# CHAPTER 6：Accessing Software

# User Environment

HPC systems can support a large and complex range of software

- Operating System (Ubuntu, CentOS)
- Compilers (e.g. Intel, GCC, Cray, PGI)
- Debuggers and Profilers (e.g. MAP , DDT)
- Performance mathematical libraries (e.g. Intel's MKL, Lapack, Petsc)
- Parallel programming libraries (e.g. MPI)
- File format libraries for parallel IO (e.g. HDF5)

Project groups may be expected to manage the installation of their own software stack.

**All of the above software is not immediately available as soon as you log in.**

# Environment Modules

To prevent conflicts between software names and versions, applications and libraries are not installed in the standard directory locations.

Modules modify the environment to easily locate software, libraries, documentation, or particular versions of the software

```
module load openmpi
```

# Module Commands

| Command | Description |
|---|---|
| `module avail` | Show available modules |
| `module list` | List loaded modules |
| `module load` ***`modulename`*** | Load a module into the current environment |
| `module rm\|unload` ***`modulename`*** | Unload a module from the environment |
| `module purge` | Unload all loaded modules |
| `module swap` ***`module1 module2`*** | Swap a loaded module with another |
| `module show` ***`modulename`*** | Give help for a particular module |
| `module help` | Show module specific help |

# How it works…

Unix shells (Bash. C-shell, etc) use system environment variables to define important values

- By convention, these variables use upper-case letters
- The shell command **printenv** prints all the variables
- To look at a particular one, we use ‘echo’ and the variable name
  - i.e. to look at the **PATH** variable we type

**echo $PATH** ←

*Note the $ is used to tell shell that we are looking for a variable called PATH*

Another important variable is **LD_LIBRARY_PATH** that lists the locations of shared libraries.

Modules work by modifying these variables in a very easy to use manner, independent of the shell you are using.

**PATH** is used by the shell to search where executables might be. The shell command **which** is used to find the path of an executable.

For example, lets look for where our Python executable may be

```
which python
/usr/bin/python
```

And we can ask Python which version it is

```
python --version
Python 2.7.5
```

Suppose we wanted the latest version of Python ?

# module avail

So before we install the latest version, or request the Help Desk to do it, let's check the modules

```
module avail python
----------/usr/local/Modules/modulefiles ----------
python/2.7.12
```

So let's use it!

```
module load python
```

Let's check we have the right version

```
 which python
/usr/local/python/2.7.12/bin/python
python --version
Python 2.7.12
```

# Removing Modules

Suppose we wanted to go back to using the system python. This can be done with a number of commands

```
module rm python
module delete python
module purge #remove ALL modules, not just python
```

Let's check we have gone back to our original version

```
which python
/usr/bin/python
python –version
module list
```

# module Versions

It's possible to have lots of different versions of the same software

```
module avail python


------/usr/local/Modules/modulefiles

python/2.7.12-gcc       python/2.7.8-gcc(default) python/2.7-intel       python/3.4.3-gcc       python/3.5.1-gcc
```

Environment module lets you specify the version, or swap between them

**module load python** #loads the default, same as *module load python/2.7.8-gcc*
**which python**
/usr/local/python/2.7.8-gcc/bin/python

**module swap python/2.7-intel** #unloads the existing Python, then loads the version
#compiled with the intel compiler
**which python**
/usr/local/python/2.7-intel/bin/python

# Exercise 6

Build on previous work, write a script (say i**nterrogate.sh**) to interrogates a node for information

- Print the hostname (*hostname*)
- Print the time of day (*date*)
- Look for an OS system python
  - What is its path?
  - What version is it?

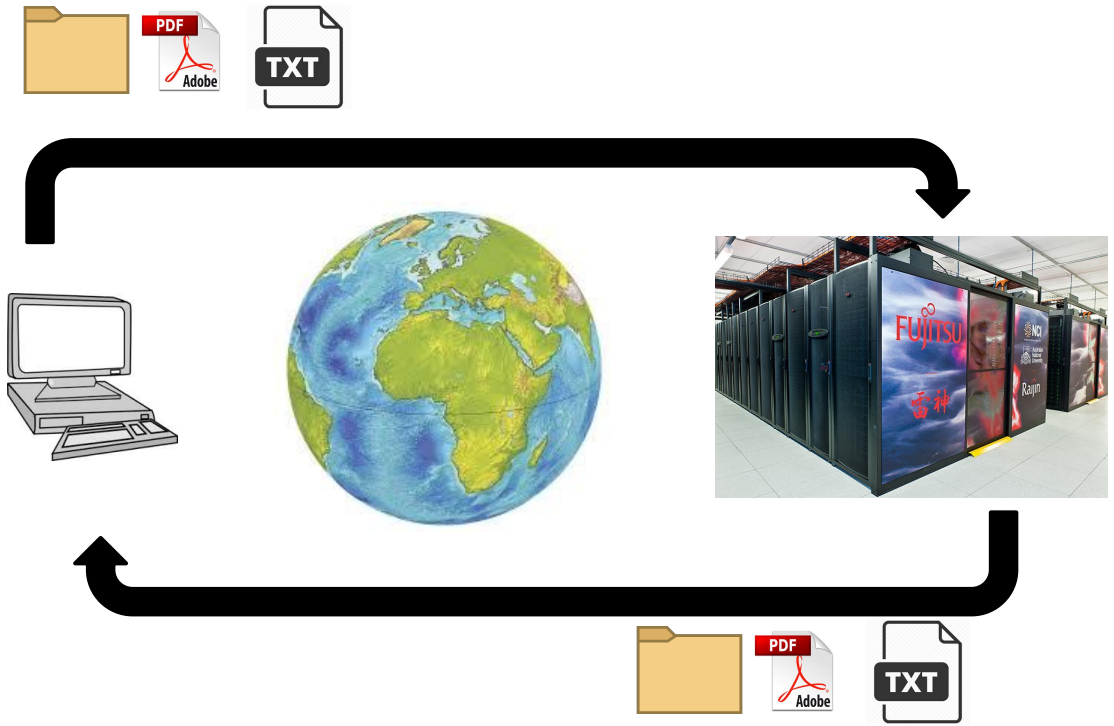
- Load Python via modules
  - What is its path?
  - What version is it?
- Print a farewell message and time

All Good

Need help

- Once you have run this on the login node, write a slurm script to run **interrogate.sh** on a single node
- If you have time, modify the slurm script to run on 2 nodes.
  - Remember to use **srun** when invoking **interrogate.sh**
  - Is the output of the text what you expect?

# CHAPTER 7 : Transferring software



# scp – Copying files between computers

You can copy files *between* computers with *scp* – 'secure cp'
scp has similar syntax to cp. You specify the username and machine name of the remote host in the command line

**scp <from> <to>**

where <from> or <to> can also specify a **username@computername:**

Note the ':'

e.g.
**scp *.db username@computer:/home/apps/dbfiles**

*The wild card *.db matches any filename whose ending is the string '.db'*

# scp – Copying files between computers

EXAMPLES (Using a computer called **raijin.nci.org.au** with a user called **vader**):

***COPYING TWO FILES TO RAIJIN***

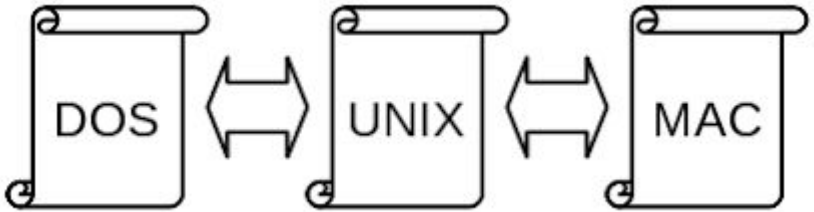
scp mfile.c myfile.h vader@raijin.nci.org.au:

***COPYING FROM RAIJIN WITH A WILDCARD***

scp vader@raijin.nci.org.au:mydata/*.dat /home/user

Other interesting options (‘man cp’)
**-i** <identity_file> use a predefined crypto-key file, removing the need to use a password. Use with caution!
**-r** copy directories recursively

# dos2unix and unix2dos




In Windows (DOS), text files have every line terminated with two characters –CARRIAGE RETURN and LINEFEED =>
*Using the C computer language syntax, this maps to* **\r\n**
In Unix and Mac, text files have every line terminated with **\n** only

When viewing a DOS file in Unix, you may see a **^M** at the end of every line.
Or worst, the character may appear invisible and cause errors later !

Two tools exist to convert text files between the two Operating systems
**dos2unix <filename>**
**unix2dos <filename>**

# Making files easier to move


**COMPRESSING FILES**
Lossless compression of files is useful:

- When transmitting files across the internet
- Saving space in your file system (NCI have quotas)

There are many tools to compress files. One of the most common is **gzip**.

To compress a file:
**gzip myfile.txt**
Produces a file called myfile.txt.gz (and removes myfile.txt)

To get the file back:
**gunzip myfile.txt.gz**
Produces a file called myfile.txt (and removes myfile.txt.gz)

# Creating and extracting archives


**Archiving files**
It is often useful to combine a large number of files into one big file

- When transmitting files across the internet
- Quicker access on tape archive systems
- Your file system may have limits on the number of files you can have (N have quotas on this – the inode quota)

There are many tools to archive files. One of the most common is **tar**.
To create an archive:
**tar –cf archive.file.name.tar <list of files>**
Produces a file called **archive.file.name.tar**

To extract the archive
**tar -xf archive.file.name.tar**
Extracts all the files in the archive

Once an archive is created, it is a good idea to compress it as well.
This can be done automatically with tar by adding the –z flag. i.e**. tar –cfz** …

# Exercise

You want to copy some data from another machine to your home directory.
You know the file is called **whatami.tar.gz** and it lives in **/tmp** of a machine called **slurm0**.
Use scp to fetch the file. Don't forget the "." at the end of the line.

All Good

```
scp slurm0:/tmp/whatami.tar.gz .
```

Need help

Check that your the scp worked and you have a file called whatami.tar.gz in your directory. Now lets see what is in the archive..

```
tar -xf whatami.tar.gz
```

Use **ls** and **cat** to see what was produced by the file.

## Chapter 7 Using resources effectively

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.

sacct -u {userID}

sacct -j {jobID} -l

- **Hostname** - Where did your job run?
- **MaxRSS** - What was the maximum amount of memory used?
- **Elapsed** - How long did the job take?
- **State** - What is the job currently doing/what happened to it?
- **MaxDiskRead** - Amount of data read from disk.
- **MaxDiskWrite** - Amount of data written to disk.

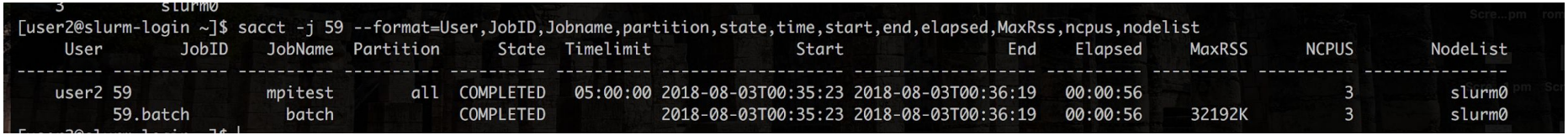
```
[user2@slurm-login ~]$ sacct -u user2
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
54             hostname        all slurmclass          2  COMPLETED      0:0
55             hostname        all slurmclass          2  COMPLETED      0:0
56               run.sh        all slurmclass          1  COMPLETED      0:0
56.batch          batch             slurmclass          1  COMPLETED      0:0
57              mpitest        all slurmclass          2  COMPLETED      0:0
57.batch          batch             slurmclass          2  COMPLETED      0:0
58              mpitest        all slurmclass          1  COMPLETED      0:0
58.batch          batch             slurmclass          1  COMPLETED      0:0
59              mpitest        all slurmclass          3  COMPLETED      0:0
59.batch          batch             slurmclass          3  COMPLETED      0:0
```

```
[user2@slurm-login ~]$ sacct -j 59
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
59             mpitest        all slurmclass          3  COMPLETED      0:0
59.batch         batch            slurmclass          3  COMPLETED      0:0
```

- The state showing that the job is completed successfully.
- The partition that was used in `all`
- The account that was used is `slurmclass`

If you would like to see more information about a job, you can always customise the format for your command

```
sacct -j 59 --format=User,JobID,Jobname,partition,state,time,start,end,elapsed,MaxRss,ncpus,nodelist
```



```
[user2@slurm-login ~]$ sacct -j 59 --format=User,JobID,Jobname,partition,state,time,start,end,elapsed,MaxRss,ncpus,nodelist
     User        JobID    JobName  Partition      State  Timelimit              Start                End    Elapsed     MaxRSS      NCPUS        NodeList
--------- ------------ ---------- ---------- ---------- ---------- ------------------- ------------------- ---------- ---------- ---------- ---------------
    user2 59             mpitest        all  COMPLETED   05:00:00 2018-08-03T00:35:23 2018-08-03T00:36:19   00:00:56                     3          slurm0
          59.batch         batch             COMPLETED            2018-08-03T00:35:23 2018-08-03T00:36:19   00:00:56     32192K          3          slurm0
```

Why is my job not running?

Use “squeue --start” command
Jobs will be listed in order expected start time
Depending upon configuration, not all jobs may have a start time set
Times are only estimates and subject to change

```
01:15:37 m3-login1:~ ctan$ squeue --start
             JOBID PARTITION     NAME     USER ST          START_TIME  NODES SCHEDNODES           NODELIST(REASON)
           2796646      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2796655      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2797577      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2797637      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2798766      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2798776      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2802511      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2802520      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2802526      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2802988      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2802994      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           2809141      comp GPR_tens      yc1 PD                 N/A      1 (null)              (DependencyNeverSatisfied)
           3133116       m3i       nt  miguelg PD                 N/A      1 (null)              (launch failed requeued held)
           3297560       m3g Refine3D joshuamh PD                 N/A      1 (null)              (QOSMaxGRESPerUser)
 3157584_[464-500]      comp nCellsDe mcfadyen PD                 N/A      1 (null)              (QOSMaxCpuPerUserLimit)
           3312396      comp hiplexpi jste0021 PD                 N/A      1 (null)              (Priority)
           3312397      comp hiplexpi jste0021 PD                 N/A      1 (null)              (Priority)
           3312365      comp hiplexpi jste0021 PD 2018-08-03T01:35:12      1 m3i009              (Resources)
           3312385      comp hiplexpi jste0021 PD 2018-08-03T01:35:12      1 m3i029              (Priority)
           3312393      comp hiplexpi jste0021 PD 2018-08-03T01:35:13      1 m3i033              (Priority)
           3312394      comp hiplexpi jste0021 PD 2018-08-03T01:35:16      1 m3i036              (Priority)
           3312395      comp hiplexpi jste0021 PD 2018-08-03T01:35:16      1 m3h005              (Priority)
           3273020       m3a hsa_chn2 mziemann PD 2018-08-03T03:21:01      1 m3a006              (Resources)
           3284320       m3a hsa_chn2 mziemann PD 2018-08-03T03:21:01      1 m3a007              (Resources)
           3130659      comp oxalipla sgwe0001 PD 2018-08-03T04:05:56      1 (null)              (Priority)
           3221688       m3i aFe5Dv3A      eyk PD 2018-08-03T07:10:34      4 (null)              (QOSMaxCpuPerUserLimit)
           3223197       m3i aFe5Dv2B      eyk PD 2018-08-03T07:10:34      4 (null)              (QOSMaxCpuPerUserLimit)
           3073122      comp 238-cpcm sgwe0001 PD 2018-08-03T07:16:35      1 m3c005              (Priority)
           3073134      comp 238-smd- sgwe0001 PD 2018-08-03T07:16:35      1 (null)              (Priority)
           3073164      comp 238-gas- sgwe0001 PD 2018-08-03T07:16:35      1 (null)              (Priority)
           3073192      comp 248-cpcm sgwe0001 PD 2018-08-03T07:16:35      1 (null)              (Priority)
           3073193      comp 248-smd- sgwe0001 PD 2018-08-03T07:16:35      1 (null)              (Priority)
           3073195      comp 248-gas- sgwe0001 PD 2018-08-03T07:16:35      1 (null)              (Priority)
```

## Be kind to the login node

- The login node is very busy managing lots and lots of jobs! It doesn't have any extra space to run computational work. Don't run jobs on the login node, you can use the interactive session to test out your script.
- Remember, the login node is to be shared with other users.
- If someone is being inappropriate and using the login node to run all of their stuff, message an administrator to take a look at things and deal with them.

## Test before scaling

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

## Understanding the filesystem

- always know where to write your data, in the production system, there's usually some space that being backed up and some space might be allocated for scratch
- make sure you check your quota for each of this filesystem

## Save time

- Compress files before transferring to save file transfer times with large datasets.
- The less resources you ask for, the faster your jobs will find a slot in which to run. Lots of small jobs generally beat a couple big jobs.

## Software tips

- You can generally install software yourself, but if you want a shared installation of some kind, it might be a good idea to message an administrator.
- Always use the default compilers if possible. Newer compilers are great, but older stuff generally has less compatibility issues.

# HPC Clusters in Monash University

# MASSIVE M3

A Computer for Next-Generation Data Science

1,700 Intel Haswell CPU-cores
2,000 Intel Skylake CPU-cores

48 NVIDIA Tesla K80 GPU
20 NVIDIA Tesla P100 GPU
60 NVIDIA Tesla V100 GPU
coprocessors for data processing and high end visualisation

A 3 petabyte Lustre parallel file system

100 Gb/s Ethernet Mellanox Spectrum
Supplied by Dell, Mellanox and NVIDIA


# Monarch

## Campus Cluster

**Provide Monash researchers with a local capability that focuses on engagement, education and community.**

## Investment

A co-investor model
Examples include Computational Chemistry, Astro and Fluid Dynamics
1/3rd of MonARCH is co-purchased

## Integrated into undergraduate study

### CHM3911 Advanced Physical Chemistry

80 students across 3 practical sessions
Gaussian and GaussView for calculations

Students taught how to use a HPC system to perform their calculations

## WRAPPING UP

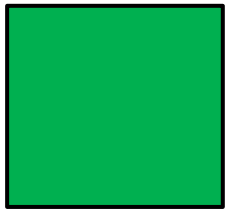
**Do you have the tools you need to do your Research?**

**Would you want to know more?**

- Email mcc-help@monash.edu
- Data Fluency Slack Channel
- Bioinformatic Drop-in (every Friday)
- https://docs.monarch.erc.monash.edu.au
- https://docs.massive.org.au
- HPC web site (i.e. www.nci.org.au)
- Google

## Your Feedback please!

ONE THING YOU LIKED



ONE THING TO IMPROVE