HPC Carpentry part 2 - High Performance Computing (./)

All the details of this workshop can be found in the site for the first section of the workshop: https://aniabrown.github.io/hpc-carpentry-shell-WHPC/ (https://aniabrown.github.io/hpc-carpentry-shell-WHPC/)

Schedule

	Setup (./setup/)	Download files required for the lesson
09:30	1. Day one follow up (./11-day1-questions/index.html)	
09:40	2. Working on a remote HPC system (./12-cluster/index.html)	How do I log on to a remote HPC system?
10:15	3. Scheduling jobs (./13-scheduler/index.html)	What is a scheduler and why are they used? How do I launch a program to run on any one node in the cluster? How do I capture the output of a program that is run on a node in the cluster?
11:30	4. Coffee Break (./131-coffee/index.html)	Break
12:00	5. Accessing software (./14-modules/index.html)	How do we load and unload software packages?
12:45	6. Transferring files (./15-transferring-files/index.html)	How do I upload/download files to the cluster?
13:25	7. Lunch (./151-lunch/index.html)	Break
14:25	8. Using resources effectively (./16-resources/index.html)	What resources should I ask for in my job script? How can I get my jobs scheduled more easily?
14:55	9. Scheduling multiple similar jobs (./161-job-arrays/index.html)	How do we launch many similar jobs? How can we monitor job arrays?
15:20	10. Using shared resources responsibly (./17-responsiblity/index.html)	How can I be a responsible user? How can I protect my data? How can I best get large amounts of data off an HPC system?
15:40	11. Coffee Break (./171-coffee/index.html)	Break
16:10	Finish	

The actual schedule may vary slightly depending on the topics and exercises chosen by the instructor.

Working on a remote HPC system

Overview

Teaching: 25 min
Exercises: 10 min
Questions

- How do I log on to a remote HPC system?

Objectives

- Connect to a remote HPC system.
- Understand the general HPC system architecture.

Logging in

To log into the cluster (Cirrus at EPCC, The University of Edinburgh) use:

```
[user@laptop ~]$ ssh yourUsername@login.cirrus.ac.uk
```

Remember to replace `yourUsername` with the username supplied by the instructors. You will be asked for your password. But watch out, the characters you type are not displayed on the screen.

You are logging in using a program known as the secure shell or `ssh`. This establishes a temporary encrypted connection between your laptop and `login.cirrus.ac.uk`. The word before the `@` symbol, e.g. `yourUsername` here, is the user account name that Lola has access permissions for on the cluster.

Where do I get this `ssh` from ?

On Linux and/or macOS, the `ssh` command line utility is almost always pre-installed. Open a terminal and type `ssh --help` to check if that is the case.

At the time of writing, the easiest way to get access to ssh is to install the git bash program. See setup (https://aniabrown.github.io/hpc-carpentry-shell-WHPC/setup/) for more details.

Where are we?

Very often, many users are tempted to think of a high-performance computing installation as one giant, magical machine. Sometimes, people will assume that the computer they've logged onto is the entire computing cluster. So what's really happening? What computer have we logged on to? The name of the current computer we are logged onto can be checked with the `hostname` command. (You may also notice that the current hostname is also part of our prompt!)

```
[yourUsername@cirrus-login0 ~]$ hostname
```

```
cirrus-login0
```

Nodes

Individual computers that compose a cluster are typically called *nodes* (although you will also hear people call them *servers*, *computers* and *machines*). On a cluster, there are different types of nodes for different types of tasks. The node where you are right now is called the *head node*, *login node* or *submit node*. A login node serves as an access point to the cluster. As a gateway, it is well suited for uploading and downloading files, setting up software, and running quick tests. It should never be used for doing actual work.

The real work on a cluster gets done by the *worker* (or *compute*) *nodes*. Worker nodes come in many shapes and sizes, but generally are dedicated to long or hard tasks that require a lot of computational resources.

All interaction with the worker nodes is handled by a specialized piece of software called a scheduler (the scheduler used in this lesson is called). We'll learn more about how to use the scheduler to submit jobs next, but for now, it can also tell us more information about the worker nodes.

For example, we can view all of the worker nodes with the `pbsnodes -a` command.

```
[yourUsername@cirrus-login0 ~]$ pbsnodes -a
```

```
r1i0n32
     Mom = r1i0n32.ib0.icexa.epcc.ed.ac.uk
     ntype = PBS
     state = offline
     pcpus = 72
     resources_available.arch = linux
     resources_available.host = r1i0n32
     resources_available.mem = 263773892kb
     resources_available.ncpus = 36
     resources_available.vnode = r1i0n32
     resources_assigned.accelerator_memory = 0kb
     resources_assigned.mem = 0kb
     resources_assigned.naccelerators = 0
     resources_assigned.ncpus = 0
     resources_assigned.netwins = 0
     resources_assigned.vmem = 0kb
     resv_enable = True
     sharing = default_shared
     license = l

...
```

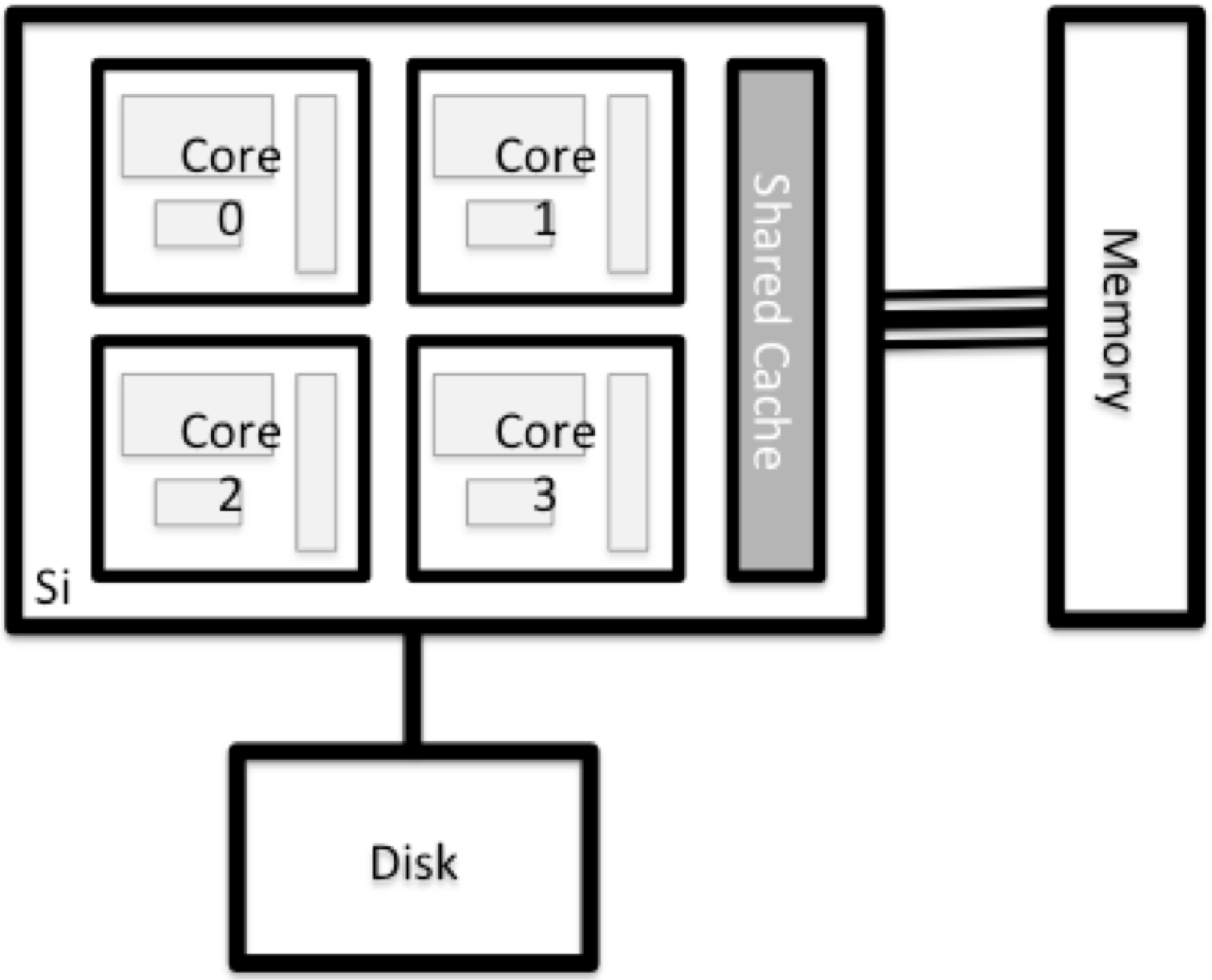
There are also specialized machines used for managing disk storage, user authentication, and other infrastructure-related tasks. Although we do not typically logon to or interact with these machines directly, they enable a number of key features like ensuring our user account and files are available throughout the HPC system.

Shared file systems

This is an important point to remember: files saved on one node (computer) are often available everywhere on the cluster!

What's in a node?

All of a HPC system's nodes have the same components as your own laptop or desktop: *CPUs* (sometimes also called *processors* or *cores*), *memory* (or *RAM*), and *disk* space. CPUs are a computer's tool for actually running programs and calculations. Information about a current task is stored in the computer's memory. Disk refers to all storage that can be accessed like a file system. This is generally storage that can hold data permanently, i.e. data is still there even if the computer has been restarted.



Explore Your Computer

Try to find out the number of CPUs and amount of memory available on your personal computer.

Explore The Head Node

Now we'll compare the size of your computer with the size of the head node: To see the number of processors, run:

```
[yourUsername@cirrus-login0 ~]$ nproc --all
```

How about memory? Try running:

```
[yourUsername@cirrus-login0 ~]$ free -h
```

Getting more information

You can get more detailed information on both the processors and memory by using different commands.

For more information on processors use `lscpu`

```
[yourUsername@cirrus-login0 ~]$ lscpu
```

For more information on memory you can look in the `/proc/meminfo` file:

```
[yourUsername@cirrus-login0 ~]$ cat /proc/meminfo
```

Explore a Worker Node

Finally, let's look at the resources available on the worker nodes where your jobs will actually run. Try running this command to see the name, CPUs and memory available on the worker nodes:

```
[yourUsername@cirrus-login0 ~]$ pbsnodes r1i0n32
```

Compare Your Computer, the Head Node and the Worker Node

Compare your laptop's number of processors and memory with the numbers you see on the cluster head node and worker node. Discuss the differences with your neighbor. What implications do you think the differences might have on running your research work on the different systems and nodes?

Units and Language

A computer's memory and disk are measured in units called *Bytes* (one Byte is 8 bits). As today's files and memory have grown to be large given historic standards, volumes are noted using the SI (https://en.wikipedia.org/wiki/International_System_of_Units) prefixes. So 1000 Bytes is a Kilobyte (kB), 1000 Kilobytes is a Megabyte, 1000 Megabytes is a Gigabyte etc.

History and common language have however mixed this notation with a different meaning. When people say "Kilobyte", they mean 1024 Bytes instead. In that spirit, a Megabyte are 1024 Kilobytes. To address this ambiguity, the International System of Quantities (https://en.wikipedia.org/wiki/International_System_of_Quantities) standardizes the binary prefixes (with base of 1024) by the prefixes kibi, mibi, gibi, etc. For more details, see here (https://en.wikipedia.org/wiki/Binary_prefix)

Differences Between Nodes

Many HPC clusters have a variety of nodes optimized for particular workloads. Some nodes may have larger amount of memory, or specialized resources such as Graphical Processing Units (GPUs).

With all of this in mind, we will now cover how to talk to the cluster's scheduler, and use it to start running our scripts and programs!

Key Points

- HPC systems typically provides login nodes and a set of worker nodes.
- The resources found on independent (worker) nodes can vary in volume and type (amount of RAM, processor architecture, availability of network mounted file systems, etc.).
- Files saved on one node are available on all nodes.

(../11-day1-questions/index.html)

(../13-schec

Scheduling jobs

❓ Overview

Teaching: 45 min
Exercises: 30 min
Questions

- What is a scheduler and why are they used?
- How do I launch a program to run on any one node in the cluster?
- How do I capture the output of a program that is run on a node in the cluster?

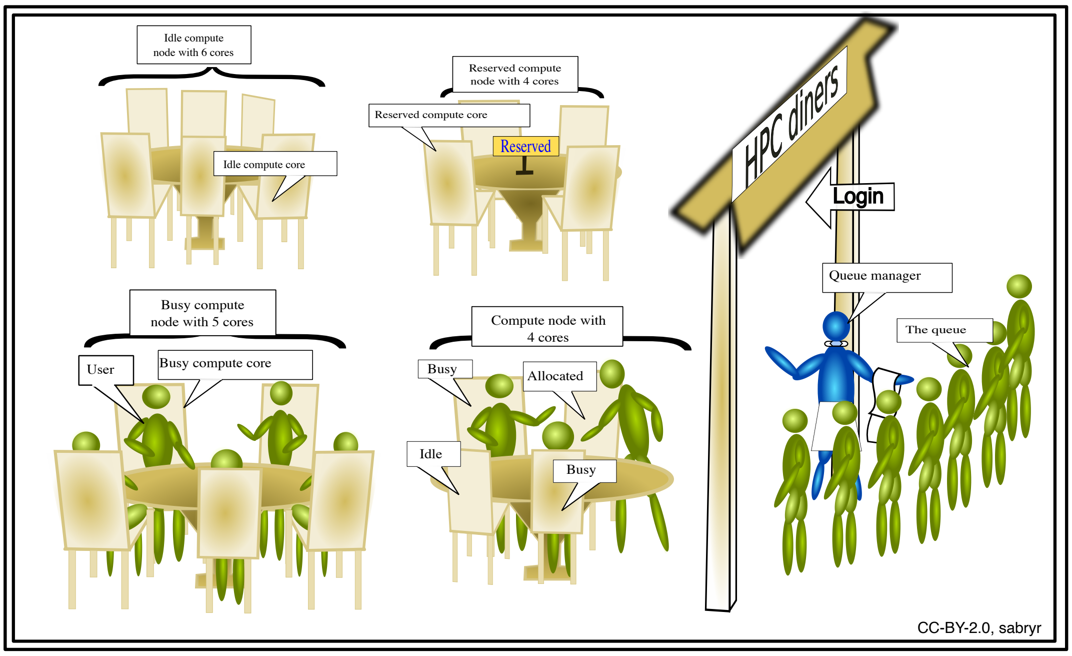
Objectives

- Run a simple Hello World style program on the cluster.
- Submit a simple Hello World style script to the cluster.
- Use the batch system command line tools to monitor the execution of your job.
- Inspect the output and error files of your jobs.

Job scheduler

An HPC system might have thousands of nodes and thousands of users. How do we decide who gets what and when? How do we ensure that a task is run with the resources it needs? This job is handled by a special piece of software called the scheduler. On an HPC system, the scheduler manages which jobs run where and when.

The following illustration compares these tasks of a job scheduler to a waiter in a restaurant. If you can relate to an instance where you had to wait for a while in a queue to get in to a popular restaurant, then you may now understand why sometimes your job do not start instantly as in your laptop.



()

Job scheduling roleplay (optional)

Your instructor will divide you into groups taking on different roles in the cluster (users, compute nodes and the scheduler). Follow their instructions as they lead you through this exercise. You will be emulating how a job scheduling system works on the cluster.

notes for the instructor here (../guide)

The scheduler used in this lesson is PBS Pro. Although PBS Pro is not used everywhere, running jobs is quite similar regardless of what software is being used. The exact syntax might change, but the concepts remain the same.

Running a batch job

The most basic use of the scheduler is to run a command non-interactively. Any command (or series of commands) that you want to run on the cluster is called a *job*, and the process of using a scheduler to run the job is called *batch job submission*.

In this case, the job we want to run is just a shell script. Let's create a demo shell script to run as a test.

Creating our test job

Using your favorite text editor, create the following script and run it. Does it run on the cluster or just our login node?

```
#!/bin/bash

echo 'This script is running on:'
hostname
sleep 60
```

If you completed the previous challenge successfully, you probably realise that there is a distinction between running the job through the scheduler and just "running it". To submit this job to the scheduler, we use the `qsub` command.

```
[[yourUsername@cirrus-login0 ~]$ qsub -A tc008 -q R1262266 example-job.sh
```

```
387775.indy2-login0
```

And that's all we need to do to submit a job. Our work is done – now the scheduler takes over and tries to run the job for us. While the job is waiting to run, it goes into a list of jobs called the *queue*. To check on our job's status, we check the queue using the command `qstat -u yourUsername`.

```
[yourUsername@cirrus-login0 ~]$ qstat -u yourUsername
```

```

indy2-login0:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
387775.indy2-lo yourUser workq    example-jo  50804   1   1    --  96:00 R 00:00
```

We can see all the details of our job, most importantly that it is in the "R" or "RUNNING" state. Sometimes our jobs might need to wait in a queue ("PENDING") or have an error. The best way to check our job's status is with `qstat`. Of course, running `qstat` repeatedly to check on things can be a little tiresome. To see a real-time view of our jobs, we can use the `watch` command. `watch` reruns a given command at 2-second intervals. This is too frequent, and will likely upset your system administrator. You can change the interval to a more reasonable value, for example 60 seconds, with the `-n 60` parameter. Let's try using it to monitor another job.

```
[yourUsername@cirrus-login0 ~]$ qsub -A tc008 -q R1262266 example-job.sh
[yourUsername@cirrus-login0 ~]$ watch -n 60 qstat -u yourUsername
```

You should see an auto-updating display of your job's status. When it finishes, it will disappear from the queue. Press `Ctrl-C` when you want to stop the `watch` command.

Customising a job

The job we just ran used all of the scheduler's default options. In a real-world scenario, that's probably not what we want. The default options represent a reasonable minimum. Chances are, we will need more cores, more memory, more time, among other special considerations. To get access to these resources we must customize our job script.

Comments in UNIX (denoted by `#`) are typically ignored. But there are exceptions. For instance the special `#!` comment at the beginning of scripts specifies what program should be used to run it (typically `/bin/bash`). Schedulers like PBS Pro also have a special comment used to denote special scheduler-specific options. Though these comments differ from scheduler to scheduler, PBS Pro's special comment is `#PBS`. Anything following the `#PBS` comment is interpreted as an instruction to the scheduler.

Let's illustrate this by example. By default, a job's name is the name of the script, but the `-N` option can be used to change the name of a job.

Submit the following job (`qsub -A tc008 -q R1262266 example-job.sh`):

```
#!/bin/bash
#PBS -N new_name

echo 'This script is running on:'
hostname
sleep 120
```

```
[yourUsername@cirrus-login0 ~]$ qstat -u yourUsername
```

```

indy2-login0:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
387778.indy2-lo yourUser workq    new_name    51536   1   1    --  96:00 R 00:00

```

Fantastic, we've successfully changed the name of our job!

Resource requests

But what about more important changes, such as the number of cores and memory for our jobs? One thing that is absolutely critical when working on an HPC system is specifying the resources required to run a job. This allows the scheduler to find the right time and place to schedule our job. If you do not specify requirements (such as the amount of time you need), you will likely be stuck with your site's default resources, which is probably not what we want.

The following are several key resource requests:

- `-l select=<nnodes>:ncpus=<ncores per node>` - how many nodes does your job need and how many cores per node? Note that there are 36 cores per node on Cirrus.
- `-l walltime=<hours:minutes:seconds>` - How much real-world time (walltime) will your job take to run?
- `-l place=scatter:excl` - Reserve your nodes just for yourself. (If you are using full nodes, you should include this as it stops other users from interfering with the performance of your job.)

Note that just *requesting* these resources does not make your job run faster! We'll talk more about how to make sure that you're using resources effectively in a later episode of this lesson.

Submitting resource requests

Submit a job that will use 1 full node and 2 minutes of walltime.

Job environment variables

When PBS Pro runs a job, it sets a number of environment variables for the job. One of these will let us check what directory our job script was submitted from. The `PBS_O_WORKDIR` variable is set to the directory from which our job was submitted. Using the `PBS_O_WORKDIR` variable, modify your job so that it prints (to stdout) the location from which the job was submitted.

Resource requests are typically binding. If you exceed them, your job will be killed. Let's use walltime as an example. We will request 30 seconds of walltime, and attempt to run a job for one minute.

```
#!/bin/bash
#PBS -l walltime=0:0:30

echo 'This script is running on:'
hostname
sleep 60
```

Submit the job and wait for it to finish. Once it is has finished, check the log file.

```
[yourUsername@cirrus-login0 ~]$ qsub -A tc008 -q R1262266 example-job.sh
[yourUsername@cirrus-login0 ~]$ watch -n 60 qstat -u yourUsername
[yourUsername@cirrus-login0 ~]$ cat example-job.sh.e387798
```

```
=>> PBS: job killed: walltime 33 exceeded limit 30
```

Our job was killed for exceeding the amount of resources it requested. Although this appears harsh, this is actually a feature. Strict adherence to resource requests allows the scheduler to find the best possible place for your jobs. Even more importantly, it ensures that another user cannot use more resources than they've been given. If another user messes up and accidentally attempts to use all of the cores or memory on a node, PBS Pro will either restrain their job to the requested resources or kill the job outright. Other jobs on the node will be unaffected. This means that one user cannot mess up the experience of others, the only jobs affected by a mistake in scheduling will be their own.

Cancelling a job

Sometimes we'll make a mistake and need to cancel a job. This can be done with the `qdel` command. Let's submit a job and then cancel it using its job number (remember to change the walltime so that it runs long enough for you to cancel it before it is killed!).

```
[yourUsername@cirrus-login0 ~]$ qsub -A tc008 -q R1262266 example-job.sh
[yourUsername@cirrus-login0 ~]$ qstat -u yourUsername
```

```
38759.indy2-login0

indy2-login0:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
38759.indy2-log yourUser workq    example-jo  32085   1   1    --  00:10 R 00:00
```

Now cancel the job with it's job number. Absence of any job info indicates that the job has been successfully cancelled.

```
[yourUsername@cirrus-login0 ~]$ qdel 38759

... Note that it might take a minute for the job to disappear from the queue ...
[yourUsername@cirrus-login0 ~]$ qstat -u yourUsername
```

```
...(no output from qstat when there are no jobs to display)...
```

Other types of jobs

Up to this point, we've focused on running jobs in batch mode. PBS Pro also provides the ability to start an interactive session.

There are very frequently tasks that need to be done interactively. Creating an entire job script might be overkill, but the amount of resources required is too much for a login node to handle. A good example of this might be building a genome index for alignment with a tool like HISAT2 (https://ccb.jhu.edu/software/hisat2/index.shtml). Fortunately, we can run these types of tasks as a one-off with `qsub` .

qsub (with the right options) can submit a job and then wait for it to start so we can use the compute node resources interactively. Let's demonstrate this by submitting an interactive job that uses a single core:

```
[yourUsername@cirrus-login0 ~]$ qsub -IVl select=1:ncpus=1 -A tc008 -q R1262266
```

You should be presented with a bash prompt. Note that the prompt will likely change to reflect your new location, in this case the worker node we are logged on. You can also verify this with `hostname` .

When you are done with the interactive job, type exit to quit your session.

Key Points

- The scheduler handles how compute resources are shared between users.
- Everything you do should be run through the scheduler.
- A job is just a shell script.
- If in doubt, request more resources than you will need.

‹
(../12-cluster/index.html)

›
(../13
coffe

Accessing software

Overview

Teaching: 30 min
Exercises: 15 min
Questions

- How do we load and unload software packages?

Objectives

- Understand how to load and use a software package.

On a high-performance computing system, it is often the case that no software is loaded by default. If we want to use a software package, we will need to "load" it ourselves.

Before we start using individual software packages, however, we should understand the reasoning behind this approach. The three biggest factors are:

- software incompatibilities;
- versioning;
- dependencies.

Software incompatibility is a major headache for programmers. Sometimes the presence (or absence) of a software package will break others that depend on it. Two of the most famous examples are Python 2 and 3 and C compiler versions. Python 3 famously provides a `python` command that conflicts with that provided by Python 2. Software compiled against a newer version of the C libraries and then used when they are not present will result in a nasty `'GLIBCXX_3.4.20' not found` error, for instance.

Software versioning is another common issue. A team might depend on a certain package version for their research project - if the software version was to change (for instance, if a package was updated), it might affect their results. Having access to multiple software versions allow a set of researchers to prevent software versioning issues from affecting their results.

Dependencies are where a particular software package (or even a particular version) depends on having access to another software package (or even a particular version of another software package). For example, the VASP materials science software may depend on having a particular version of the FFTW (Fastest Fourer Transform in the West) software library available for it to work.

Environment modules

Environment modules are the solution to these problems. A *module* is a self-contained description of a software package - it contains the settings required to run a software packace and, usually, encodes required dependencies on other software packages.

The `module` command is used to interact with environment modules. An additional subcommand is usually added to the command to specify what you want to do. For a list of subcommands you can use `module -h` or `module help`. As for all commands, you can access the full help on the *man* pages with `man module`.

On login you may start out with a default set of modules loaded or you may start out with an empty environment, this depends on the setup of the system you are using.

Listing currently loaded modules

You can use the `module list` command to see which modules you currently have loaded in your environment. If you have no modules loaded, you will see a message telling you so

```
[yourUsername@cirrus-login0 ~]$ module list
```

```
No Modulefiles Currently Loaded.
```

Listing available modules

To see available software modules, use `module avail`

```
[yourUsername@cirrus-login0 ~]$ module avail
```

```
---------------------------------------------- /usr/share/Modules/modulefiles -----------------------
-----------------------
dot         module-git  module-info modules     mpt/2.16    null        perfboost   perfcatcher use.o
wn

-------------------------------------------------- /lustre/sw/modulefiles ---------------------------
-----------------------
abinit/8.2.3-intel17-mpt214(default)    gdal/2.1.2-gcc6                         ipm/2.0.6-impi
allinea/7.0.0(default)                  gnu-parallel/20170322(default)          lammps/31Mar2017-gcc6
-mpt214(default)
altair-hwsolvers/13.0.213               gnuplot/5.0.5(default)                  matlab/R2016b(defaul
t)
altair-hwsolvers/14.0.210               gnuplot/5.0.5-x11                       matlab/R2018a
amber/16                                gromacs/2016.3(default)                 meqtrees/1.5.1(defaul
t)
anaconda/python2(default)               gromacs/2018.3                          mercurial/3.9.1(defau
lt)
anaconda/python3                        gsl/2.5-gcc6(default)                   miniconda/python2
ansys/17.2                              hdf5parallel/1.10.1-gcc6-mpt214         miniconda/python3
ansys/18.0                              hdf5parallel/1.10.1-intel17-intel-mpi   molpro/2012.1.22(defa
ult)
ansys/19.0                              hdf5parallel/1.10.1-intel17-mpt214      mpt/2.14
blast/2.7.1+(default)                   intel-cc-16/16.0.2.181                  mpt/2.17
casa/5.4.0(default)                     intel-cc-16/16.0.3.210(default)         namd/2.12(default)
casacore/2.4.1(default)                 intel-cc-17/17.0.2.174(default)         nano/2.6.3
castep/16.11                            intel-cc-18/18.0.5.274                  ncl/6.4.0
castep/18.1.0-intel17(default)          intel-cmkl-16/16.0.2.181                nco/4.6.9
cmake/3.10.0(default)                   intel-cmkl-16/16.0.3.210(default)       ncview/2.1.7
cp2k/4.1                                intel-cmkl-17/17.0.2.174(default)       netcdf/4.4.1
cp2k-mpt/4.1                            intel-cmkl-18/18.0.5.274                netcdf-parallel/4.5.0
cuda/9.0                                intel-compilers-16/16.0.2.181           netcdf-parallel/4.5.0
-gcc6-mpt214
cuda/9.1(default)                       intel-compilers-16/16.0.3.210(default)  netcdf-parallel/4.5.0
-intel17
dolfin/2017.1.0(default)                intel-compilers-17/17.0.2.174(default)  netcdf-parallel/4.5.0
-intel17-mpt214
dolfin/2017.1.0-python-2.7              intel-compilers-18/18.05.274            openfoam/foundation/
5.0
dolfin/2017.2.0                         intel-fc-16/16.0.2.181                  openfoam/v1706
dolfin/2017.2.0-intel-mpi               intel-fc-16/16.0.3.210(default)         openfoam/v1712
dolfin/2017.2.0-mpt                     intel-fc-17/17.0.2.174(default)         petsc/3.8.4-intel-mpi
dolfin/2018.1.0-intel-mpi               intel-fc-18/18.0.5.274                  petsc/3.8.4-mpt
doxygen/1.8.14(default)                 intel-itac/9.1.2.024                    qe/6.1(default)
eclipse/4.2                             intel-itac-17/2017.2.028(default)       qe/6.1+d3q
epcc-tools/1.1(default)                 intel-itac-18/2018.5.025                R/3.4.0(default)
fenics/2016.2.0(default)                intel-mpi-16/16.0.3.210                 scalasca/2.3.1-gcc6
fenics/2017.2.0                         intel-mpi-17/17.0.2.174                 scalasca/2.3.1-intel1
7
fenics/2017.2.0-intel-mpi               intel-mpi-18/18.0.5.274                 singularity/2.4(defau
lt)
fenics/2017.2.0-mpt                     intel-tbb/16.0.2.181                    sionlib/1.7.2-gcc6-mp
t214
fenics/2018.1.0-intel-mpi               intel-tbb/16.0.3.210(default)           spack/20161205
fftw/3.3.5-gcc6                         intel-tbb-17/17.0.2.174(default)        spack/cirrus(default)
fftw/3.3.5-intel17                      intel-tbb-18/18.0.5.274                 spark/2.1.1(default)
flacs/10.5.1                            intel-tools-16/16.0.2.181               starccm+/12.04.011(de
fault)
flacs/10.6.3                            intel-tools-16/16.0.3.210(default)      starccm+/12.04.011-R8
gaussian/09.E01                         intel-tools-17/17.0.2.174(default)      szip/2.1.1
gaussian/16.A03(default)                intel-tools-18/18.0.5.274               testing/qe/6.1-intel
gcc/6.2.0                               intel-vtune-16/2016.2.0.444464          tinker/8.2.1
gcc/6.3.0(default)                      intel-vtune-16/2016.3.0.463186(default) valgrind/3.11.0
gcc/7.2.0                               intel-vtune-17/2017.2.0.499904(default) vasp/5.4.4-intel17-mp
t214(default)
gcc/8.2.0                               intel-vtune-18/2018.4.0.573462          xflow/98.00
```

Loading and unloading software

To load a software module, use `module load`. In this example we will use Python 3.

Initially, Python 3 is not loaded. We can test this by using the `which` command. `which` looks for programs the same way that Bash does, so we can use it to tell us where a particular piece of software is stored.

```
[yourUsername@cirrus-login0 ~]$ which python3
```

```
/usr/bin/which: no python3 in (/lustre/home/z04/aturner/miniconda2/bin:/opt/sgi/sbin:/opt/sgi/bin:/us
r/lib64/qt-3.3/bin:/opt/pbs/default/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/c3/bi
n:/sbin:/bin:/lustre/home/z04/aturner/bin)
```

We can load the `python3` command with `module load`:

```
[yourUsername@cirrus-login0 ~]$ module load anaconda/python3
[yourUsername@cirrus-login0 ~]$ which python3
```

```
/lustre/sw/anaconda/anaconda3-5.1.0/bin/python3
```

So, what just happened?

To understand the output, first we need to understand the nature of the `$PATH` environment variable. `$PATH` is a special environment variable that controls where a UNIX system looks for software. Specifically `$PATH` is a list of directories (separated by `:`) that the OS searches through for a command before giving up and telling us it can't find it. As with all environment variables we can print it out using `echo`.

```
[yourUsername@cirrus-login0 ~]$ echo $PATH
```

```
/lustre/home/z04/aturner/miniconda2/bin:/opt/sgi/sbin:/opt/sgi/bin:/usr/lib64/qt-3.3/bin:/opt/pbs/def
ault/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/c3/bin:/sbin:/bin:/lustre/home/z04/at
urner/bin
```

You'll notice a similarity to the output of the `which` command. In this case, there's only one difference: the different directory at the beginning. When we ran the `module load` command, it added a directory to the beginning of our `$PATH`. Let's examine what's there:

```
[yourUsername@cirrus-login0 ~]$ ls /lustre/sw/anaconda/anaconda3-5.1.0/bin
```

```
[output truncated]

conda-convert                    gio-querymodules       jupyter-run              python
tiff2rgba
conda-develop                    glacier                jupyter-serverextension  python3
tiffcmp
conda-env                        glib-compile-resources jupyter-troubleshoot     python3.6
tiffcp
conda-index                      glib-compile-schemas   jupyter-trust            python3.6-confi
g      tiffcrop
conda-inspect                    glib-genmarshal        kill_instance            python3.6m
tiffdither
conda-metapackage                glib-gettextize        launch_instance          python3.6m-conf
ig     tiffdump
conda-render                     glib-mkenums           lconvert                 python3-config
tiffinfo
conda-server                     gobject-query          libpng16-config          pyuic5
tiffmedian
conda-skeleton                   gresource              libpng-config            pyvenv
tiffset
elbadmin                         hb-view                patchelf                 rst2man.py

[output truncated]
```

Taking this to it’s conclusion, `module load` will add software to your `$PATH`. It “loads” software. A special note on this - depending on which version of the `module` program that is installed at your site, `module load` will also load required software dependencies.

To demonstrate, let’s load the `gromacs` module and then use the `module list` command to show which modules we currently have loaded in our environment. (Gromacs (http://www.gromacs.org/About_Gromacs) is a molecular dynamics modelling software package.)

```
[yourUsername@cirrus-login0 ~]$ module load gromacs
[yourUsername@cirrus-login0 ~]$ module list
```

```
Currently Loaded Modulefiles:
  1) anaconda/python3   3) mpt/2.18
  2) gcc/6.3.0          4) gromacs/2018.3
```

So in this case, loading the `gromacs` module also loaded a variety of other modules. Let’s try unloading the `gromacs` package.

```
[yourUsername@cirrus-login0 ~]$ module unload gromacs
[yourUsername@cirrus-login0 ~]$ module list
```

```
Currently Loaded Modulefiles:
  1) anaconda/python3
```

So using `module unload` “un-loads” a module along with its dependencies. If we wanted to unload everything at once, we could run `module purge` (unloads everything).

```
[yourUsername@cirrus-login0 ~]$ module load gromacs
[yourUsername@cirrus-login0 ~]$ module purge
```

```
No Modulefiles Currently Loaded.
```

Note that `module purge` has removed the `anaconda/python3` module as well as `gromacs` and its dependencies.

Software versioning

So far, we've learned how to load and unload software packages. This is very useful. However, we have not yet addressed the issue of software versioning. At some point or other, you will run into issues where only one particular version of some software will be suitable. Perhaps a key bugfix only happened in a certain version, or version X broke compatibility with a file format you use. In either of these example cases, it helps to be very specific about what software is loaded.

Let's examine the output of `module avail` more closely.

```
[yourUsername@cirrus-login0 ~]$ module avail
```

```
---------------------------------------------- /usr/share/Modules/modulefiles -----------------------
-----------------------
dot         module-git  module-info modules     mpt/2.16    null        perfboost   perfcatcher use.o
wn

-------------------------------------------------- /lustre/sw/modulefiles ---------------------------
-----------------------
abinit/8.2.3-intel17-mpt214(default)    gdal/2.1.2-gcc6                         ipm/2.0.6-impi
allinea/7.0.0(default)                  gnu-parallel/20170322(default)          lammps/31Mar2017-gcc6
-mpt214(default)
altair-hwsolvers/13.0.213               gnuplot/5.0.5(default)                  matlab/R2016b(defaul
t)
altair-hwsolvers/14.0.210               gnuplot/5.0.5-x11                       matlab/R2018a
amber/16                                gromacs/2016.3(default)                 meqtrees/1.5.1(defaul
t)
anaconda/python2(default)               gromacs/2018.3                          mercurial/3.9.1(defau
lt)
anaconda/python3                        gsl/2.5-gcc6(default)                   miniconda/python2
ansys/17.2                              hdf5parallel/1.10.1-gcc6-mpt214         miniconda/python3
ansys/18.0                              hdf5parallel/1.10.1-intel17-intel-mpi   molpro/2012.1.22(defa
ult)
ansys/19.0                              hdf5parallel/1.10.1-intel17-mpt214      mpt/2.14
blast/2.7.1+(default)                   intel-cc-16/16.0.2.181                  mpt/2.17
casa/5.4.0(default)                     intel-cc-16/16.0.3.210(default)         namd/2.12(default)
casacore/2.4.1(default)                 intel-cc-17/17.0.2.174(default)         nano/2.6.3
castep/16.11                            intel-cc-18/18.0.5.274                  ncl/6.4.0
castep/18.1.0-intel17(default)          intel-cmkl-16/16.0.2.181                nco/4.6.9
cmake/3.10.0(default)                   intel-cmkl-16/16.0.3.210(default)       ncview/2.1.7
cp2k/4.1                                intel-cmkl-17/17.0.2.174(default)       netcdf/4.4.1
cp2k-mpt/4.1                            intel-cmkl-18/18.0.5.274                netcdf-parallel/4.5.0
cuda/9.0                                intel-compilers-16/16.0.2.181           netcdf-parallel/4.5.0
-gcc6-mpt214
cuda/9.1(default)                       intel-compilers-16/16.0.3.210(default)  netcdf-parallel/4.5.0
-intel17
dolfin/2017.1.0(default)                intel-compilers-17/17.0.2.174(default)  netcdf-parallel/4.5.0
-intel17-mpt214
dolfin/2017.1.0-python-2.7              intel-compilers-18/18.05.274            openfoam/foundation/
5.0
dolfin/2017.2.0                         intel-fc-16/16.0.2.181                  openfoam/v1706
dolfin/2017.2.0-intel-mpi               intel-fc-16/16.0.3.210(default)         openfoam/v1712
dolfin/2017.2.0-mpt                     intel-fc-17/17.0.2.174(default)         petsc/3.8.4-intel-mpi
dolfin/2018.1.0-intel-mpi               intel-fc-18/18.0.5.274                  petsc/3.8.4-mpt
doxygen/1.8.14(default)                 intel-itac/9.1.2.024                    qe/6.1(default)
eclipse/4.2                             intel-itac-17/2017.2.028(default)       qe/6.1+d3q
epcc-tools/1.1(default)                 intel-itac-18/2018.5.025                R/3.4.0(default)
fenics/2016.2.0(default)                intel-mpi-16/16.0.3.210                 scalasca/2.3.1-gcc6
fenics/2017.2.0                         intel-mpi-17/17.0.2.174                 scalasca/2.3.1-intel1
7
fenics/2017.2.0-intel-mpi               intel-mpi-18/18.0.5.274                 singularity/2.4(defau
lt)
fenics/2017.2.0-mpt                     intel-tbb/16.0.2.181                    sionlib/1.7.2-gcc6-mp
t214
fenics/2018.1.0-intel-mpi               intel-tbb/16.0.3.210(default)           spack/20161205
fftw/3.3.5-gcc6                         intel-tbb-17/17.0.2.174(default)        spack/cirrus(default)
fftw/3.3.5-intel17                      intel-tbb-18/18.0.5.274                 spark/2.1.1(default)
flacs/10.5.1                            intel-tools-16/16.0.2.181               starccm+/12.04.011(de
fault)
flacs/10.6.3                            intel-tools-16/16.0.3.210(default)      starccm+/12.04.011-R8
gaussian/09.E01                         intel-tools-17/17.0.2.174(default)      szip/2.1.1
gaussian/16.A03(default)                intel-tools-18/18.0.5.274               testing/qe/6.1-intel
gcc/6.2.0                               intel-vtune-16/2016.2.0.444464          tinker/8.2.1
gcc/6.3.0(default)                      intel-vtune-16/2016.3.0.463186(default) valgrind/3.11.0
gcc/7.2.0                               intel-vtune-17/2017.2.0.499904(default) vasp/5.4.4-intel17-mp
t214(default)
gcc/8.2.0                               intel-vtune-18/2018.4.0.573462          xflow/98.00
```

Let's take a closer look at the `gcc` module. GCC is an extremely widely used C/C++/Fortran compiler. Lots of software is dependent on the GCC version, and might not compile or run if the wrong version is loaded. In this case, there are three different versions: `gcc/6.2.0` , `gcc/6.3.0` and `gcc/7.2.0` . How do we load each copy and which copy is the default?

In this case, `gcc/6.3.0` has a `(default)` next to it. This indicates that it is the default - if we type `module load gcc` , this is the copy that will be loaded.

```
[yourUsername@cirrus-login0 ~]$ module load gcc
[yourUsername@cirrus-login0 ~]$ gcc --version
```

```
gcc (GCC) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

So how do we load the non-default copy of a software package? In this case, the only change we need to make is be more specific about the module we are loading. There are three GCC modules: `gcc/6.2.0` , `gcc/6.3.0` and `gcc/7.2.0` To load a non-default module, we need to make add the version number after the `/` in our `module load` command

```
[yourUsername@cirrus-login0 ~]$ module load gcc/7.2.0
```

```
gcc/7.2.0(17):ERROR:150: Module 'gcc/7.2.0' conflicts with the currently loaded module(s) 'gcc/6.3.0'
gcc/7.2.0(17):ERROR:102: Tcl command execution failed: conflict gcc
```

What happened? The module command is telling us that we cannot have two `gcc` modules loaded at the same time as this could cause confusion about which version you are using. We need to remove the default version before we load the new version.

```
[yourUsername@cirrus-login0 ~]$ module unload gcc
[yourUsername@cirrus-login0 ~]$ module load gcc/7.2.0
[yourUsername@cirrus-login0 ~]$ gcc --version
```

```
gcc (GCC) 7.2.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

We now have successfully switched from GCC 6.3.0 to GCC 7.2.0.

As switching between different versions of the same module is often used you can use `module swap` rather than unloading one version before loading another. The equivalent of the steps above would be:

```
[yourUsername@cirrus-login0 ~]$ module purge
[yourUsername@cirrus-login0 ~]$ module load gcc
[yourUsername@cirrus-login0 ~]$ gcc --version
[yourUsername@cirrus-login0 ~]$ module swap gcc gcc/7.2.0
[yourUsername@cirrus-login0 ~]$ gcc --version
```

```
gcc (GCC) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

gcc (GCC) 7.2.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

This achieves the same result as unload followed by load but in a single step.

Using software modules in scripts

Create a job that is able to run `python3 --version`. Remember, no software is loaded by default! Running a job is just like logging on to the system (you should not assume a module loaded on the login node is loaded on a compute node).

Installing software of our own

Most HPC clusters have a pretty large set of preinstalled software. Nonetheless, it's unlikely that all of the software we'll need will be available. Sooner or later, we'll need to install some software of our own.

Though software installation differs from package to package, the general process is the same: download the software, read the installation instructions (important!), install dependencies, compile, then start using our software.

As an example we will install the bioinformatics toolkit `seqtk`. We'll first need to obtain the source code from GitHub using `git`.

```
[yourUsername@cirrus-login0 ~]$ git clone https://github.com/lh3/seqtk.git
```

```
Cloning into 'seqtk'...
remote: Counting objects: 316, done.
remote: Total 316 (delta 0), reused 0 (delta 0), pack-reused 316
Receiving objects: 100% (316/316), 141.52 KiB | 0 bytes/s, done.
Resolving deltas: 100% (181/181), done.
```

Now, using the instructions in the README.md file, all we need to do to complete the install is to `cd` into the seqtk folder and run the command `make`.

```
[yourUsername@cirrus-login0 ~]$ cd seqtk
[yourUsername@cirrus-login0 ~]$ make
```

```
gcc -g -Wall -O2 -Wno-unused-function seqtk.c -o seqtk -lz -lm
seqtk.c: In function ‘stk_comp’:
seqtk.c:399:16: warning: variable ‘lc’ set but not used [-Wunused-but-set-variable]
    int la, lb, lc, na, nb, nc, cnt[11];
                ^
```

It's done! Now all we need to do to use the program is invoke it like any other program.

```
[yourUsername@cirrus-login0 ~]$ ./seqtk
```

```
Usage:   seqtk <command> <arguments>
Version: 1.2-r101-dirty

Command: seq       common transformation of FASTA/Q
         comp      get the nucleotide composition of FASTA/Q
         sample    subsample sequences
         subseq    extract subsequences from FASTA/Q
         fqchk     fastq QC (base/quality summary)
         mergepe   interleave two PE FASTA/Q files
         trimfq    trim FASTQ using the Phred algorithm

         hety      regional heterozygosity
         gc        identify high- or low-GC regions
         mutfa     point mutate FASTA at specified positions
         mergefa   merge two FASTA/Q files
         famask    apply a X-coded FASTA to a source FASTA
         dropse    drop unpaired from interleaved PE FASTA/Q
         rename    rename sequence names
         randbase  choose a random base from hets
         cutN      cut sequence at long N
         listhet   extract the position of each het
```

We've successfully installed our first piece of software!

Key Points

- Load software with `module load softwareName`
- Unload software with `module purge`
- The module system handles software versioning and package conflicts for you automatically.

(../131-coffee/index.html)

(../15 trans files/i

Transferring files

Overview

Teaching: 30 min
Exercises: 10 min
Questions

- How do I upload/download files to the cluster?

Objectives

- Be able to transfer files to and from a computing cluster.

Computing with a remote computer offers very limited use if we cannot get files to or from the cluster. There are several options for transferring data between computing resources, from command line options to GUI programs, which we will cover here.

Download files from the internet using wget

One of the most straightforward ways to download files is to use `wget`. Any file that can be downloaded in your web browser with an accessible link can be downloaded using `wget`. This is a quick way to download datasets or source code.

The syntax is: `wget https://some/link/to/a/file.tar.gz`. For example, download the lesson sample files using the following command:

```
[yourUsername@cirrus-login0 ~]$ wget https://aniabrown.github.io/hpc-carpentry-WHPC/files/bash-lesson.t
ar.gz
```

Transferring single files and folders with scp

To copy a single file to or from the cluster, we can use `scp`. The syntax can be a little complex for new users, but we'll break it down here:

To transfer *to* another computer:

```
[user@laptop ~]$ scp /path/to/local/file.txt yourUsername@remote.computer.address:/path/on/remote/compu
ter
```

To download *from* another computer:

```
[user@laptop ~]$ scp yourUsername@remote.computer.address:/path/on/remote/computer/file.txt /path/to/lo
cal/
```

Note that we can simplify doing this by shortening our paths. On the remote computer, everything after the `:` is relative to our home directory. We can simply just add a `:` and leave it at that if we don't care where the file goes.

```
[user@laptop ~]$ scp local-file.txt yourUsername@remote.computer.address:
```

To recursively copy a directory, we just add the `-r` (recursive) flag:

```
[user@laptop ~]$ scp -r some-local-folder/ yourUsername@remote.computer.address:target-directory/
```

A note on rsync

As you gain experience with transferring files, you may find the `scp` command limiting. The rsync (https://rsync.samba.org/) utility provides advanced features for file transfer and is typically faster compared to both `scp` and `sftp` (see below). It is especially useful for transferring large and/or many files and creating synced backup folders.

The syntax is similar to `scp`. To transfer *to* another computer with commonly used options:

```
[local]$ rsync -avzP /path/to/local/file.txt yourUsername@remote.computer.address:/path/on/remote/comp
uter
```

The `a` (archive) option preserves file timestamps and permissions among other things; the `v` (verbose) option gives verbose output to help monitor the transfer; the `z` (compression) option compresses the file during transit to reduce size and transfer time; and the `P` (partial/progress) option preserves partially transferred files in case of an interruption and also displays the progress of the transfer.

To recursively copy a directory, we can use the same options:

```
[local]$ rsync -avzP /path/to/local/dir yourUsername@remote.computer.address:/path/on/remote/computer
```

The `a` (archive) option implies recursion.

To download a file, we simply change the source and destination:

```
[local]$ rsync -avzP yourUsername@remote.computer.address:/path/on/remote/computer/file.txt /path/to/l
ocal/
```

Transferring files interactively with FileZilla (sftp)

FileZilla is a cross-platform client for downloading and uploading files to and from a remote computer. It is absolutely fool-proof and always works quite well. It uses the `sftp` protocol. You can read more about using the `sftp` protocol in the command line here (/hpc-carpentry-WHPC/discuss/index.html).

Download and install the FileZilla client from https://filezilla-project.org (https://filezilla-project.org). After installing and opening the program, you should end up with a window with a file browser of your local system on the left hand side of the screen. When you connect to the cluster, your cluster files will appear on the right hand side.

To connect to the cluster, we'll just need to enter our credentials at the top of the screen:

- Host: `sftp://login.cirrus.ac.uk`
- User: Your cluster username
- Password: Your cluster password
- Port: (leave blank to use the default port)

Hit "Quickconnect" to connect! You should see your remote files appear on the right hand side of the screen. You can drag-and-drop files between the left (local) and right (remote) sides of the screen to transfer files.

Archiving files

One of the biggest challenges we often face when transferring data between remote HPC systems is that of large numbers of files. There is an overhead to transferring each individual file and when we are transferring large numbers of files these overheads combine to slow down our transfers to a large degree.

The solution to this problem is to *archive* multiple files into smaller numbers of larger files before we transfer the data to improve our transfer efficiency. Sometimes we will combine archiving with *compression* to reduce the amount of data we have to transfer and so speed up the transfer.

The most common archiving command you will use on (Linux) HPC cluster is `tar`. `tar` can be used to combine files into a single archive file and, optionally, compress. For example, to collect all files contained inside `output_data` into an archive file called `output_data.tar` we would use:

```
[user@laptop ~]$ tar -cvf output_data.tar output_data/
```

The options we used for `tar` are:

- `-c` - Create new archive
- `-v` - Verbose (print what you are doing!)
- `-f mydata.tar` - Create the archive in file *output_data.tar*

The tar command allows users to concatenate flags. Instead of typing `tar -c -v -f`, we can use `tar -cvf`. We can also use the `tar` command to extract the files from the archive once we have transferred it:

```
[user@laptop ~]$ tar -xvf output_data.tar
```

This will put the data into a directory called `output_data`. Be careful, it will overwrite data there if this directory already exists!

Sometimes you may also want to compress the archive to save space and speed up the transfer. However, you should be aware that for large amounts of data compressing and un-compressing can take longer than transferring the un-compressed data so you may not want to transfer. To create a compressed archive using `tar` we add the `-z` option and add the `.gz` extension to the file to indicate it is compressed, e.g.:

```
[user@laptop ~]$ tar -czvf output_data.tar.gz output_data/
```

The `tar` command is used to extract the files from the archive in exactly the same way as for uncompressed data as `tar` recognizes it is compressed and un-compresses and extracts at the same time:

```
[user@laptop ~]$ tar -xvf output_data.tar.gz
```

Transferring files

Using one of the above methods, try transferring files to and from the cluster. Which method do you like the best?

Working with Windows

When you transfer files to from a Windows system to a Unix system (Mac, Linux, BSD, Solaris, etc.) this can cause problems. Windows encodes its files slightly different than Unix, and adds an extra character to every line.

On a Unix system, every line in a file ends with a `\n` (newline). On Windows, every line in a file ends with a `\r\n` (carriage return + newline). This causes problems sometimes.

Though most modern programming languages and software handles this correctly, in some rare instances, you may run into an issue. The solution is to convert a file from Windows to Unix encoding with the `dos2unix` command.

You can identify if a file has Windows line endings with `cat -A filename`. A file with Windows line endings will have `^M$` at the end of every line. A file with Unix line endings will have `$` at the end of a line.

To convert the file, just run `dos2unix filename`. (Conversely, to convert back to Windows format, you can run `unix2dos filename`.)

A note on ports

All file transfers using the above methods use encrypted communication over port 22. This is the same connection method used by SSH. In fact, all file transfers using these methods occur through an SSH connection. If you can connect via SSH over the normal port, you will be able to transfer files.

Key Points

- `wget` downloads a file from the internet.
- `scp` transfer files to and from your computer.
- You can use an SFTP client like FileZilla to transfer files through a GUI.

‹ (../14-modules/index.html)

› (../15
lunch

Using resources effectively

Overview

Teaching: 15 min
Exercises: 15 min
Questions

- What resources should I ask for in my job script?
- How can I get my jobs scheduled more easily?

Objectives

- Understand job size implications.
- Practice benchmarking a parallel code
- Understand how to submit a job that uses multiple nodes

We now know virtually everything we need to know about getting stuff on a cluster. We can log on, submit different types of jobs, use preinstalled software, and install and use software of our own. What we need to do now is use the systems effectively.

Estimating required resources using the scheduler

Although we covered requesting resources from the scheduler earlier, how do we know how much and what type of resources we will need in the first place?

Answer: we don't. Not until we've tried it ourselves at least once. We'll need to benchmark our job and experiment with it before we know how much it needs in the way of resources.

A good rule of thumb is to ask the scheduler for more time and memory (perhaps 20% more) than you expect your job to need after benchmarking. This ensures that minor fluctuations in run time or memory use will not result in your job being canceled by the scheduler. Keep in mind that the more resources you ask for, the longer your job will wait in the scheduler queue to run.

Benchmarking example

As an example, let's try benchmarking a very simple parallel program. This program calculates the temperature at a series of points along a beam over time given an initial temperature distribution, using an equation based on the values of neighbouring points.

This work can be done in parallel by splitting the points equally among all processors that the job is running on. Every time step, there will need to be some communication among processes to share values at the edges of each region, which will use up some time.

With that in mind, let's grab the code. It is available at https://github.com/aniabrown/ARC_parallel_programming_exercises.

We can clone the code from GitHub onto Cirrus after first loading the git module:

```
$ module load git
$ git clone https://github.com/aniabrown/ARC_parallel_programming_exercises
```

```
Cloning into 'ARC_parallel_programming_exercises'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 30 (delta 8), reused 19 (delta 6), pack-reused 0
Unpacking objects: 100% (30/30), done.
```

The example is in the pde directory:

```
$ cd ARC_parallel_programming_exercises/pde
$ ls
```

```
heat_serial.c heat_mpi.c  makefile  mpi_job.pbs
```

There are two different versions of the code: a serial version designed to run on one process and an MPI version designed to run on multiple processes.

This code is written in C, which needs to be compiled into a machine readable executable before we can run it. To compile the serial version we can use the command:

```
$ make heat_serial
```

```
gcc -O3 -mavx -std=c99 -Wall -Wextra -pedantic -c heat_serial.c
gcc -O3 -mavx -std=c99 -Wall -Wextra -pedantic -o heat_serial heat_serial.o -lm
```

Just this once, we will see what the code does by running on the login node – we know that the program is quick enough to do this.

```
$ ./heat_serial
```

```
The RMS error in the final solution is 9.3145898e-12
On 1 process the time taken was 0.088503 seconds
```

This code is fast enough than we would usually not bother running it on more than one core (the problem size is very small) but we can still use it to get a feel for how to choose how many cores to request. The main things to keep in mind are that eventually adding more cores will likely not make the program run much faster (it may even run slower) and that the more cores you ask for the longer your job will wait in the scheduler queue.

Running across multiple processes

To compile the parallel version of the code, we will need to load the MPI module:

```
module load mpt
make heat_mpi
```

```
mpicc -O3 -mavx -std=c99 -Wall -Wextra -pedantic -c heat_mpi.c
mpicc -O3 -mavx -std=c99 -Wall -Wextra -pedantic -o heat_mpi heat_mpi.o -lm
```

There is a submission script, mpi_job.pbs, in the pde folder:

```
#!/bin/bash

# job configuration
#PBS -N pde
#PBS -l select=1:ncpus=2
#PBS -l walltime=00:00:30

# Change to the directory that the job was submitted from
# (remember this should be on the /work filesystem)
cd $PBS_O_WORKDIR

module load mpt
module load intel-compilers-17

mpiexec_mpt -ppn 2 -n 2 ./heat_mpi
```

Edit this to run on job counts between 1 and 72 processes and record the average calculation time reported in each case. How does the program scale? Is there an ideal process count, taking into account queuing time?

Note, the mpiexec_mpt command is used to launch the program on multiple processes. It takes the number of processes per node as the first argument and the total number of processes in the job as the second argument. E.g. for a job running 72 processes across two nodes you would use -ppn 36 -n 72. You will also need to edit the #PBS -l select statement to match.

Hint: A good rule of thumb is to make a guess and then roughly double or halve the process count for each new run.

Key Points

- The smaller your job, the faster it will schedule.
- For a constant proble size, most parallel codes will eventually stop getting faster with more processes.
- If you have a serial code, it will still only run on one core even if you ask for multiple cores in your submission script.

(../151-lunch/index.html)

(../16 job- array

Scheduling multiple similar jobs

Overview

Teaching: 15 min
Exercises: 10 min
Questions

- How do we launch many similar jobs?
- How can we monitor job arrays?

Objectives

- Understand how to launch similar jobs using a job array.

We just saw an example of how parallelising a job to run over too many processes may eventually result in negative returns due to the time taken for all those processes to communicate with each other. However, sometimes we are in the lucky situation where the work we need to do can be separated into truly independent tasks, with no communication required between them. The classic example of this is a parameter search, where we run the same algorithm on a range of different inputs. In that case we can make use of a feature of the job scheduler known as a job array.

Job Arrays

Job arrays are a way to cleanly submit many similar jobs while only having to define and launch one submission script.

Here is an example submission script. Either copy paste it to `example-job.sh` or download it using:

```
$ wget https://aniabrown.github.io/hpc-carpentry-WHPC/files/simple_job_array_example.tar.gz
$ tar -xzf simple_job_array_example.tar.gz
$ cd simple_job_array_example
```

```
#!/bin/bash

# job configuration
#PBS -N job_array_example
#PBS -l select=1:ncpus=1
#PBS -l walltime=00:02:00
#PBS -J 1-4

# Change to the directory that the job was submitted from
# (remember this should be on the /work filesystem)
cd $PBS_O_WORKDIR

echo "Running job ${PBS_ARRAY_INDEX} of job array"

sleep 120
```

This is very similar to scripts we've seen before, with two changes. The configuration parameter `#PBS -J 1-4` tells the scheduler to launch four jobs, assigning each an index which will range between one and four. We can also access this index that has been assigned to each job using the PBS variable `PBS_ARRAY_INDEX`.

We can launch the job array as we would launch a single job

```
$ [yourUsername@cirrus-login0 ~]$ qsub -A tc008 -q R1262266 example-job.sh
```

The status of the job array as a whole can be viewed using `qstat -u $USER`

```
indy2-login0:
                                                           Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1266791[].indy2 ania     workq    job_array_    --    1   1    --  00:00 B   --
```

And we can also look at the individual jobs using `qstat -t jobId[]` . Eg `qstat -t 1266791[]`

```
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
1266791[].indy2-l job_array_examp  ania                     0 B workq
1266791[1].indy2- job_array_examp  ania              00:00:00 R workq
1266791[2].indy2- job_array_examp  ania              00:00:00 R workq
1266791[3].indy2- job_array_examp  ania              00:00:00 R workq
1266791[4].indy2- job_array_examp  ania              00:00:00 R workq
```

Remember, for all purposes other than ease of launching, these are completely separate jobs. They may get scheduled to run at different times, and each will generate its own output and error files:

```
$ ls
```

```
example_job.pbs               job_array_example.e1266791.3  job_array_example.o1266791.2
job_array_example.e1266791.1  job_array_example.e1266791.4  job_array_example.o1266791.3
job_array_example.e1266791.2  job_array_example.o1266791.1  job_array_example.o1266791.4
```

Currently these jobs only print out their unique id, which is not very useful. Additionally, the output of these jobs is scattered among all the different output files, which we will need to process somehow.

Example with input/output files per job

Let's take a look at a more realistic example. Download and untar the example files using:

```
$ wget https://aniabrown.github.io/hpc-carpentry-WHPC/files/job_array_example.tar.gz
$ tar -xzf job_array_example.tar.gz
$ cd job_array_example
$ ls
```

```
input_1.txt  input_3.txt  process_file.py       summarise_outputs.py
input_2.txt  input_4.txt  submit_job_array.pbs
```

There are four input files, each containing a list of numbers. We can see an example using `cat input_1.txt` .

```
31587
16729
29533
25846
21477
6016
25138
30079
4120
12355
793
26439
31226
18139
4081
9797
32245
6563
15591
7784
```

The short python script `process_file.py` takes one of these input files, finds the maximum number in the file and prints that value to an output file that we specify – very slightly more useful than our first example.

The script takes two arguments – the input and output file names. For example:

```
$ module load anaconda/python3
$ python3 process_file.py input_1.txt output_1.txt
```

Process all four input files using a job array

The example folder contains the same submission script as in the previous example. Can you add a line that runs the `process_file.py` script on each input file? Remember, submitting the job array submission script will launch four separate jobs each with their own value of `${PBS_ARRAY_INDEX}`.

Solution

```
#!/bin/bash

# job configuration
#PBS -N job_array_example
#PBS -l select=1:ncpus=1
#PBS -l walltime=00:00:30
#PBS -J 1-4

# Change to the directory that the job was submitted from
# (remember this should be on the /work filesystem)
cd $PBS_O_WORKDIR

echo "Running job ${PBS_ARRAY_INDEX} of job array"

module load anaconda/python3

python3 process_file.py input_${PBS_ARRAY_INDEX}.txt output_${PBS_ARRAY_INDEX}.txt
```

In this example, we create one output file per job in the job array – this is fairly typical. The last piece of work we need to do is then to process those output files – in this example there is a python script, `summarise_outputs`, which reads through all the output files and puts them into a summary results table, `summary_file.csv`.

```
$ module load anaconda/python3
$ python3 summarise_outputs.py
$ cat summary_file.csv
```

```
job, max
1,32245
2,32244
3,29748
4,30474
```

Responsible use

Warning: Job arrays give you a lot of power – use it wisely! All the jobs in a job array will go through the queuing system so if you submit a job array that is too large eventually your jobs will get held to allow other users through. However, it's still best not to submit thousands of jobs at once, and always remember to test on a small numbers of jobs first!

Key Points

- Job arrays allow you to launch many jobs that are each assigned a different index value, using just one submission script.

(../16-resources/index.html) (../17-respo

Using shared resources responsibly

Overview

Teaching: 15 min
Exercises: 5 min
Questions

- How can I be a responsible user?
- How can I protect my data?
- How can I best get large amounts of data off an HPC system?

Objectives

- Learn how to be a considerate shared system citizen.
- Understand how to protect your critical data.
- Appreciate the challenges with transferring large amounts of data off HPC systems.
- Understand how to convert many files to a single archive file using tar.

One of the major differences between using remote HPC resources and your own system (e.g. your laptop) is that they are a shared resource. How many users the resource is shared between at any one time varies from system to system but it is unlikely you will ever be the only user logged into or using such a system.

We have already mentioned one of the consequences of this shared nature of the resources: the scheduling system where you submit your jobs, but there are other things you need to consider in order to be a considerate HPC citizen, to protect your critical data and to transfer data

It’s pretty hard to break a cluster

While it is important to keep in mind the following guidelines for using the cluster responsibly, it is also possible to be too scared of breaking something – remember, these systems exist to be used! An HPC cluster is a relatively robust system that is specifically designed to be able to handle multiple users working on it at once – for instance, there are systems preventing you from deleting files created by other users, and if you accidentally submit too many jobs the scheduler will notice and hold some of them for other users to get through. If you’re ever in doubt about something you want to try, get in touch with the support team for the cluster you’re using – their email should be prominent in the cluster documentation

Be kind to the login nodes

The login node is often very busy managing lots of users logged in, creating and editing files and compiling software! It doesn’t have any extra space to run computational work.

Don’t run jobs on the login node (though quick tests are generally fine). A “quick test” is generally anything that uses less than 1 minute of time. If you use too much resource then other users on the login node will start to be affected - their login sessions will start to run slowly and may even freeze or hang.

Login nodes are a shared resource

Remember, the login node is shared with all other users and your actions could cause issues for other people. Think carefully about the potential implications of issuing commands that may use large amounts of resource.

You can always use the commands `top` and `ps ux` to list the processes you are running on a login node and the amount of CPU and memory they are using. The `kill` command can be used along with the *PID* to terminate any processes that are using large amounts of resource.

Login Node Etiquette

Which of these commands would probably be okay to run on the login node?

- python physics_sim.py
- make
- create_directories.sh
- molecular_dynamics_2
- tar -xzf R-3.3.0.tar.gz

If you experience performance issues with a login node you should report it to the system staff (usually via the helpdesk) for them to investigate. You can use the `top` command to see which users are using which resources.

Test before scaling

Can you see what's wrong with this submission script?

```
#!/bin/bash

# job configuration
#PBS -N long_python_job
#PBS -l select=1:ncpus=36
#PBS -l walltime=24:00:00

# Change to the directory that the job was submitted from
# (remember this should be on the /work filesystem)
cd $PBS_O_WORKDIR

python my_script.py
```

Remember that you are generally charged for usage on shared systems. A simple mistake in a job script can end up costing a large amount of resource budget. Imagine a job script with a mistake that makes it sit doing nothing for 24 hours on 1000 cores or one where you have requested 2000 cores by mistake and only use 100 of them! This problem can be compounded when people write scripts that automate job submission (for example, using job arrays). When this happens it hurts both you (as you waste lots of charged resource) and other users (who are blocked from accessing the idle compute nodes).

On very busy resources you may wait many days in a queue for your job to fail within 10 seconds of starting due to a trivial typo in the job script. This is extremely frustrating! Most systems provide dedicated resources for testing that have short wait times to help you avoid this issue.

Test job submission scripts that use large amounts of resource

Before submitting a very large or very long job submit a short truncated test to ensure that the job starts as expected

Before submitting a large collection of jobs, submit one as a test first to make sure everything works as expected.

Have a backup plan

Although many HPC systems keep backups, it does not always cover all the file systems available and may only be for disaster recovery purposes (*i.e.* for restoring the whole file system if lost rather than an individual file or directory you have deleted by mistake). Your data on the system is primarily your responsibility and you should ensure you have secure copies of data that are critical to your work.

Version control systems (such as Git) often have free, cloud-based offerings (e.g. Github, Gitlab) that are generally used for storing source code. Even if you are not writing your own programs, these can be very useful for storing job scripts, analysis scripts and small input files.

For larger amounts of data, you should make sure you have a robust system in place for taking copies of critical data off the HPC system wherever possible to backed-up storage. Tools such as `rsync` can be very useful for this.

Your access to the shared HPC system will generally be time-limited so you should ensure you have a plan for transferring your data off the system before your access finishes. The time required to transfer large amounts of data should not be underestimated and you should ensure you have planned for this early enough (ideally, before you even start using the system for your research).

In all these cases, the helpdesk of the system you are using should be able to provide useful guidance on your options for data transfer for the volumes of data you will be using.

Your data is your responsibility

Make sure you understand what the backup policy is on the file systems on the system you are using and what implications this has for your work if you lose your data on the system. Plan your backups of critical data and how you will transfer data off the system throughout the project.

Transferring data

As mentioned above, many users run into the challenge of transferring large amounts of data onto or off HPC systems at some point.

If you have related data that consists of a large number of small files it is strongly recommended to pack the files into a larger *archive* file for long term storage and transfer. A single large file makes more efficient use of the file system and is easier to move, copy and transfer because significantly fewer meta-data operations are required. Archive files can be created using tools like `tar` and `zip` . We have already met `tar` when we talked about data transfer earlier.

If you are transferring large (>10GB) amounts of data it is always useful to run some tests that you can use to extrapolate how long it will take to transfer your data.

Getting help

If you believe you may have unusual needs, such as needing to transfer very large amounts of data to the cluster, talk to the support team for the cluster you are using – you should be able to find their contact details in the cluster documentation. They are there to help, and in general will be much happier answering a query ahead of time than needing to fix something after the fact!

After checking the cluster documentation, the support team is also your first port of call for any general advice on how to use the system. Depending on the cluster, the preferred method of contact may be direct email to the support team, or there may be a forum that is public for all users to view.

Finally, there will often be further training available for the particular cluster you are using. Many clusters will have a mailing list with updates on these events as well as important notices for users such as maintenance shut downs.

Key Points

- Clusters are designed to handle large number of users
- The shared resources to be careful with are memory and cpus on the login nodes, and I/O
- Your data on the system is your responsibility.
- Plan and test large data transfers.
- It is often best to convert many files to a single archive file before transferring.
- Don't run stuff on the login node.
- Really don't run stuff on the login node.
- When in doubt, talk to the cluster support team first

‹ (../161-job-arrays/index.html)

› (../17 coffe