On Demand (*https://kod.restech.unsw.edu.au/*)
- Log in using your zID and zPass
- From the menu at the top of the page, select **Clusters**, then **Katana shell access**
- You will get a command line prompt: something like **`z9693022@katana1:~ $`**
- Press Ctrl and = (Equals) to increase the font size, Ctrl and – (Minus) to decrease it
- To exit, type `exit` and press ENTER


z9693022@katana1:~ — Mozilla Firefox <@rigel>
Dashboard - Katana OnDe
z9693022@katana1:~
https://kod.restech.unsw.edu.au/pun/sys/shell/ssh/katana1.gen
Search

```
Last login: Mon Apr 12 20:59:34 2021 from kod.gen

Information about this cluster including software, available storage and job
submission can be found on the Katana web site at research.unsw.edu.au/katana.

Remember that the scratch directories are NOT backed up. You should generate
an account on the UNSW Data Archive (www.dataarchive.unsw.edu.au) to make
a copy of any important data.

Finally, do not run compute jobs on the Katana login nodes. Processes have a
4GB per user memory limit and may be killed without warning.

*** ANNOUNCEMENTS ***

05/01/2021 - We have installed R/4.0.2-clean which only has the default R
             libraries installed. You may want to use it if you want total
             control over the versions of R libraries installed.

                      Personal  Personal  Personal     Total     Total   Total
                         Usage     Quota    % Used     Usage  Capacity  % Used
       /home/z9693022   16.0MB    10.0GB        0%     3.2TB     6.0TB     54%
/srv/scratch/z9693022    1.4GB      --          0%     9.5TB    16.0TB     59%

z9693022@katana1:~ $
```

Some common questions

- **Why does my browser refuse to connect to Katana On Demand (KOD)?**
  - You need to be connected to the UNSW VPN (*https://vpn.unsw.edu.au/*)

- **Why do I get “Your username and/or password do not match” from KOD?**
  - You may be typing your zID and/or zPass incorrectly
  - You must apply for a Katana account before you can use KOD

- **Why don’t I get a green prompt like that in the screenshot?**
  - This is part of a custom setup created by John Zaitseff, which you can also use

(Optional) Try it now (but please read the comments after “#”)**:**

```
source ~z9693022/.bashrc        # … to get a green prompt temporarily (until exit)
cp -p ~z9693022/.bashrc ~       # … to get John’s custom setup permanently
```

Connecting to a HPC system directly

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

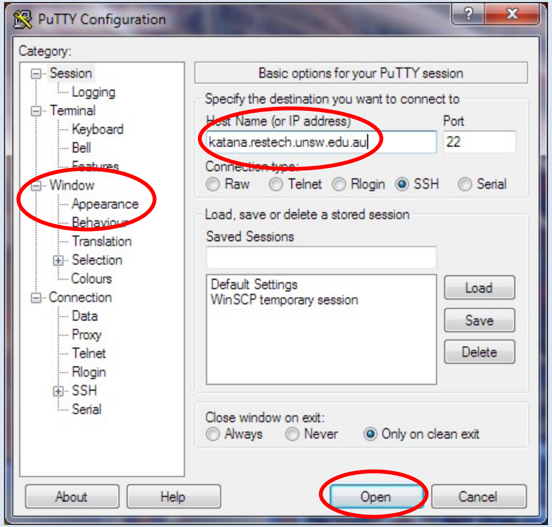
- Under Linux or macOS:
  - Open a terminal and type: `ssh` *`username@hostname`*
    (for example, `ssh z1234567@katana.restech.unsw.edu.au`)

- Under Windows:
  - Use **PuTTY**: can be downloaded from *https://www.putty.org/*
  - 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/*) but check licensing
  - Under Windows 10 or 11, can use SSH under **Windows Subsystem for Linux** (WSL)
  - Can also install **Cygwin**: “that Linux feeling on Windows” (*https://www.cygwin.com/*)

Connecting to a HPC system directly

Try it now:

- If you are running Windows, start PuTTY
- Specify Host Name as **katana.restech.unsw.edu.au**
- Select Window » Appearance on left-hand side, click Change, change the font to **Consolas**, **Regular**, size **16**, click OK
- Click Open
- Check first and last few digits of RSA2 fingerprint for security: `9b:4c:ba:a4:09:f3:4c:bd:39:ce:17:d9:18:5c:02:47`
- At the “login as:” prompt, enter your zID (e.g., `z1234567`), press ENTER, then enter the password (nothing will be shown) and press ENTER again
- You will get a command line prompt: something like `z9693022@katana1:~ $`
- To exit, type `exit` and press ENTER



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): katana.restech.unsw.edu.au

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

Saved Sessions

Default Settings
WinSCP temporary session

Close window on exit: ○ Always ○ Never ● Only on clean exit

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:

```
ls /apps                                   — command “ls”, argument “/apps”
~z9693022/bin/cmdline a1 a2                — command “~z9693022/bin/cmdline”, 2 arguments
~z9693022/bin/cmdline a1 a2 "a3 with spaces" — command with 3 arguments
```

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 “`info command`”
- To quit the **man** or **info** 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

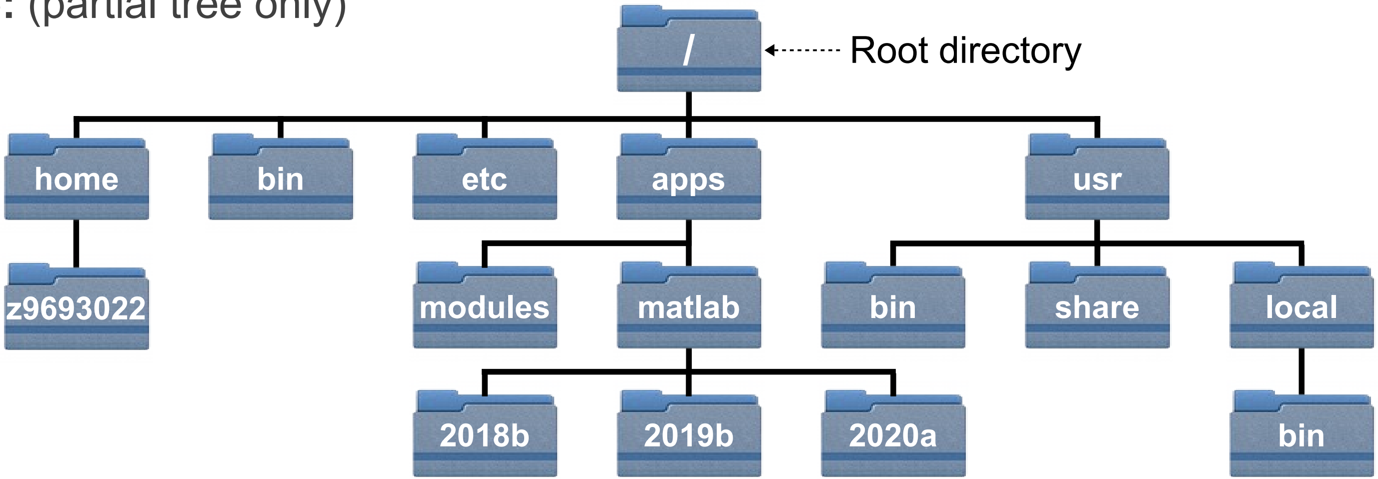
Try it now:

```
cd ~z9693022/src/trader-7.20      # Change directory to ~z9693022/src/trader-7.20
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
info 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
```

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

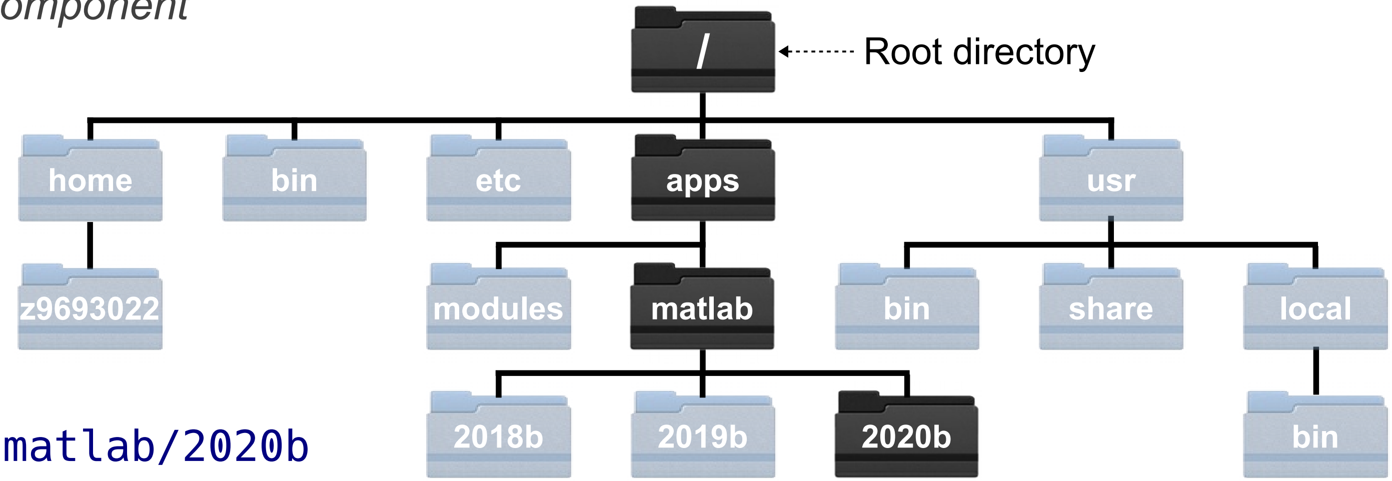
Example: (partial tree only)



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/2020b`

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/z9693022/src/trader-7.20*:

README	→	/home/z9693022/src/trader-7.20/README
src/trader.c	→	/home/z9693022/src/trader-7.20/src/trader.c
../trader-7.20.tar.xz	→	/home/z9693022/src/trader-7.20.tar.xz
src/.././README	→	/home/z9693022/src/trader-7.20/README
./README	→	/home/z9693022/src/trader-7.20/README

Important directories

- Home directory (system dependent): on Katana, `/home/`*`zID`*
- Binary directories for utility programs:
  - `/usr/bin` — for essential utilities and some applications
  - `/usr/local/bin` — for local utilities and applications
  - `/home/`*`zID`*`/bin` — for your own utilities
- On Katana, scratch directory for temporary files: `/srv/scratch/`*`zID`*
- On Katana, applications: `/apps`
- On Katana, module files: `/apps/Modules`

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 ~z9693022/src/trader-7.20   # Change directory to ~z9693022/src/trader-7.20
pwd                            # Should show “/home/z9693022/src/trader-7.20”
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/z9693022/src/trader-7.20/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/z9693022/src/trader-7.20” 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 almost 6000 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 ~z9693022/src/trader-7.20/src
alias z=~z9693022/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 ~z9693022                    # … and for user z9693022’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:

```
cd ~z9693022/src/trader-7.20/src
alias z=~z9693022/bin/cmdline      # Make a temporary shortcut “z” to the cmdline script

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?

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 macOS system (e.g., from Katana to Gadi), 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`*`:`"
- Can use **rsync**: "`rsync -vauSH [--delete] [--dry-run]` *`srcdir/ destdir/`*"
  - Powerful command but tricky! Note the trailing "`/`" on the directory arguments

Examples: (remember, don't type in the examples!)

```
cp -pr ~z9693022/src/trader-7.20 .
scp -p ~/file1.txt jjz561@gadi.nci.org.au:file2.txt
scp -p john@zap.org.au:src/README .
rsync -vauSH --delete ~/src/ jjz561@gadi.nci.org.au:~/src-unsw/
```

Playing with pathname expansion

Try it now:

```
cd ~; mkdir src; cd src

cp -pr ~z9693022/src/trader-7.20 .      # Note the trailing “.”!
cd trader-7.20                          # 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 macOS system: use “`scp`” or “`rsync`”
  - same as within a HPC/Linux system
- To copy files to and from a Windows machine: use **WinSCP**, **FileZilla**, or “`scp`” or “`rsync`” under Windows Subsystem for Linux or Cygwin
  - WinSCP may be downloaded from *https://winscp.net/eng/index.php*
  - FileZilla may be downloaded from *https://filezilla-project.org/*
  - both of these programs use a “drag-and-drop” graphical interface
  - the MobaXterm client (*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`”
- 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? On Katana, “`disk-usage`”, on Gadi “`lquota`” or (on other systems) “`quota`” or “`quota -s`”
  - On Katana: quota for your home directory is 15.0 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:

```
cd ~z9693022/src/trader-7.20

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 /usr/bin | grep Oct             # Which files were last modified in October?
ls -l /usr/bin | grep Oct | sort -nk5 # … numerically 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:

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

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

```
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:

```
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: "`info bash`"; "`man bash`"
- Some books:
  - William E. Shotts Jr., *The Linux Command Line*, No Starch Press, January 2012. ISBN 9781593273897, 9781593274269
  - Cameron Newham, *Learning the bash Shell, 3rd Edition*, O'Reilly Media, March 2005. ISBN 9780596009656, 9780596158965

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 `~z9693022/.emacs.d/init.el` on Katana — currently almost 2600 lines
  - Debra Cameron et al., *Learning GNU Emacs, 3rd Edition*, O’Reilly Media, December 2004. ISBN 9780596006488, 9780596104184
  - Arnold Robbins et al., *Learning the vi and Vim Editors, 7th Edition*, O’Reilly Media, July 2008. ISBN 9780596529833, 9780596159351

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:

```
qsub -I                          # After pressing ENTER, wait about 5 minutes until
                                 #     a new command line prompt is printed
mkdir ~/ex2; cd ~/ex2            # Create and change to ~/ex2
cp -p ~z9693022/doc/hpc-tutorial/examples/make-matlab-scripts .
                                 # Don't forget the trailing "."!
less ./make-matlab-scripts       # Examine the make-matlab-scripts script
                                 # Remember: "q" to quit less
./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, …)

Once you have answered these questions, type "`exit`" and press ENTER

Applications on the cluster

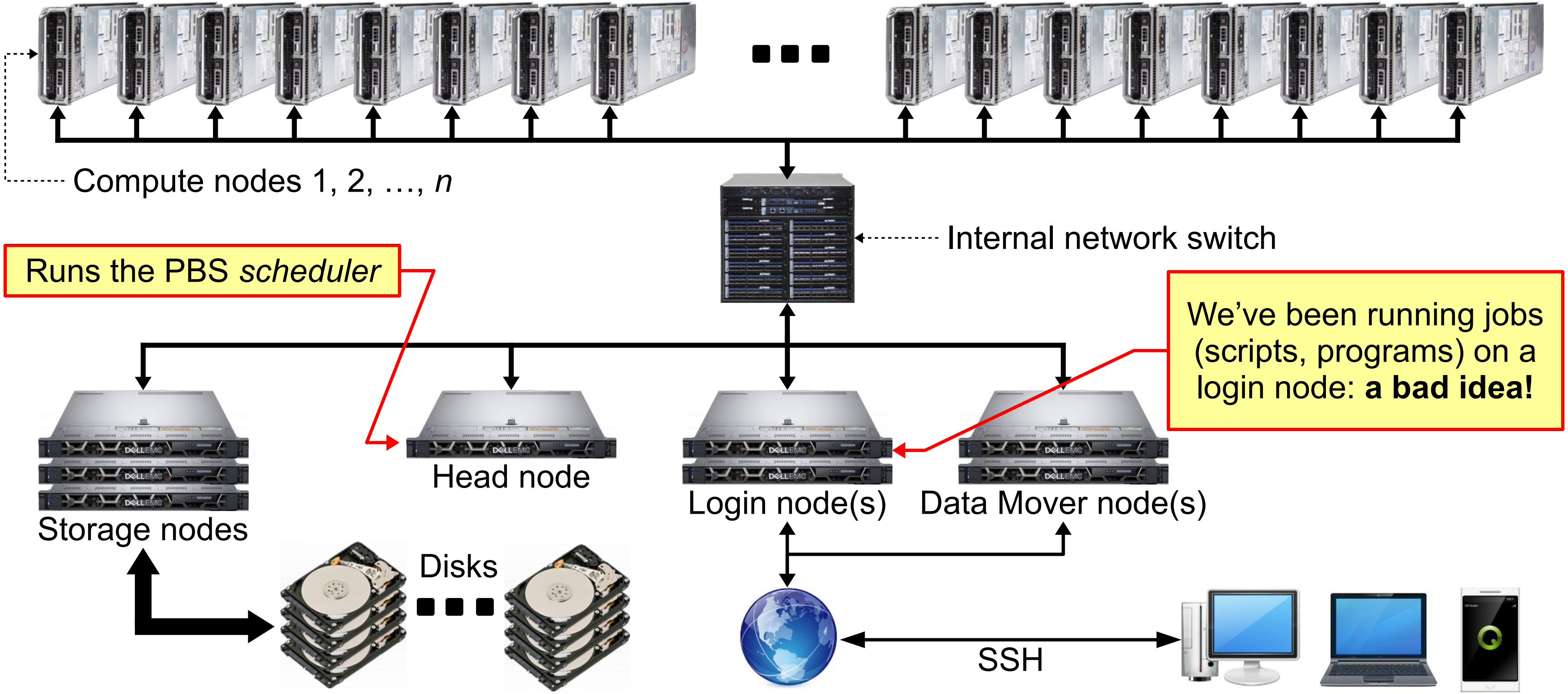
- Applications are managed using the *module system*
- On Katana, applications are stored in `/apps`
- On Katana, module files are stored in `/apps/Modules`
- Module files set shell environment variables such as `PATH`
- `PATH` controls where applications are searched (the *search path*)
- To see available 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/R2023b         # Set the PATH to include Matlab R2023b
echo $PATH                        # What does PATH look like now?
module unload matlab/R2023b       # We don’t want to use Matlab R2023b any more…
echo $PATH                        # PATH no longer contains the Matlab directory
```

HPC architecture revisited



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
  - 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 Katana (see *https://docs.restech.unsw.edu.au/*, “`man qsub`” and “`man pbs_resources`” for full details); many options have reasonable defaults:
  - `#PBS -N` *`scriptname`* — Set a name for the script
  - `#PBS -l select=`*`n`*`:ncpus=`*`m`*`:mem=`*`size`*`GB` — Request *n* compute nodes with *m* processor cores and *size* memory in GB in each
  - `#PBS -l walltime=`*`hh:mm:ss`* — How much time is required for running the job
  - `#PBS -M` *`email`* — Send notifications to the email address
  - `#PBS -m abe` — What notifications to send by email
  - `#PBS -j oe` — Join standard output and standard error into a single file instead of creating two files

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] [-u zID]`”
- Check status of each node on Katana: “`pstat | less -S`”
- Many systems have an overall system status page
  - On Gadi, the live status page is *https://nci.org.au/our-systems/status*

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 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 see the status of all nodes on Katana: "`pstat | less -S`"
  - The columns are **node name**, **queue name** (indicates nominal owner of the node), **node state**, number of processor **cores used/total**, **memory used/total**, and a **list of jobs** using that node `*` number of processor cores requested in each job.

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 -M replaceWithYourEmailAddress@unsw.edu.au
#PBS -m abe
#PBS -l walltime=00:05:00
#PBS -l select=1:ncpus=1:mem=1GB
cd $PBS_O_WORKDIR
```

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)
                           # … but please wait at least half a minute before doing so!
```

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:

```
PBS Job Id: 1133074.kman.restech.unsw.edu.au
Job Name:   job1
Execution terminated
Exit_status=0                                        — Successful job!
resources_used.cpupercent=0
resources_used.cput=00:00:00
resources_used.mem=2652kb
resources_used.ncpus=1
resources_used.vmem=2652kb
resources_used.walltime=00:00:31                     — 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:

```
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`”:

```
#PBS -l walltime=hh:mm:ss
                     → “qsub … -l walltime=hh:mm:ss …”
#PBS -l select=n:ncpus=m:mem=sizeGB
                     → “qsub … -l select=n:ncpus=m:mem=sizeGB …”
…
```

Running interactively

Try it now:

```
cd ~/ex1
hostname                              # Where am I running?  katana1–katana3 are login nodes
qsub -l walltime=0:10:00 -l select=1:ncpus=1:mem=4GB -I
                                      # Request an interactive job (you may need to wait)
```

Once a command line prompt appears:

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

Where to from here?

- Read the documentation for your HPC system:
  - Katana User Documentation: *https://docs.restech.unsw.edu.au/*
- Talk to your colleagues and/or supervisor about how they use High Performance Computing: with permission, copy their scripts to get started
- Undertake additional training through Research Technology Training:
  - Over 50 free courses run every year!
  - See *https://unsw.sharepoint.com/sites/Restech/SitePages/Events-&-Training.aspx*
- Come to **Drop-In Hour** with your questions, problems with code, HPC, data and more:
  - Currently via Microsoft Teams every Wednesday 1–2pm

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/vdZI1XIHfXXebuFy1

Keep in contact:

https://unsw.sharepoint.com/sites/Restech
<*restech@unsw.edu.au*>

Image credit: UNSW Sydney