

# An Introduction to High Performance Computing on the CSD3 Cluster

Paul Sumption

[support@hpc.cam.ac.uk](mailto:support@hpc.cam.ac.uk)

Research Computing Services (<http://www.hpc.cam.ac.uk/>)  
University Information Services (<http://www.uis.cam.ac.uk/>)

# Welcome

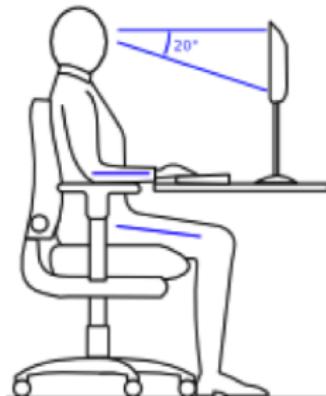
- ▶ Please sign in on the **attendance sheet**.
- ▶ Please fill in the **online feedback** at the end of the course: There is a link to this on your desktop.
- ▶ Keep your belongings with you.
- ▶ Course files can be downloaded from: [www.cs3.cam.ac.uk](http://www.cs3.cam.ac.uk)

# Plan of the Course

- 10:00 Part 1: Course Introduction
- 10:15 Part 2: HPC - Basic Concepts
- 11:00 BREAK
- 11.20 Part 3: HPC - Facilities
- 12:00 Part 4: HPC - Connecting
- 13:00 LUNCH
- 14:00 Part 5: HPC - User Environment
- 15:00 Part 6: HPC - Submission scripts
- 16:30 FEEDBACK and CLOSE

## **Part I: Introduction**

# Health and Safety



Your trainers for today will be:

- ▶ Paul Sumption — Research Computing Technical Liaison
- ▶ Simon Flood — Research Computing System Administrator
- ▶ Matthew Archer — Research Software Engineering Team
- ▶ Please ask questions and let us know if you need assistance.

# Introduction: Training accounts

- ▶ For our practical exercise's we will use HPC training accounts.
- ▶ You will find two pieces of paper on your desk.
- ▶ 1: A terms and conditions form for you to sign.
- ▶ 2: Your training account details.
- ▶ Your training account will only be valid for today.

## Introduction: Login node

- ▶ For our practical exercise's we will use the login node:  
[login-cpu.hpc.cam.ac.uk](http://login-cpu.hpc.cam.ac.uk).
- ▶ Later in the course you will see that different nodes types may require you to use other login nodes.

# Introduction: Security

- ▶ Boring but very, very important...
- ▶ Cambridge IT is under constant attack by would-be intruders.
- ▶ Your data and research career is threatened by intruders.
- ▶ Cambridge systems are high profile and popular targets.
- ▶ Don't let intruders in.

# Introduction: Security

1. Keep your password (or private key passphrase) safe.
2. Always choose strong passwords.
3. Your UIS password is used for multiple systems so keep it secure!
4. Keep the software on your laptops/tablets/PCs up to date this includes home computers especially if you are using the VPN to connect in.
5. Don't share accounts (this is against the rules anyway).

# Pre-requisites

- ▶ Pre-requisites: Basic Unix/Linux command line experience
- ▶ Suggested Courses:
- ▶ Unix: Introduction to the Command Line Interface (Self-paced)  
<https://www.training.cam.ac.uk/ucs/Course/ucs-unixintro1>
- ▶ Shell scripting experience is desirable:
- ▶ Unix: Simple Shell Scripting for Scientists  
<https://www.training.cam.ac.uk/ucs/Course/ucs-scriptscli>

# Navigating your terminal

Useful commands for navigating your terminal.

- ▶ `cd <dirname >` - change into a directory
- ▶ `ls <dirname >` - list the contents of a directory
- ▶ `cd` or `cd ~` - change into your home folder
- ▶ `cd ..` - change back one folder
- ▶ `man ls` - will bring up the manual page for the ls command
- ▶ `pwd` - print working directory

## **Part II: Basics**

# Basics: Why Buy a Big Computer?

What types of big problem might require a “Big Computer”?

*Compute Intensive*: A single problem requiring a large amount of computation.

*Memory Intensive*: A single problem requiring a large amount of memory.

*Data Intensive*: A single problem operating on a large amount of data.

*High Throughput*: Many unrelated problems to be executed in bulk.

## Basics: Compute Intensive Problems

- ▶ Distribute the **work** for a **single problem** across multiple CPUs to reduce the execution time as far as possible.
- ▶ Program workload must be *parallelised*:
  - Parallel programs split into copies (processes or threads).
  - Each process/thread performs a part of the work on its own CPU, concurrently with the others.
  - A well-parallelised program will fully exercise as many CPUs as there are processes/threads.
- ▶ The CPUs typically need to exchange information rapidly, requiring specialized communication hardware.
- ▶ Many use cases from Physics, Chemistry, Engineering, Astronomy, Biology...
- ▶ The traditional domain of **HPC** and the **Supercomputer**.

## Basics: Scaling & Amdahl's Law

- ▶ Using more CPUs is not necessarily faster.
- ▶ Typically parallel codes have a scaling limit.
- ▶ Partly due to the system overhead of managing more copies, but also to more basic constraints;
- ▶ Amdahl's Law (idealized):

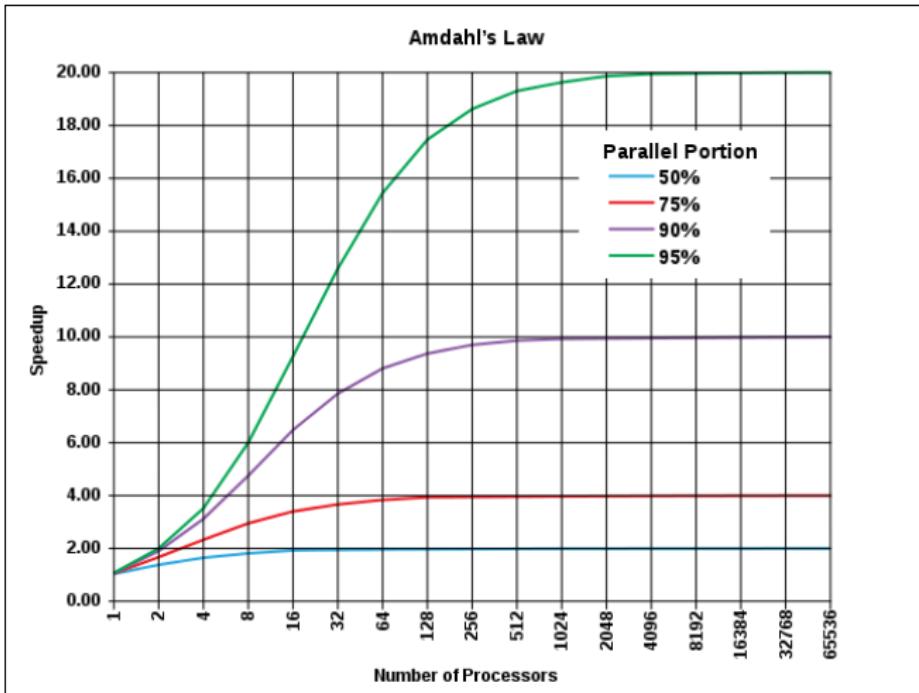
$$S(N) = \frac{1}{(1 - p + \frac{p}{N})}$$

where

$S(N)$  is the fraction by which the program has sped up  
relative to  $N = 1$

$p$  is the fraction of the program which can be parallelized  
 $N$  is the number of CPUs.

# Basics: Amdahl's Law



<http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg>

# The Bottom Line

- ▶ Parallelisation requires effort:
  - ▶ There are libraries to help (e.g. [OpenMP](#), [MPI](#)).
  - ▶ First optimise performance on one CPU, then make  $p$  as large as possible.
- ▶ The scaling limit: eventually using more CPUs becomes [detrimental](#) instead of helpful.

## Basics: Data Intensive Problems

- ▶ Distribute the **data** for a **single problem** across multiple CPUs to reduce the overall execution time.
- ▶ The *same* work may be done on each data segment.
- ▶ Rapid movement of data to and from disk is more important than inter-CPU communication.
- ▶ **Big Data** problems of great current interest -
- ▶ Hadoop/MapReduce
- ▶ Life Sciences (genomics) and elsewhere.

## Basics: High Throughput

- ▶ Distribute **independent, multiple problems** across multiple CPUs to reduce the overall execution time.
- ▶ Workload is trivially (or *embarrassingly*) parallel:
  - \* Workload breaks up naturally into *independent* pieces.
  - \* Each piece is performed by a separate process/thread on a separate CPU (concurrently).
  - \* **Little or no inter-CPU communication.**
- ▶ Emphasis is on throughput over a period, rather than on performance on a single problem.
- ▶ Compute intensive capable  $\Rightarrow$  high throughput capable
- ▶ **Compute intensive capable  $\neq$  high throughput capable**

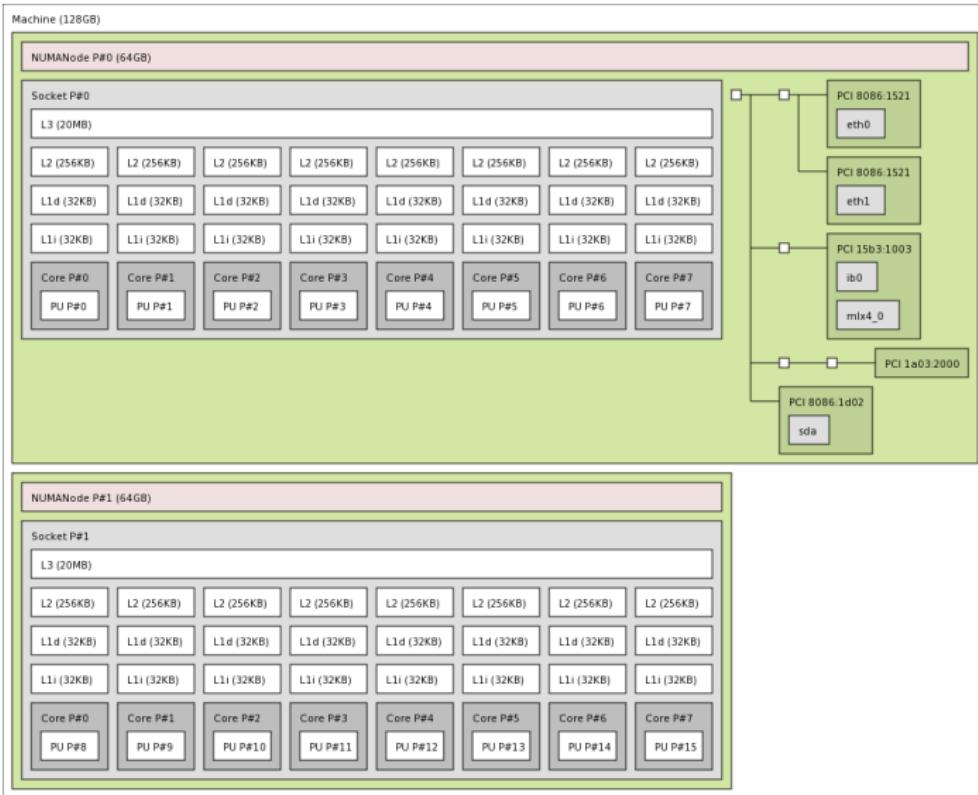
## Basics: Memory Intensive Problems

- ▶ Require aggregation of large memory into a **single system image** (i.e. a single computer running Linux).
- ▶ Technically more challenging to build machines (very fast, low latency interconnection between **all** CPUs and **all** memory).
- ▶ Coding/porting easier (memory appears seamless).
- ▶ Optimisation harder (memory is actually highly nonuniform).
- ▶ Historically, the arena of large **SGI** systems.
- ▶ Nowadays, similar complexity inside single commodity servers.

# Basics: Inside a Modern Computer

- ▶ Today's commodity servers already aggregate both CPUs and memory to make a single system image in a single box.
- ▶ Even small computers now have multiple CPU cores per socket  
    ⇒ each socket is a **Symmetric Multi-Processor (SMP)**.
- ▶ Larger computers have multiple sockets (each with local memory):  
    all CPU cores (unequally) share the node memory  
    ⇒ the node is a **shared memory** multiprocessor  
    with Non-Uniform Memory Architecture (**NUMA**)  
    but users still see a single computer (**single system image**).

# Basics: Inside a Modern Computer

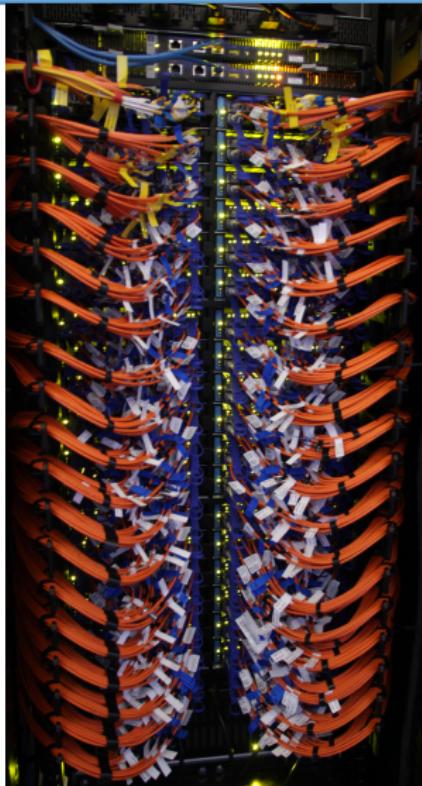


# Basics: How to Build a Supercomputer

- ▶ A supercomputer aggregates contemporary CPUs and memory to obtain increased computing power.
  - ▶ Usually today these are **clusters**.
1. Take some (multicore) CPUs plus some memory.
    - ▶ Could be an off-the-shelf server, or something more special.
    - ▶ A NUMA, shared memory, multiprocessor building block: a **node**.

# Basics: How to Build a Supercomputer

2. Connect the nodes with one or more **networks**.  
Faster network for **inter-CPU communication across nodes**.  
Slower network for **management** and **provisioning**.  
**Storage** may use either.



# Basics: How to Build a Supercomputer

### 3. Logically bind the nodes

- ▶ Clusters consist of distinct nodes (i.e. separate Linux computers) on common private network(s) and controlled centrally.
  - \* Private networks allow CPUs in different nodes to communicate.
  - \* Clusters are **distributed memory** machines:  
Each process/thread sees only its local node's CPUs and memory (without help).
  - \* **Each process/thread must fit within a single node's memory.**

# Basics: Programming a Multiprocessor Machine

- ▶ Non-parallel (serial) code
  - \* For a single node as for a workstation.
  - \* Typically run as many copies per node as CPUs, assuming node memory is sufficient.
  - \* Replicate across multiple nodes.
- ▶ Parallel code
  - \* Shared memory methods within a node.  
E.g. pthreads, OpenMP.
  - \* Distributed memory methods spanning multiple nodes.  
Message Passing Interface (MPI).

## Basics: Summary

- ▶ Why have a supercomputer?
  - ▶ Big single problems, many problems, Big Data.
- ▶ Most current supercomputers are clusters of separate nodes.
- ▶ Each node has multiple CPUs and non-uniform shared memory.
- ▶ Parallel code uses shared memory (threads/OpenMP) within a node, distributed memory (MPI) spanning multiple nodes.
- ▶ Non-parallel code uses the memory of one node, but may be copied across many nodes (i.e. High Throughput, single core).

## **Part III: HPC Facilities**

- ▶ Cambridge Service for Data Driven Discovery
- ▶ Peta4 — Intel CPU cluster
- ▶ Wilkes2 — NVIDIA GPU cluster
- ▶ Hadoop-based data analytic platform
- ▶ Burst buffer
- ▶ Industry users through CORE.

- ▶ Each compute node:
  - \* 2x16 cores, Intel Skylake 2.6 GHz **32 CPUs**
  - \* 192 GB or 384 GB RAM **6 GB or 12 GB per CPU**
  - \* 100 Gb/sec Omni-Path **10 GB/sec (for MPI and storage)**
- ▶ 768 compute nodes
- ▶ 8 login nodes ([login-cpu.hpc.cam.ac.uk](http://login-cpu.hpc.cam.ac.uk))

- ▶ Each compute node:
  - \* 4 × NVIDIA P100 GPU~~4 GPUs~~
  - \* 1x12 cores, Intel Broadwell 2.2 GHz~~12 CPUs~~
  - \* 96 GB RAM~~96 GB RAM~~
  - \* 100 Gb/sec (4X EDR) Infiniband.~~10 GB/sec (for MPI and storage)~~
- ▶ 90 compute nodes.
- ▶ 8 login nodes ([login-gpu.hpc.cam.ac.uk](http://login-gpu.hpc.cam.ac.uk)).

# Peta4-KNL (Intel Phi)

- ▶ Each compute node:
  - \* 64 cores, Intel Phi 7210~~256~~ CPUs
  - \* 96 GB RAM~~96 GB~~ RAM
  - \* 100 Gb/sec Omni-Path~~10 GB/sec~~ (for MPI and storage)
- ▶ 342 compute nodes
- ▶ Shared login nodes with Peta4-Skylake

# HPCS: Storage

- ▶ Multi-petabytes split across multiple filesystems with tape.
- ▶ Lustre cluster filesystem:
  - \* Multiple RAID6 back-end disk volumes.
  - \* Multiple object storage servers.
  - \* Single metadata server.
  - \* Tape-backed HSM on newest filesystems.
  - \* 4 GB/sec overall read or write.
  - \* Prefers big read/writes over small.
- ▶ For active HPC work only.

# Obtaining an Account and Support

- ▶ To apply for an account, complete our online form:
- ▶ <https://www.cs3.cam.ac.uk/services/applying-for-resources>
- ▶ For support enquiries please email our Service Desk:
- ▶ [support@hpc.cam.ac.uk](mailto:support@hpc.cam.ac.uk)

## Part IV: Using HPC

## Basics: Connecting

- ▶ When connecting to the cluster will we use the SSH secure protocol only.
- ▶ We will use the Linux workstations during this course
- ▶ Please check your workstation is booted into Ubuntu Linux, ask if you need help with this.
- ▶ You are welcome to use your own laptop, however you may need to install some software in order to connect.
- ▶ Later in this section of the slides we will cover the software you may need to install on your own computer.

# Basics: Connecting

- ▶ SSH secure protocol only.  
Supports login, file transfer, remote desktop...
- ▶ HPCS now allows direct access from outside of the CUDN.  
It is still worth configuring the UIS VPN service  
<http://www.ucs.cam.ac.uk/vpn>  
Useful for other CUDN only services.

# Basics: Connecting

- ▶ To access the Peta4-Skylake (CPU cluster) nodes:  
ssh <username>@login-cpu.hpc.cam.ac.uk
- ▶ To access the Peta4-KNL (KNL cluster) nodes:  
ssh <username>@login-knl.hpc.cam.ac.uk
- ▶ To access the Wilkes2-GPU (GPU cluster) nodes:  
ssh <username>@login-gpu.hpc.cam.ac.uk

# Connecting: Linux Clients

- ▶ `ssh`, `scp`, `sftp`, `rsync`  
Installed (or installable), in Ubuntu we will use 'Terminal'.
- ▶ X Windows, for using graphical applications remotely This is already installed on your desktop.

## Connecting: Login

- ▶ From Linux/MacOSX/UNIX (or Cygwin):  
`ssh -Y abc123@login-cpu.hpc.cam.ac.uk`
- ▶ From graphical clients:  
Host: `login-cpu.hpc.cam.ac.uk`  
Username: `abc123` (your UCAM account name)
- ▶ `login-cpu.hpc` will map to a random login node  
i.e. one of `login-e-9, ..., login-e-16`  
**NB you'll never connect to the head node.**

## Connecting: First time login

- ▶ The first connection to a particular hostname produces the following:

The authenticity of host 'login-e-10.hpc.cam.ac.uk (128.232.224.47)' can't be established.

RSA key fingerprint is

0b:ef:59:90:fb:13:4a:c9:56:82:7b:cd:4b:2b:e1:3b.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'login-sand2.hpc.cam.ac.uk' (RSA) to the list of known hosts.

- ▶ One should always check the fingerprint before typing “yes”.
- ▶ Graphical SSH clients *should* ask a similar question.
- ▶ Designed to detect fraudulent servers.

## Connecting: First time login

- ▶ You may be presented with any of the following fingerprints (depending on your client):

MD5:0b:ef:59:90:fb:13:4a:c9:56:82:7b:cd:4b:2b:e1:3b

SHA256:sSkVfzpwpjwiFvxLcdPoDpN8IsN3kt0ZSywhDhPKZPAG

MD5:34:9b:f2:d2:c6:b3:5c:63:99:b7:27:da:5b:c8:16:fe

SHA256:HsiY10e0M8tS6JwR76PeQQA/VB7r8675BzG50YQ4h34

MD5:64:7c:7c:ff:05:9d:0e:dc:06:fe:f1:c2:10:37:7a:85

SHA256:wq91jBfPa71XXpQq+rk5JTBXLJ0/kXj0c5A7rp4ENzA

# Connecting: File Transfer

- ▶ From Linux/MacOSX/UNIX (or Cygwin):

`rsync -av old_directory/ abc123@login.hpc.cam.ac.uk:scratch/new_directory`  
copies contents of old\_directory to `~/scratch/new_directory`.

`rsync -av old_directory abc123@login.hpc.cam.ac.uk:scratch/new_directory`  
copies old\_directory (and contents) to  
`~/scratch/new_directory/old_directory`.

- \* For transfers in the opposite direction, place the remote machine as the first argument.

## Basics: Connecting from other clients

- ▶ When using your own computer you may need to install some software in order to connect.
- ▶ There are quite a few choices of software packages for this, we will just cover a few.
- ▶ Our website

<https://www.cs3.cam.ac.uk/using-clusters/connecting>  
covers this in more detail.

# Connecting: MacOSX/UNIX Clients

- ▶ **ssh, scp, sftp, rsync**

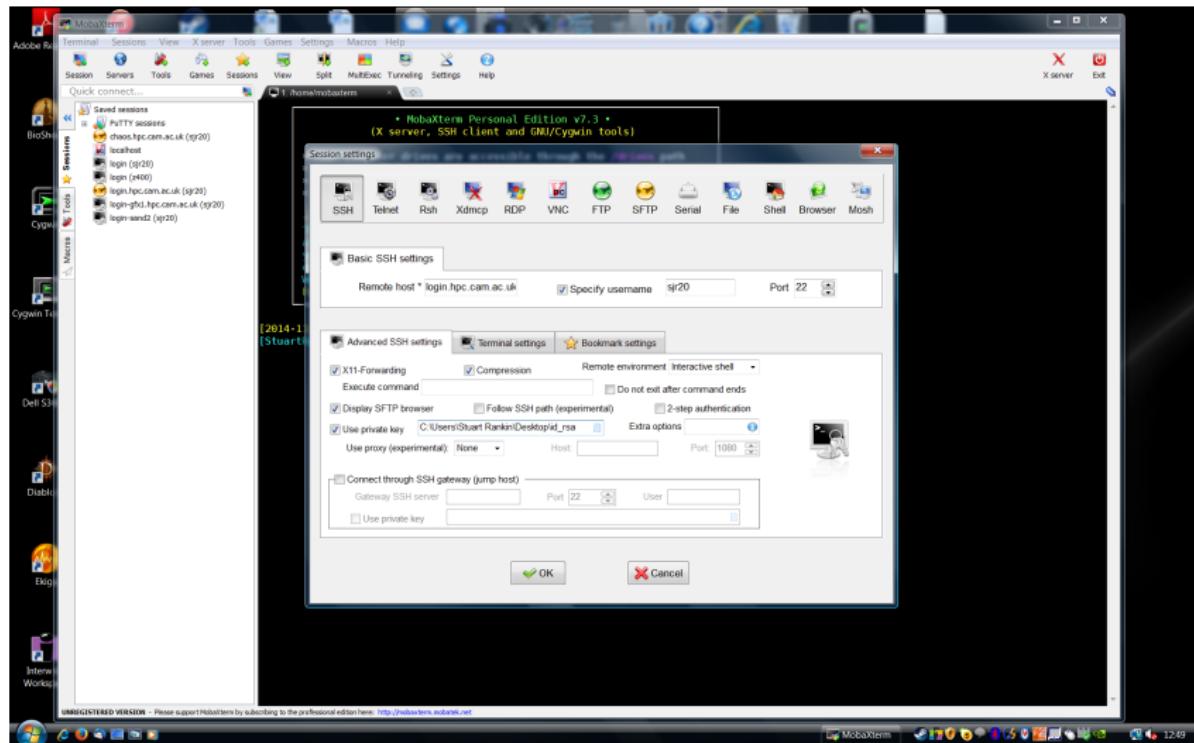
Installed (or installable) OS X has a native terminal package. This can be launched by clicking Apple >Go >Utilities and then clicking the Terminal icon.

- ▶ TurboVNC (for remote desktop, 3D optional)  
<http://sourceforge.net/projects/turbovnc/files/>
- ▶ On MacOSX, install XQuartz to display remote graphical applications.  
<http://xquartz.macosforge.org/landing/>

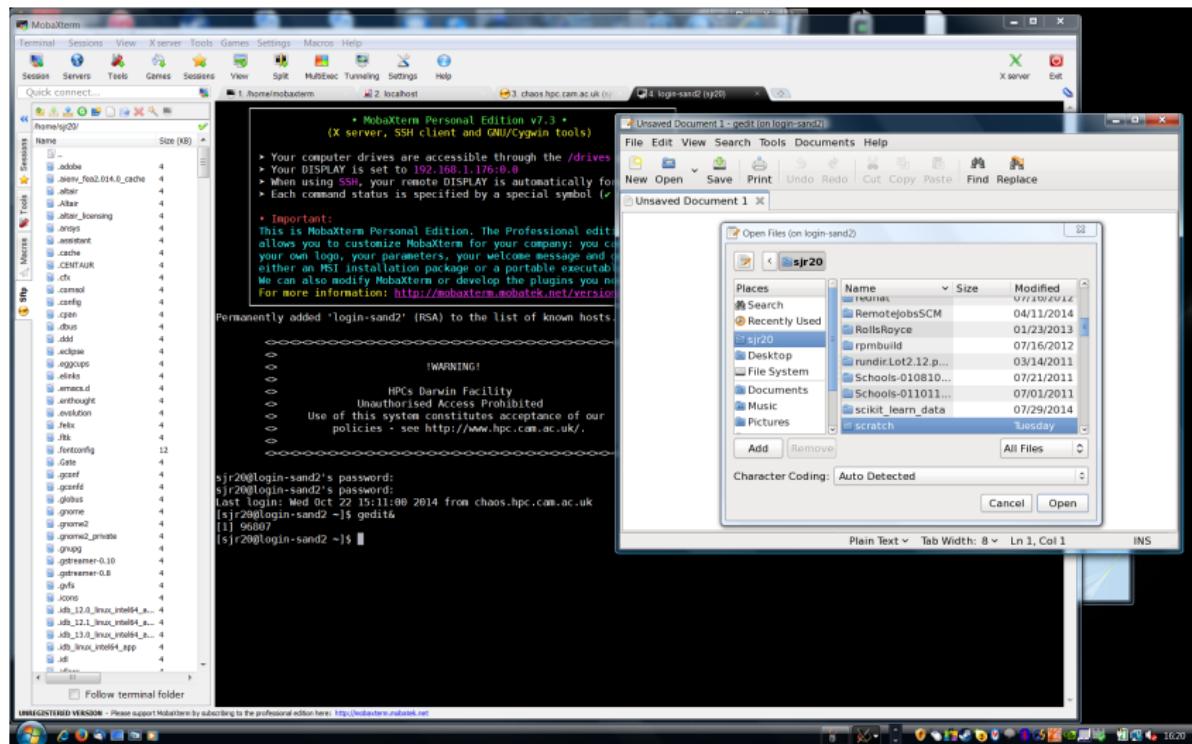
# Connecting: Windows Clients

- ▶ putty, pscp, psftp  
<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>
- ▶ WinSCP  
<http://winscp.net/eng/download.php>
- ▶ TurboVNC (remote desktop, 3D optional)  
<http://sourceforge.net/projects/turbovnc/files/>
- ▶ Cygwin (provides an application environment similar to Linux)  
<http://cygwin.com/install.html>  
Includes X server for displaying graphical applications running remotely.
- ▶ MobaXterm  
<http://mobaxterm.mobatek.net/>

# MobaXterm SSH (Windows)



# MobaXterm SSH (Windows)



# Simple command line operations

- ▶ Some simple exercises will help us gauge your experience.
- ▶ Exercise 1: Login with SSH
- ▶ Exercise 2: SFTP file transfer
- ▶ Exercise 3: Learn more about a command
- ▶ Exercise 4: Unzip the exercises.tar file
- ▶ Exercise 5: Navigating the command line

## Exercise 1: Login

Using a Linux terminal you will login to the cluster with your HPC training account.

- ▶ Start the terminal by double clicking on the terminal icon
- ▶ In your terminal enter:
- ▶ ssh -Y **abc123@login-cpu.hpc.cam.ac.uk**  
Replace abc123 with your training account username
- ▶ Enter your password as supplied on the sheet
- ▶ Leave this terminal open, you will need it for exercise 3!

## Exercise 2: Transfer some files

You will need to transfer the exercise files to the cluster.

- ▶ Open a second Linux terminal on your training computer.
- ▶ Enter this command: `cd ~\Course_material`
- ▶ Check the file 'exercises.tar' is in your directory listing
- ▶ Hint: `ls`

## Exercise 3: Transfer some files

Transfer the exercises.tar to your HPC home folder.

- ▶ In the local terminal on your training computer enter the command:
- ▶ `sftp abc123@login.hpc.cam.ac.uk`  
Change abc123 to your training account username
- ▶ The command: `put exercises.tar` will transfer the file from your local computer to the remote one
- ▶ Type 'exit' to close the local terminal

## Exercise 3: Learn more about a command

- (a) View the man page for the `cp` command by doing `man cp`. Use **SPACE** to page down and **b** to page up. Press **q** to exit the manual page command.
- (b) Copy `exercises.tar` to the `~/hpc-work` directory. Note that `~` is just a convenient shorthand for your home directory. Omitting the `~/` will look for a `hpc-work` in the current directory.

*Hints:* Do `cp exercises.tar ~/hpc-work/`. Note that you can often reduce typing by pressing **TAB**.

## Exercise 4: Unzip the excercises.tar file

- (a) Use the `cd` command to enter the `~/hpc-work` directory and then list the contents — you should see the copy of `exercises.tgz`.

*Hints:* Do `cd ~/hpc-work/` then `ls -al`. Note that `cd ..` will take you back up one step to the home directory.

- (b) Unpack the tar archive to create an exercise subdirectory.

*Hints:* Do `tar -zxvf exercises.tar`

## Exercise 5: File listings

- (a) In a terminal logged into the cluster list the contents of your current directory `ls`. This won't show everything — use `ls -al` for a long listing showing all files. Initially you will start in your home directory — use `pwd` to print the name of your current working directory. If you get lost, you can always do `cd` without arguments to return to your home directory.
- (b) Focus your long listing on `all files with names beginning "exercise"`.

*Hints:* Do `ls -al exercise*`

- (c) Print a long listing of the subdirectory `hpc-work`.

*Hints:* Do `ls -al hpc-work/`. Note that omitting the `/` reveals that the item `hpc-work` is actually a shortcut (technically a symbolic link) to `/hpc-work/username`.

## Part V: Using HPC

# Using HPC: User Environment

- ▶ CentOS Linux 7.4 ([Red Hat Enterprise Linux 7.4 rebuild](#))
  - ▶ bash shell
  - ▶ Gnome or XFCE4 desktop ([if you want](#))
  - ▶ GCC compilers and other development software.
- ▶ But you don't need to know that.

# User Environment: Filesystems

When you apply for an HPC account a home directory is created for you.

- ▶ [/home/abc123](#)

- ▶ 40GB quota.
- ▶ Visible equally from all nodes.
- ▶ Single storage server.
- ▶ Hourly, daily, weekly snapshots copied to tape.
- ▶ Not intended for job outputs or large/many input files.

- ▶ [/rds/user/abc123/hpc-work](#)

- ▶ Visible equally from all nodes.
- ▶ Larger and faster (1TB initial quota).
- ▶ Intended for job inputs and outputs.
- ▶ **Not backed up.**

# Our storage services

- ▶ We have several storage services for users that need to exceed 1TB.
- ▶ <http://www.uis.cam.ac.uk/initiatives/storage-strategy/storage-services>
- ▶ The most relevant services to HPC are RCS and RDS.
- ▶ RCS - Research Cold Store is for data that isn't changing, data goes to disk then two sets of tapes.
- ▶ RDS - Research Data Store, non backed up high performance storage for projects.

## Filesystems: Snapshots

- ▶ ZFS snapshots on /home are taken hourly, daily and weekly
- ▶ No backups are made of data in the RDS directories - take care when deleting.
- ▶ Snapshots are not backups, they are retained for two weeks
- ▶ Very short lived files are unlikely to get written to snapshots
- ▶ The file needs exist long enough to get into a snapshot.
- ▶ It is possible to search /home/.zfs/snapshot and browse the snapshots
- ▶ Locate the most recent version of the file then copy it back to your home folder

# Filesystems: Quotas

- ▶ quota

```
[ps459@login-e-13 hpc-work]$ quota
Filesystem  GiBytes   quota   limit   grace   files   quota   limit   grace User/group
/home        12.4     40.0    40.0    0       ---- No ZFS File Quotas  ----- U:ps459
/rds-d1      89.2    1024.0  1126.4  -       757345  1048576  1048576  -       G:ps459
/rds-d1      22.1    1024.0  1024.0  -       113475  1048576  1048576  -       G:rds-ps459-test
```

- ▶ Aim to stay below the soft limit (*quota*).
- ▶ Once over the soft limit, you have 7 days grace to return below.
- ▶ When the grace period expires, or you reach the hard limit (*limit*), no more data can be written.
- ▶ It is important to rectify an out of quota condition ASAP.

# Filesystems: Quotas

- ▶ quota

```
[ps459@login-e-13 hpc-work]$ quota
Filesystem  GiBytes   quota   limit   grace   files   quota   limit   grace User/group
/home        12.4     40.0    40.0     0      ---- No ZFS File Quotas  ----- U:ps459
/rds-d1      89.2    1024.0  1126.4   -      757345 1048576 1048576  -      G:ps459
/rds-d1      22.1    1024.0  1024.0   -      113475 1048576 1048576  -      G:rds-ps459-test
```

- ▶ Aim to stay below the soft limit (*quota*).
- ▶ Once over the soft limit, you have 7 days grace to return below.
- ▶ When the grace period expires, or you reach the hard limit (*limit*), no more data can be written.
- ▶ It is important to rectify an out of quota condition ASAP.

# Filesystems: Permissions

- ▶ Be careful and if unsure, please ask support.
  - ▶ Can lead to accidental destruction of your data or account compromise.
- ▶ Avoid changing the permissions on your home directory.
  - ▶ Files under /home are particularly security sensitive.
  - ▶ Easy to break passwordless communication between nodes.

# Using HPC: Software

- ▶ Free software accompanying Red Hat Enterprise is (or can be) provided.
- ▶ Other software (free and non-free) is available via modules.
- ▶ Some proprietary software may not be generally accessible.
- ▶ See <http://www.hpc.cam.ac.uk/using-clusters/software>.
- ▶ New software may be possible to provide on request.
- ▶ Self-installed software must be properly licensed.
- ▶ *sudo will not work. (You should be worried if it did.)*

# User Environment: Environment Modules

- ▶ Modules load or unload additional software packages.
- ▶ Some are **required** and automatically loaded on login.
- ▶ Others are optional extras, or possible replacements for other modules.
- ▶ **Beware** unloading default modules in `~/.bashrc`.
- ▶ **Beware** overwriting environment variables such as `PATH` and `LD_LIBRARY_PATH` in `~/.bashrc`. If necessary append or prepend.

# User Environment: Environment Modules

- ▶ Currently loaded:

```
module list
Currently Loaded Modulefiles:
  1) dot                      2) singularity/current
  ....plus several more default modules
                                         2) intel
```

- ▶ Available:

```
module av
```

# User Environment: Environment Modules

- ▶ What is:

```
module whatis openmpi-3.0.0-gcc-4.8.5-n2hvjgm
openmpi-3.0.0-gcc-4.8.5-n2hvjgm: The Open MPI Project is an open source...
```

- ▶ Load:

```
module load openmpi-3.0.0-gcc-5.4.0-4ryklvu
```

- ▶ Unload:

```
module unload openmpi-3.0.0-gcc-5.4.0-4ryklvu
```

# User Environment: Environment Modules

- ▶ Matlab

```
module load matlab/r2017b
```

- ▶ Invoking matlab in batch mode:

```
matlab -nodisplay -nojvm -nosplash command
```

where the file `command.m` contains your matlab code.

- ▶ The current site license contains the Parallel Computing Toolbox.

# User Environment: Environment Modules

- ▶ Purge and Load Defaults:

```
module purge
module load rhel7/default-peta4
```

# User Environment: Compilers

- ▶ Load a newer GCC

```
module show gcc-7.2.0-gcc-4.8.5-pqn7o2k
module load gcc-7.2.0-gcc-4.8.5-pqn7o2k
Your commands to compile your software....
```

- ▶ If you have compiled software yourself your run time environment must match compile time environment!

```
gcc -O3 -mtune=native code.c -o prog
gfortran -O3 -mtune=native code.f90 -o prog
```

```
module load openmpi-3.0.0-gcc-4.8.5-n2hvjgm
mpicc -O3 -mtune=native mpi_code.c -o mpi_prog
mpif90 -O3 -mtune=native mpi_code.f90 -o mpi_prog
```

## Excercise 6: Environment Modules

- ▶ Connect to the cluster using your training account: See excercise 1 if you have closed your terminal.
- ▶ Get a list of modules that are currently loaded

*Hints:* [module list](#)

- ▶ Get a list of available R modules

*Hints:* [module av R](#)

## Excercise 7: Run an Rscript

- ▶ Connect to the cluster using your training account: See exercise 1 if you have closed your terminal.
- ▶ In the exercises folder you transferred earlier there is a file called test.r
- ▶ Run this script using: Rscript hello.r
- ▶ Load the module for: R/3.3.2

*Hints:* [module load R/3.3.2](#)

- ▶ Run the script again: Rscript hello.r
- ▶ What happens? what changes?

## Exercise 8: Install the R library locally

As a user you can create a local R library directory for packages that you want to install.

- ▶ Load an R module: `module load r-3.4.3-gcc-5.4.0-rbvhnge`
- ▶ Create a folder in your home for your own R package installs:  
`mkdir /my-R-libs`
- ▶ Make R aware of the new library location:  
`echo "R_LIBS_USER=~/my-R-libs" \textgreater .Renviron`
- ▶ Start R: `R`
- ▶ Display your library paths: `.libPaths()`
- ▶ Try loading a library: `require(pander)`
- ▶ Its not installed, lets install it: `install.packages("pander")`
- ▶ Try loading a library: `require(pander)`
- ▶ Library is now installed, lets quit R: `quit()`

## Excercise 8: Explained

- ▶ Our R modules: module load r/(version). We have two sets of R modules, those with an upper case R where compiled for an older version of Linux and should be ignored (Darwin legacy)
- ▶ echo " " outputs the text between the quotes, >redirects the text into the .Renviron file.
- ▶ When we start R the .Renviron file is read and R will now be aware of our local library directory.
- ▶ .libPaths() is how to check your library locations

## Exercise 9: Modules and Compilers

- ▶ Connect to the cluster using your training account: See exercise 1 if you have closed your terminal.
- ▶ Go to the `exercises` directory that you unzipped in hpcwork.  
Try to compile the `hello.c` program using the default `gcc` compiler (it will fail because there is a deliberate bug).

*Hints:* `gcc hello.c -o hello`

- ▶ To fix the problem, open the `hello.c` file in the `gedit` editor.  
*Hints:* Launch gedit in the background by doing `gedit&`. A gedit window should appear. Remove the word `BUG`, save the file and recompile. Do `./hello` to run the program.

- ▶ If you get this error:

WARNING \*\*: cannot open display:

then you have missed the '`-Y`' in your SSH command

## Exercise 9: Modules and Compilers

- ▶ The default version of gcc is 4.8.5. Compile hello.c again with [gcc 5.4.0](#).

*Hints:* module av, module load gcc-5.4.0-gcc-4.8.5-fis24gg, then  
[gcc hello.c -o hello2](#)

## **Part VI: HPC Job Submission**

# Using HPC: Job Submission



# Using HPC: Job Submission

- ▶ Compute resources are managed by a scheduler:  
[SLURM](#)/PBS/SGE/LSF/...
- ▶ Jobs are submitted to the scheduler
  - analogous to submitting jobs to a print queue
  - a file (*submission script*) is copied and queued for processing.

# Using HPC: Job Submission

- ▶ Jobs are submitted from the **login node**
  - not itself managed by the scheduler.
- ▶ Jobs may be either non-interactive (**batch**) or **interactive**.
- ▶ **Batch** jobs run a shell script on the first of a list of allocated nodes.
- ▶ **Interactive** jobs provide a command line on the first of a list of allocated nodes.

# Using HPC: Job Submission

- ▶ Jobs may use **part** or **all** of one or more nodes
  - the owner can specify --exclusive to force exclusive node access.
- ▶ Template submission scripts are available under  
[~/job\\_templates](#).

# Job Submission: Using SLURM

- ▶ Prepare a shell script and submit it to SLURM:

```
[abc123@login-a-1]$ sbatch slurm_submission_script  
Submitted batch job 790299
```

# Job Submission: Show Queue

- Submitted job scripts are copied and stored in a queue:

```
[abc123@login-a-1]$ squeue -u abc123
      JOBID PARTITION     NAME     USER ST          TIME  NODES NODELIST(REASON)
    790299    skylake   Test3  abc123 PD          0:00      2  (PriorityResourcesAssocGrpCPUMinsL
    790290    skylake   Test2  abc123 R  27:56:10      2 cpu-a-[1,10]
```

# Job Submission: Monitor Job

- ▶ Examine a particular job:

```
[abc123@login-a-1]$ scontrol show job=790290
```

## Job Submission: Cancel Job

- ▶ Cancel a particular job:

```
[abc123@login-a-1]$ scancel 790290
```

# Job Submission: Scripts

## ► SLURM

In `~/job_templates`, see examples: [slurm\\_submit.skylake.generic](#), [slurm\\_submit.skylake.matlab](#).

```
#!/bin/bash
#! Name of the job:
#SBATCH -J myjob
#! Which project should be charged:
#SBATCH -A TRAINING-CPU
#! How many whole nodes should be allocated?
#SBATCH --nodes=1
#! How many tasks will there be in total? (<= nodes*32)
#SBATCH --ntasks=116
#! How much wallclock time will be required?
#SBATCH --time=02:00:00
#! Select partition:
#SBATCH -p skylake
...
```

## ► **#SBATCH** lines are *structured comments*

— correspond to sbatch command line options.

► The above job will be given 1 **cpu16 cpus** on 1 node for 2 hours  
(by default there is 1 task per node, and 1 cpu per task).

# Job Submission: Accounting Commands

- ▶ How many core hours available do I have?

```
mybalance
```

User	Usage	Account	Usage	Account Limit	Available (hours)
sjr20	3	SUPPORT-CPU	2,929	22,425,600	22,422,671
sjr20	0	SUPPORT-GPU	0	87,600	87,600

- ▶ How many core hours does some other project or user have?

```
gbalance -p SUPPORT-CPU
```

User	Usage	Account	Usage	Account Limit	Available (hours)
pfb29	2,925	SUPPORT-CPU	2,929	22,425,600	22,422,671
sjr20 *	3	SUPPORT-CPU	2,929	22,425,600	22,422,671
...					

(Use -u for user.)

- ▶ List all jobs charged to a project/user between certain times:

JobID	User	Account	JobName	Partition	End	ExitCode	State	CompHrs
263	xyz10	support-c+ _interact+		skylake	2018-04-18T19:44:40	0:0	TIMEOUT	1.0
264	xyz10	support-c+ _interact+		skylake	2018-04-18T19:48:07	0:0	CANCELLED+	0.1
275	xyz10	support-c+ _interact+		skylake	Unknown	0:0	RUNNING	0.3
...								

# Job Submission: Single Node Jobs

- ▶ Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=
#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS= # For OpenMP across cores
$application $options
...
```

# Job Submission: Single Node Jobs

- ▶ Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=1
# Default is 1 cpu (core) per task
#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS= # For OpenMP across cores
$application $options
...
```

# Job Submission: Single Node Jobs

- ▶ Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=32 # Whole node

#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=32 # For OpenMP across 32 cores
$application $options
...
```

# Job Submission: Single Node Jobs

- ▶ Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=16 # Half node

#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=16 # For OpenMP across 16 cores
$application $options
...
```

# Job Submission: Single Node Jobs

- ▶ Serial jobs requiring large memory, or OpenMP codes.

```
#!/bin/bash
...
#SBATCH --nodes=1
#SBATCH --ntasks=1
# Default is 1 task per node
#SBATCH --cpus-per-task=32 # Whole node

#SBATCH --mem=5990
# Memory per node in MB - default is pro rata by cpu number
# Increasing --mem or --cpus-per-task implicitly increases the other
...
export OMP_NUM_THREADS=16 # For OpenMP across 16 cores (using all memory)
$application $options
...
```

# Job Submission: MPI Jobs

- ▶ Parallel job across multiple nodes.

```
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=128      # i.e. 32x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
mpirun -np 128 $application $options
...
```

- ▶ SLURM-aware MPI launches remote tasks via SLURM.
- ▶ The template script uses \$SLURM\_TASKS\_PER\_NODE to set PPN.

# Job Submission: MPI Jobs

- ▶ Parallel job across multiple nodes.

```
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=64      # i.e. 16x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
mpirun -ppn 16 -np 64 $application $options
...
```

- ▶ SLURM-aware MPI launches remote tasks via SLURM.
- ▶ The template script uses \$SLURM\_TASKS\_PER\_NODE to set PPN.

# Job Submission: Hybrid Jobs

- ▶ Parallel jobs using both MPI and OpenMP.

```
#!/bin/bash
...
#SBATCH --nodes=4
#SBATCH --ntasks=64      # i.e. 16x4 MPI tasks in total.
#SBATCH --cpus-per-task=2
...
export OMP_NUM_THREADS=2  # i.e. 2 threads per MPI task.
mpirun -ppn 16 -np 64 $application $options
...
```

- ▶ This job uses 128 CPUs (each MPI task splits into 2 OpenMP threads).

# Job Submission: High Throughput Jobs

- ▶ Multiple serial jobs across multiple nodes.
- ▶ Use `srun` to launch tasks (**job steps**) within a job.

```
#!/bin/bash
...
#SBATCH --nodes=2
...
cd directory_for_job1
srun --exclusive -N 1 -n 1 $application $options_for_job1 > output 2> err &
cd directory_for_job2
srun --exclusive -N 1 -n 1 $application $options_for_job2 > output 2> err &
...
cd directory_for_job64
srun --exclusive -N 1 -n 1 $application $options_for_job64 > output 2> err &
wait
```

## Exercise 10: Submitting a Matlab job

- ▶ Submit a job which will run `matlab` on the `file.m` command file (which contains just the `ver` command).

- Hints:*
1. Load the matlab module using the `job_script` in your exercises directory.
  2. Set the value of application to  
`"matlab -nodesktop -nosplash -nojvm"`
  3. Set the value of options to `"-r file"`
  4. Submit the job with `sbatch job_script`. The jobid is then printed.
  5. Watch the job in the queue with `squeue`.
  6. After it has disappeared, open the output file `slurm-jobid.out` in your editor. It should contain a list of licensed Matlab features.
  7. For more demanding work you can increase the available memory by increasing the number of cpus.

## Exercise 11: Submitting compiled code

- ▶ Submit a job which will run a copy of your hello program on 1 cpu.

*Hints:* 1. Edit the script `job_script` in your exercises directory.

Set:

```
#SBATCH --nodes=1  
#SBATCH --ntasks=1  
application=".//hello"
```

2. Submit the job with `sbatch job_script`. The jobid is then printed.
3. Watch the job in the queue with `squeue`.
4. After it has disappeared, open the output file `slurm-jobid.out` in your editor. There should be exactly one “Hello, World!” message.

# Job Submission: Interactive

- ▶ Compute nodes are accessible via SSH while you have a job running on them.
- ▶ Alternatively, submit an interactive job:  
`sintr -A TRAINING-CPU -N1 -n8 -t 2:0:0`
- ▶ Within the window (screen session):
  - \* Launches a shell on the first node (when the job starts).
  - \* Graphical applications should display correctly.
  - \* Create new shells with `ctrl-a c`, navigate with `ctrl-a n` and `ctrl-a p`.
  - \* `ssh` or `srun` can be used to start processes on any nodes in the job.
  - \* SLURM-aware MPI will do this automatically.

# Job Submission: Array Jobs

- ▶ [http://slurm.schedmd.com/job\\_array.html](http://slurm.schedmd.com/job_array.html)
- ▶ Used for submitting and managing large sets of similar jobs.
- ▶ Each job in the array has the same **initial** options.
- ▶ SLURM

```
[abc123@login-a-1]$ sbatch --array=1-7:21,3,5,7 -A TRAINING-CPU submit_script
Submitted batch job 791609
```

```
[abc123@login-a-1]$ squeue -u abc123
      JOBID PARTITION     NAME     USER ST      TIME  NODES NODELIST(REASON)
    791609_1 skylake     hpl    abc123 R      0:06      1 cpu-a-6
    791609_3 skylake     hpl    abc123 R      0:06      1 cpu-a-16
    791609_5 skylake     hpl    abc123 R      0:06      1 cpu-a-7
    791609_7 skylake     hpl    abc123 R      0:06      1 cpu-a-7
```

791609\_1, 791609\_3, 791609\_5, 791609\_7

i.e. \${SLURM\_ARRAY\_JOB\_ID} - \${SLURM\_ARRAY\_TASK\_ID}

SLURM\_ARRAY\_JOB\_ID = SLURM\_JOBID for the first element.

## Job Submission: Array Jobs (ctd)

- ▶ Updates can be applied to specific array elements using  `${SLURM_ARRAY_JOB_ID} . ${SLURM_ARRAY_TASK_ID}`
- ▶ Alternatively operate on the entire array via  `${SLURM_ARRAY_JOB_ID}`.
- ▶ Some commands still require the SLURM\_JOB\_ID (sacct, sreport, sshare, sstat and a few others).
- ▶ Exercise 7 - Array Jobs.

## Exercise 11: Array Jobs

- ▶ Submit your last job in the form of an array with indices 1-32. Use -H with sbatch to mark the array as held (so that it won't run immediately).

*Hints:*

1. Use `sbatch -H --array=1-32 job_script`
2. Use `squeue -u userid` to see your array job. Note that `-r` reports each array element individually.

- ▶ Release array element 1 and allow it to run. Then release the others.

*Hints:*

1. Use `scontrol release ${SLURM_ARRAY_JOB_ID}_1`
2. Use `squeue -u userid` again to watch what happens.
3. Release the others with  
`scontrol release ${SLURM_ARRAY_JOB_ID}`  
i.e. use the array id to release the entire array.
4. When all the jobs complete you should have 64 `slurm-${SLURM_ARRAY_JOB_ID}_N.out` files saying hello from various cpus on possibly multiple nodes.

# Scheduling

- ▶ SLURM scheduling is multifactor:
  - ▶ QoS — payer or non-payer?
  - ▶ Age — how long has the job waited?  
*Don't cancel jobs that seem to wait too long.*
  - ▶ Fair Share — how much recent usage?  
*Payers with little recent usage receive boost (not implemented yet).*
  - ▶ `sprio -j jobid`
- ▶ Backfilling
  - ▶ Promote lower priority jobs into gaps left by higher priority jobs.
  - ▶ Demands that the higher priority jobs not be delayed.
  - ▶ Relies on reasonably accurate wall time requests for this to work.
  - ▶ Jobs of default length will not backfill readily.

# Wait Times

- ▶ 36 hour job walltimes are permitted.
- ▶ This sets the timescale at busy times (*without* backfilling).
- ▶ Use backfilling when possible.
- ▶ Short (1 hour or less) jobs have higher throughput.

# Checkpointing

- ▶ Insurance against failures during long jobs.
- ▶ Restart from checkpoints to work around finite job length.
- ▶ Application native methods are best. Failing that ...

# Job Submission: Scheduling Top Dos & Don'ts

## ► Do ...

- ▶ Give reasonably accurate wall times (allows [backfilling](#)).
- ▶ Check your balance occasionally ([mybalance](#)).
- ▶ Test on a small scale first.
- ▶ Implement [checkpointing](#) if possible (reduces resource wastage).

## ► Don't ...

- ▶ Request more than you need
  - you will wait longer and use more credits.
- ▶ Cancel jobs unnecessarily
  - priority increases over time.