Introduction to High Performance Computing: course notes and exercises

Part 1: Introduction

The 'Introduction to HPC' course materials consist of these notes and the course slides. Together they provide a practical introduction to HPC concepts and the University's 'Rocket' HPC service. If you are working through the materials on your own, start with the course slides and switch to these notes where indicated.

You will need an account on the Rocket HPC service in order to complete the course exercises. People who have booked to attend a scheduled course should already have been given a temporary account; if you have not been booked onto the course then please get in touch with the Rocket team via the IT service desk (https://nuservice.ncl.ac.uk) to arrange access.

Logging in to Rocket

Log into Rocket using one of the methods listed below, which all provide a *secure shell (ssh)* connection. The first time you login by any method, you will see a message asking if you are sure about connecting to this server. Agree to connect. This message should only appear the first time you login from that machine.

If you cannot log in to Rocket, please contact the course leader.

From outside the campus

Rocket is directly accessible only from machines on the campus network. From outside, you will need to connect first to a more accessible system such as the University's linux time-sharing service, unix.ncl.ac.uk. Then connect to Rocket from there. If you will connect this way routinely, use the instructions at http://ncl.ac.uk/itservice/research/hpc/remoteaccess to streamline and simplify the process.

Note that it is also possible to connect via RAS but not from the Windows Virtual Desktop.

Working off-campus on a Windows PC

If you have not done so already, install the home edition of MobaXterm from https://mobaxterm.mobatek.net/). An older product, PuTTY, is an alternative, available from https://putty.org/.

Start MobaXterm, then:

- Select Session (top left)
- From the Session settings panel, select SSH
- In the Remote host field, enter unix.ncl.ac.uk
- Click on OK
- If you see a message asking if you are sure, agree to connect
- Login using your IT Service username (e.g. nab789, b1234567)
- Nothing will be displayed on the screen while you type your password. This is normal.

At this point you should be logged into the University's linux time-sharing service, Aidan. Your screen should display:

```
Welcome to AIDAN

-sh-4.2$
```

The line '-sh-4.2$' is your 'command prompt', where you can type commands. Type the following command to connect to Rocket:

```
ssh rocket.hpc
```

Log in using your IT Service username and password. You should now see a welcome message and the Rocket command prompt.

Working off-campus on a Mac or linux machine

Open a terminal window and type:

ssh <*username*>@unix.ncl.ac.uk

Where <*username*> is your IT Service username, e.g. nab789 or b1234567. If you see a message asking if you are sure, agree to connect.

Then login using your IT service password. Note that nothing will be displayed on the screen while you type your password. This is normal.

At this point you should be logged into the University's linux time-sharing service, Aidan. Your screen should display:

```
Welcome to AIDAN

-sh-4.2$
```

The line '-sh-4.2$' is your 'command prompt', where you can type commands. Type the following command to connect to Rocket:

```
ssh rocket.hpc
```

Log in using your IT Service username and password. You should now see a welcome message and the Rocket command prompt.

From a Windows machine on campus

From a desktop in a PC cluster room, use PuTTY to login to Rocket:

- Click on the Windows Start menu, search for PuTTY and start it
- In PuTTY's Host Name field, enter rocket.hpc.ncl.ac.uk
- Click on Open
- Login using your IT Service username (e.g. nab789, b1234567)

PuTTY is *not* installed on the Windows Virtual Desktop (WVD).

If you work in an office or on your own PC on the campus network, you may find that MobaXterm provides a more modern interface. It can be downloaded from https://mobaxterm.mobatek.net/

In MobaXterm:

- click on `Session`
- In the Session window, select SSH
- In the Remote host field, type `rocket.hpc.ncl.ac.uk`
- Click on OK
- Login using your IT Service username (e.g. nab789, b1234567)

Some other free tools are also worth considering. WinSCP is particularly helpful for easy transfer of files between Rocket and a Windows PC. Download PuTTY to go with it if you wish to run a login window from WinSCP.

In WinSCP,

- Set the host name to rocket.hpc.ncl.ac.uk
- Enter your username and password
- Type Ctrl-P, or click on 'New session in PUTTY' on the top menu
- In the black pop-up window, enter your username and password again

From a linux/Apple machine on campus

From a linux or Apple machine, open a terminal window and type:

```
ssh rocket.hpc
```

Login using your IT Service username (e.g. nab789, b1234567)

Logging out

No matter how you logged in, you can logout from Rocket by typing:

```
exit
```

Working with command-line linux

Rocket does not have a desktop environment. Once you have logged in as described above, you will be presented with a command prompt at which you will type commands:

```
Rocket-login 1 ->
```

Although it is possible to open windows on Rocket, most of your work – and all of this course – will involve typed commands rather than mouse clicks.

There are some basic features that apply to most linux commands:

1. Linux is case-sensitive, so ABC is not the same as Abc. This applies to all of linux, including filenames, commands, usernames and passwords.
2. Commands are usually in lower case.
3. After typing a command, press the Enter key to execute it.
4. Many commands have options to extend or change the results. These are usually invoked with a hyphen, in the form:
   command -option1 -option2
5. An asterisk (*) will act as a wildcard, e.g. for filenames
6. Use the tab key to auto-complete command names and file/directory names. Start typing the command until the remainder is likely to be unique, and then press the tab key to attempt to auto-complete the command.
7. Use the up-arrow to return to a previous command.

Linux also uses *environment variables* as a way to hold information that can be passed to commands and applications. Environment variable names begin with a $ and are in capitals. For example, $HOME contains the location of your home directory and can be used instead of typing that location. The following commands produce the same results:

```
ls /mnt/nfs/home/<userid>
ls $HOME
```

Some exercises in this course ask you to use a text editor. Several editors are available on Rocket, including nano, emacs and vim. Beginners are advised to use ‘nano’. To enter the editor, type:

```
nano <filename>
```

To exit from nano, saving your changes, type Ctrl-X followed by Y. Further options are shown at the bottom of the nano screen, where ‘^’ indicates the Ctrl key.

If you need to know more details about nano or any other linux command, try using the ‘man’ command to look at the *man pages* (manual pages). For example:

```
man nano
```

Man pages can be quite long. Navigate around them as follows:

Command	**Action**
Space bar	Scroll down through the file by one screen
Enter	Scroll down through the file by one line
q	quit the man page
Page Down/Up keys	scroll down/up by 1 screen
Down/Up arrow keys	scroll down/up by one line
/string	search for the next instance of ‘string’, and then
n	move to the next instance of ‘string’
p	move to the previous instance of ‘string’

Setting up

Before you start the exercises below, please execute the following command. This will ensure that you have access to the course resources. You will not need this command outside the course.

```
source /nobackup/proj/training/setup.sh
```

In the sections below, some commands will be listed with an item in angular brackets, e.g. <*username*>. You should replace these items, including the brackets, with your own text.

Part 2. Finding your way around

There are several areas on Rocket where you may keep files:

Name	**Address**	**Access**	**Size**	**Retention Policy**	**Best for**
Home directory	/mnt/nfs/home/<userid> or $HOME	System-wide	40GB quota	None	Small files
Lustre filestore: personal space	/nobackup/<userid>	System-wide	Share of 500TB	Deletion after 3 months without use	Large data
Lustre filestore: project space	/nobackup/proj/<project_code>	System-wide	Share of 500TB	Deletion after 3 months without use	Shared files
TMPDIR, Scratch space	$TMPDIR	Local to compute node(s)	Depends on node: share of 469GB-8.7TB	Exists only during job execution	Temporary and frequently accessed files

By default, all your files and directories are private, even in your project space.

None of the space on Rocket is intended as secure, long-term storage. Your research project should have access to such storage elsewhere, e.g. in the University's research filestore. That filestore, previously known as the Research Data Warehouse, can be accessed from the Rocket login nodes as /rdw. Speak to your local IT officer if you want to set up a share in this filestore.

University filestore (RDW)	/rdw	From login nodes ONLY	1TB cost-free per project	Agreed on project basis	Backups and files not in active use

Your home directory

When you log in, you will initially be working in your home directory, /mnt/nfs/home/<*username*>. To print the name of your current working directory, type:

```
pwd
```

To list the contents of your current directory, type:

```
ls
```

You may find it useful to make a subdirectory where you can put the example files from this course. Type:

```
mkdir hpc_course
ls
```

Your hpc_course directory should now be listed. Change directory to it:

```
cd hpc_course
```

Linux has some shortcuts that are helpful for navigating around your directories

Meaning	**Shortcut**	**Example**
home directory	~	ls ~/hpc_course
current directory	.	source ./myscript.sh
directory above/parent directory	..	ls ../..

You have a 40GB quota in your home directory that you cannot exceed. Check your usage with:

```
quota -s
```

Use your home directory for smaller files, such as job scripts, log files, program source code and small data files. Large data files (many GB) should be placed in your /nobackup directories.

Your /nobackup directories

The /nobackup directory tree is on a fast, 500TB *Lustre* system that is designed for HPC and performs particularly well for parallel access to large datasets. Your project’s shared space is also on Lustre and may contain e.g. software and reference datasets.

Find your personal directory on /nobackup. It may still be empty:

```
ls -ld /nobackup/<username>
```

All Rocket users are members of one or more registered projects, and each project has a directory on /nobackup for shared files. To list your project memberships, type:

```
groups
```

The *rocketloginaccess* group.is shared by all users, while the entry *rockhpc_training* shows that you are currently a member of the *training* project.

View the contents of the *training* project directory:

```
ls -l /nobackup/proj/training
```

You may already be a member of other projects too. Each of them will have a directory in /nobackup/proj/ that you can access.

Local scratch space

Your home directory and /nobackup are central storage areas accessible across the Rocket cluster. There is also local space on each node, which can be used for temporary files and for more efficient I/O during jobs. This space is only accessible to that node and is always called /scratch. When you run jobs on the compute nodes, a subdirectory will be created on /scratch for each job and can be referred to using the environment variable $TMPDIR.   Use this rather than the top level /scratch: it helps avoid conflicts between jobs and allows the automatic removal of files when jobs end.

Use of $TMPDIR can make a significant difference to your computations' speed, for example if you:

- Write temporary files that are used only while a computation is running
- Read from or write to a file numerous times during a computation
- Access numerous small files

On the login nodes, $TMPDIR is set to the top level /scratch, which can contain files belonging to any user. View the local storage on the login node where you are working:

```
ls $TMPDIR
```

Part 3. Software and modules

Rocket has an extensive set of applications installed. Compilers and most specialist applications on Rocket are available as modules, which must be loaded before they can be used. Use of modules helps avoid conflicts between different versions and applications and helps you to reproduce computations knowing that you are using exactly the same software versions.

Matlab is a common mathematical application that is installed on Rocket. Type:

```
which matlab
```

Matlab is not found because the Matlab module is not yet loaded.

In order to load a module you will need to know its name. The following command displays all available modules, but the list is very long:

```
module avail
```

Look for Matlab explicitly by typing:

```
module avail matlab
```

Note the (D) in the output, which indicates the default version. To load the default version (note that the application name in this command is case-sensitive):

```
module load MATLAB
```

Or to load a version explicitly:

```
module load MATLAB/2017a
```

Check that the Matlab command is now available:

```
which matlab
```

Loading a module automatically loads any other modules that it depends on. Show all the modules you have loaded with:

```
module list
```

Unload just the Java module:

```
module unload Java
```

Unload all modules:

```
module purge
```

Apart from 'module avail', most module commands are case-sensitive. What happens if you type the following command?

```
module load matlab
```

Some applications are difficult to search for, such as the statistical computing package R:

```
module avail R
```

In this case it can helpful to try 'module spider' instead. The command also displays information about applications:

```
module spider R
```

Alternatively you can use 'module avail' with the option --*redirect*, which allows you to pipe the output to another command. The following example lists all modules whose names begin with ' R'. It may also include other entries if your command window is wide.

```
module --redirect avail R |grep ' R'
```

If you cannot find an application that you need, you can either ask the IT Service to install it centrally or you can build it in your own (or your project's) space. Rocket has modules for Intel, PGI and GNU compilers, along with an extensive list of libraries and other components. You are responsible for complying with any licence terms and conditions associated with software that you install.

Now return to the course slides

Part 4. Running jobs

So far you have worked on Rocket's login nodes, which are intended for light, interactive tasks. Computations that need significant resources (CPU, memory, I/O) should not be run on the login nodes. Instead, they are submitted to the resource management application *SLURM* as 'jobs'. SLURM adds them to the queue of jobs and runs them on a compute node when resources become available. SLURM is configured so that it aims to make access fair for all users.

This section of the course demonstrates how to submit some different types of job to SLURM. Before you begin, return to the hpc_course directory you created earlier:

```
cd ~/hpc_course
```

Exercise 1 - Running a batch job using SLURM

Create this simple script using a text editor and call it job1.sh

```
#!/bin/bash
#
#SBATCH -t 00:05:00
#SBATCH -p short
#
module load Python
python --version
date
sleep 60
```

The first line of this script tells linux to run the script using *bash*. *Bash* is the type of unix that you use at the command prompt on Rocket.

```
#!/bin/bash
```

The two lines containing only a # are interpreted as comments. You could add some text after the #.

The *#SBATCH* line gives SLURM directions on how to run the job. The format of these lines is:

```
#SBATCH -t <dd-hh:mm:ss>
#SBATCH -p <partition>
```

So the example script has a time limit of 5 minutes and it will be submitted to the *short* partition. Note that the #SBATCH lines must come *before* the executable commands. Any #SBATCH lines after an executable line will be ignored.

The executable lines in the script do the following:

- load the Python module
- print the version of python
- print the date
- ‘sleep’ for 60 seconds

Submit the script to SLURM so that it runs on a compute node:

```
sbatch job1.sh
```

Note the job ID that is displayed and check if this job is queueing or running:

```
sacct
```

If you ran the executable commands from job1.sh at the Rocket command prompt, any output would be displayed on the screen like this:

```
Python 3.6.3
Tue Jan 22 13:46:47 GMT 2019
```

When you submit a script using sbatch, it will run completely separately from your interactive login. Any output that would normally be displayed on the screen will be written to a file instead, by default called slurm-<*jobID*>.out. If a job fails, the reasons can often be found in this file. When your job has completed, check the contents of the SLURM output file:

```
more slurm-<jobID>.out
```

Exercise 2 – Simple troubleshooting

Using script job1.sh as a basis, create three new jobs that each have one of the following errors (marked in bold). Submit the new jobs to SLURM so that you can see how the errors are handled. Change:

```
job2.sh:   sleep 60 to ssleep 60
job3.sh:   #SBATCH -t 00:05:00 to #SBATCH -t 40-00:05:00
job4.sh:   #SBATCH -t 00:05:00 to #SBATHC -t 40-00:05:00
```

Note that the maximum job length in the default queue is 2 days.

You should find that:

- job2.sh fails. The file slurm-<*jobID*>.out contains an error message.

- job3.sh queues indefinitely. Use *squeue* to see why, and then cancel this job:

```
squeue -u <username>
scancel <job_ID>
```

- job4.sh runs to completion, even though it has contains two errors. This is because the mis-spelled #SBATCH instruction is ignored by SLURM and treated as a normal comment line. An error like this could cause a job to fail because of insufficient resources.

The *sacct* command displays information about current and past jobs. It can help you pinpoint problems and understand your jobs' resource needs. The command's many options are listed on the sacct man page. By default, it will show all the jobs you have run today. Try these examples:

```
sacct
sacct -j <jobID> -o Elapsed,TimeLimit,AllocCPUs,MaxRSS
```

Use *sacct* to show that the time limit specified in (c) was not recognised.

Exercise 3 – Specifying job requirements

It is important that you understand the resource needs of your jobs. Jobs may crash if they are allocated insufficient resources, while jobs that over-request resources or that are submitted to an inappropriate partition can make very poor use of the Rocket hardware. High-memory jobs should be run in the *bigmem* partition.

Specifying your job's requirements carefully can also help with scheduling. A job that requests less than the default 2-day time limit may be able to 'backfill' space that is waiting for a large multi-core job to start.

SLURM has numerous options that allow you to specify the needs of your job. They can be typed on the command line and included in job scripts. Some common SLURM options are listed at the end of this document; more options are on the sbatch man page ('man sbatch') and on the SLURM web pages: https://slurm.schedmd.com/sbatch.html.

Copy job1.sh to job5.sh:

```
cp job1.sh job5.sh
```

Edit the script to add extra #SBATCH options that:

- request 10GB memory
- submit the job to the 'bigmem' partition
- send you an email when the job starts, fails or completes

Your new script might look something like this:

```
#!/bin/bash
#
#SBATCH -t 00:05:00
#SBATCH --mem-per-cpu=10G
#SBATCH -p bigmem
#SBATCH --mail-type=ALL
#
module load Python
python --version
date
sleep 60
```

Submit this job to SLURM, make sure that it has run, and check the results in the slurm-<*jobID*>.out file.

SLURM options can be written into your job scripts as above or typed on the command line. The example above is equivalent to:

```
sbatch –t 00:05:00 --mem-per-cpu=10G -p bigmem \
  --mail-type=ALL job1.sh
```

Exercise 4 – Associating jobs with different projects

If you belong to just one Rocket project, your jobs will automatically be associated with that project. However, if you are a member of more than one project, you will need to make sure that your jobs are associated with the correct project. This will allow you to benefit from any priority boost awarded to a project and will help us understand how different research areas use HPC.

SLURM refers to projects as accounts. To list your project memberships and display your default, type the following and look at the 'Def Acct' and 'Account' columns:

```
sacctmgr show user <username> withassoc
```

If you have more than one project listed, change your default by typing:

```
sacctmgr modify user <username> set defaultaccount=<project>
```

Submit one of your existing job scripts and use the `sacct` command to show that the job is associated with your new default.

Your default is overridden by any directives in your job script or on the command line. Edit the job script you used above to specify the *training* project:

```
#SBATCH –A training
```

Submit this as a job and use `sacct` to check that the job has been associated with the *training* project.

Exercise 5 – Running many jobs at once: job arrays

Some types of work require the running of many separate jobs that are very similar. A ‘job array’ can be an effective way to run sets of jobs and is handled well by SLURM.

Create the following script, job_array.sh, which runs an array of 10 separate jobs. The SLURM environment variable *${SLURM_ARRAY_TASK_ID}* can be used to control how each element of a job array runs, e.g. by using a different input file.

```
#!/bin/bash
#SBATCH -t 00:05:00
#
#SBATCH --array=1-10
echo This is element ${SLURM_ARRAY_TASK_ID}
mkdir test_${SLURM_ARRAY_JOB_ID}_${SLURM_ARRAY_TASK_ID}
```

While the array jobs are queueing, they appear as a single item in the queue. Notice that each job produces a separate SLURM output file, named slurm_<*jobID_element*>.out

Submit your array job using *sbatch* and check that your job has produced a set of directories.

Now resubmit an edited job in which the array is limited to running 3 concurrent jobs. This can be useful to control your resource usage, e.g in the bigmem partition:

```
#SBATCH --array=5-15%3
```

Arrays can also be specified with explicit job numbers, e.g.

```
#SBATCH --array=1,39,47,94
```

Note that the same resource limits apply to every element.

You may have up to 10,000 separate jobs or array jobs with a total of 10,000 elements in the system, ie. queued, running or just finished. Array jobs are can run on up to 528 cores at once.

Although Rocket allows you to submit many jobs at once, very short jobs (<20 minutes) should be bundled if possible so that each submitted job runs several tasks. This reduces the scheduling overheads for each task.

SLURM allows you to create chains of array jobs, in which the scheduling of each element is, for example, dependent on the outcome of the matching element in a previous array. This is a popular way to construct task pipelines. SLURM support for job arrays is described more fully at https://slurm.schedmd.com/job_array.html

Exercise 6 – Running a multi-threaded job

In this exercise you will run a multi-threaded program. Multi-threaded programs can run on multiple CPU cores **within one node**. This type of parallelisation is relatively easy to achieve and is used by numerous applications.

Copy an example program to your *hpc_course* directory:

```
cd ~/hpc_course
cp /nobackup/proj/training/parallel_examples/omp_example .
```

This program executes a loop to populate a simple array. It is parallelised using OpenMP. The source code is in /nobackup/proj/training/parallel_examples/omp_example.f90

Write a job script that will run *omp_example* on 8 cores. You will need to:

- request 8 cores with shared memory
- load the GCC module (for the GNU compilers)
- run the program in parallel

Your script might look something like this:

```
#!/bin/bash
#
# simple OMP job script for SLURM
#
#SBATCH -t 00:05:00
#SBATCH -c 8

module load GCC

./omp_example
```

Note: OpenMP uses an environment variable *$OMP_NUM_THREADS* to set the number of threads it creates. On Rocket this defaults to the number of cores allocated in the line ‘#SBATCH -c 8’.

The example program executes a loop 8 times. How would if affect the job’s performance and the use of compute resources if you ran it on 4, 7, or 9 cores? How would you choose the number of cores for your own multi-threaded job?

Exercise 7 – Running a multi-node MPI job

The message passing interface (MPI) allows the creation of very large computations, sometimes spread over many nodes. MPI applications run multiple instances of the same program, each with a separate memory allocation. Each instance executes its own portion of the overall calculation and communicates with other instances using MPI 'messages'. For codes that use MPI only and do not use OpenMP, it is usual to run one MPI instance ('task') per core.

MPI programs are normally run using the 'mpirun' command. SLURM has an equivalent command 'srun' that you should use on Rocket as it recognises your SLURM resource allocation.

Copy a simple example MPI program to your *hpc_course* directory:

```
cp /nobackup/proj/training/parallel_examples/mpi_example .
```

Write a job script to submit this job to the *short* partition. The job should run on 88 cores across 2 nodes. Your script will need to:

- Specify the partition
- Specify the number of MPI tasks
- Specify how many tasks to run on each node
- Load the OpenMPI module
- Use srun to run the program

Your final script might look something like this:

```
#!/bin/bash
#
# simple MPI job script for SLURM
#
#SBATCH -t 00:05:00
#SBATCH --ntasks=88
#SBATCH --tasks-per-node=44
#SBATCH -p short
#
module load OpenMPI

srun mpi_example
```

Submit this job using *sbatch.* Use *squeue* or *sacct* to check on the job's progress.

Note: SLURM's default scheduling can cause MPI jobs to be spread across many nodes, which is not efficient. Use the options above to make sure that this does not happen. For large jobs that fill entire nodes, like the one above, you could also consider the option --exclusive. This ensures that the job does not share nodes with any other jobs, which can improve performance and ease scheduling for other large jobs. Use this option only if your jobs will come close to filling the cores or memory on the nodes in question.

When the job has finished, use 'sacct' to see how long the job took to execute. Remember that SLURM reports on the job's total elapsed time, including the overheads of starting and finishing the job.

Some 'hybrid' codes use a combination of OpenMP and MPI. For example, an application might be designed to run 2 MPI tasks on each node, and use 22 OpenMP threads within each MPI task. Jobs of this type may need to be constructed carefully to match them to the nodes' hardware and should be monitored carefully for performance issues.

Exercise 8 – Running an interactive job

The login nodes can be used for light interactive work, but they are shared by many users and it is easy for one user to cause problems for others, e.g. if they run memory- or CPU-intensive jobs.

If you do need to run something intensive interactively, SLURM allows you to run an interactive session on a compute node. The job can be specified on the command line with the normal SLURM options, and will start only if there is space for it. Earlier in this course you ran a shared-memory program called *omp_example*. Try running it interactively instead:

```
srun -A training -t 00:05:00 -c 8 omp_example
```

It is also possible to start an interactive shell on a compute node. This can be useful for development work and testing. **When you have finished with an interactive shell, make sure you type 'exit' to release the resources for other users.** Type:

```
srun --pty bash
```

Once the interactive resource allocation has been granted for your shell, check that you are now working on a compute node:

```
hostname
```

Exit from the interactive session and return to your session on a login node with:

```
exit
```

Part 5. More about data

Although the heart of Rocket is its computing power, data are at the heart of the work done on it. This section contains some advice on how to manage your data and use Rocket's storage well.

Copying data to and from Rocket

There are multiple ways to copy data to and from Rocket. The simplest way is to drag+drop, while linux commands such as *rsync* provide powerful tools to keep up-to-date copies of your work.

Before you copy data between systems, consider whether you can compress them to reduce their size. Linux tools for this include *zip*, *gzip*, *bzip* and *xz*, and details are given on man pages. It can also help to *tar* the contents of a directory into a single *tar file*.

Create a tar file *jobscripts.tar* containing all the job scripts you have created on this course. The option *–z* means that the file will compressed automatically:

```
tar cvzf jobscripts.tar job*.sh
```

View the size of this file:

```
ls -l jobscripts.tar
```

It would be more efficient to transfer this file than to transfer the individual scripts that are in it.

To list the contents of the *tar* file:

```
tar tvf jobscripts.tar
```

To extract all the contents of the *tar* file:

```
tar xvf jobscripts.tar
```

To extra just one item:

```
tar xvf jobscripts.tar job1.sh
```

If you prefer to keep your files separate, you could compress them individually. This would allow you to make more use of *rsync* to manage your data, for example.

```
gzip job1*.sh
```

Use *ls -l* or *du -hs* to check whether compression makes a difference.

The table below lists some options for transferring files between systems. Additional tools (especially rsync) are available on Windows to users who have Cygwin or WSL installed.

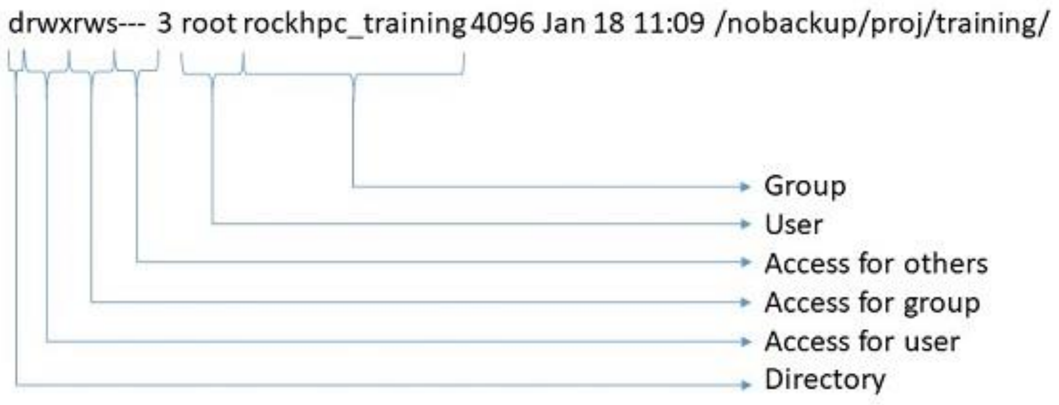
Copy to/from Windows	**WinSCP**: drag+drop between panels **MobaXterm**: drag+drop between the 'sftp' panel and Windows File Explorer
Copy to/from linux/Mac	From Rocket command line: **scp** *myusername@myserver.ncl.ac.uk*:*myfiles* \\ */nobackup/myusername/* **rsync** –av –e ssh /*mydirectory1*/ \\ *remoteuser@myserver.ncl.ac.uk:/mydirectory1/* From remote machine: **scp** *myfiles* \\ *myusername@rocket.hpc.ncl.ac.uk:/nobackup/myusername/*
Copy to/from /rdw	**cp** */nobackup/myfiles* /rdw/*share_path/mydirectory* **rsync** –av */nobackup/mydirectory/* /rdw/*myshare_path*/
Download from web	**wget** <URL>

Setting permissions in a project space

The project directories /nobackup/proj/<*projectcode*> are intended to enable sharing of data and applications. However, group access to your files in this space is not granted by default so you must allow it explicitly.

The command below displays information, including the access permissions, for the directory /nobackup/proj/training:

```
ls -ld /nobackup/proj/training
```


drwxrws--- 3 root rockhpc_training 4096 Jan 18 11:09 /nobackup/proj/training/
Group
User
Access for others
Access for group
Access for user
Directory

Notice that the group associated with this directory is called *rockhpc_training*. Each Rocket project is associated with a linux group, called *rockhpc_<project_code>*, which controls access to the project directory. All project members are members of this group.

Change directory to this shared directory:

```
    cd /nobackup/proj/training
```

Use a text editor to create a file in this directory (name it according to your username to avoid conflicts), and check the access permissions for this file:

```
    ls -l <filename>
```

This should show that you have read and write access to your file, but the group has none.

Change the permissions so that other members of the group can read the file:

```
    chmod g+r <filename>
    ls –l <filename>
```

For directories and executable programs, you will need to add permission to execute:

```
    chmod g+rx <file or directory>
```

If you are already a member of another Rocket project and know its project code, check that you can view the contents of that project directory and that you can write a test file there that is accessible to other members of the group.

Most linux files and directories have just read, write and execute (rwx) permissions set. The ‘s’ in the group permissions for /nobackup/proj/training sets the group automatically to *rockhpc_training* for anything *created* here. However, take care if you move or copy a file into a project directory from elsewhere: if you *copy* a file to this directory from elsewhere, it will inherit its group-ownership from the project directory, but if you *move* it, it won’t.

Create a file in your personal /nobackup directory /nobackup/<*username*> and move it to /nobackup/proj/training. Check the ownership and access permissions and use ‘chgrp’ to change a file’s group ownership:

```
cd /nobackup/<username>
nano <filename>
ls -l <filename>
mv <filename> /nobackup/proj/training/
cd /nobackup/proj/training/
ls -l <filename>
chgrp rockhpc_training <filename>
ls -l <filename>
```

When you have finished this section, remove everything you have created in /nobackup/proj/training

```
rm <filename>
```

Note: to change your default group to *rockhpc_<projectcode>* and allow members of your project to access all your files, you could use e.g. ‘newgrp rockhpc_<*projectcode*>’ and ‘umask 027’. This would change the rights to ALL new files and directories. However, your home directory and personal directory on /nobackup are both private at their top level, so any files or directories created in those locations will still be private. Files outside these locations, e.g. temporary files, might not be private.

Lustre-specific commands

Some common linux commands are not ideal on a Lustre filesystem and can be extremely slow, especially if you have a large number of files or directories to search. Try these Lustre-specific replacements:

To check on your overall disk usage on /nobackup, replace

```
du -ks /nbackup/<username>
```

with

```
lfs quota /nobackup
```

To search for a file using find, replace

```
find /nobackup/<username> -name <filename> -print
```

with

```
lfs find /nobackup/<username> -name <filename> -print
```

About striping on Lustre

A key reason for Lustre's HPC performance is its ability to stripe large files. This causes them to be distributed across several storage devices within the Lustre system. Striping can give faster I/O and is also a more efficient way to store large files, as it helps balance the load across storage devices.

The striping configuration can be set for a file or a directory. A directory's contents will inherit the directory's striping by default. The default stripe count on Rocket is 1. This is the most appropriate setting for small files. Consider changing this if you work with files larger than 10GB.

'cd' to /nobackup/proj/training and view the current striping of that directory:

```
cd /nobackup/proj/training
lfs getstripe –d .
```

Check the striping of the setup.sh file:

```
lfs getstripe setup.sh
```

Striping can be set with the 'lfs setstripe' command. To create a new file that is striped, either:

a) Use lfs setstripe as below to set the striping for a new, empty directory that will store the file. When you write a file in this directory, it will inherit the directory's stripe configuration by default.
b) Use lfs setstripe to create an empty file with the striping you want, then write to this file as normal.

Create a new striped directory:

```
lfs setstripe –c stripe_count <directory_or_filename>
```

The option –c gives the number of storage devices to stripe across. The maximum is 12, to match the 12 storage devices in /nobackup. '-c -1' stripes across all available storage devices. Adding e.g. '-S 1G' will set the stripe size. Note that over-striping a small file will increase I/O overheads and could affect performance.

A file will **not** be restriped if you move it to a striped directory. You must copy it instead.

Create an unstriped file in a location of your choice. Then use one of the 'lfs setstripe' commands above and copy the file to the new, striped location. Check that your new file is striped correctly.

Note: if your software application expects a file to be in a particular location, use a symbolic link between that location and the real (striped) location. The following example checks that the copy has completed correctly before deleting the original:

```
lfs setstripe –c -1 –S 1G striped_dir
cp unstriped_dir/bigfile striped_dir
md5sum unstriped_dir/bigfile striped_dir
rm unstriped_dir/bigfile
ln –s striped_dir/bigfile unstriped_dir/bigfile
```

You are now at the end of the course. Before you leave, remove any files you have created in /nobackup/proj/training and cancel any jobs that are still queueing. Anything you leave will be deleted.

Part 6: Further resources

Newcastle HPC help and information

Rocket HPC Service web pages, including sample job scripts and troubleshooting tips: http://ncl.ac.uk/itservice/research/hpc

Research data management: https://research.ncl.ac.uk/rdm/

IT service desk: http://nuservice.ncl.ac.uk

UK HPC resources

Courses and information from the national supercomputing service (ARCHER) http://www.archer.ac.uk/training

Other UK and international resources and training http://www/hpc-uk.ac.uk

SLURM

Common SLURM options:

Option	**Meaning**	**Example**
-A	SLURM account (project)	-A training
-p, --partition=	partition	-p bigmem
--mem=	Memory in M/G/T per node	--mem=10G
--mem-per-cpu=	Memory in M/G/T per core	--mem-per-cpu=10G
-t, --timelimit=	Time limit	-t 0-02:30:00
--mail-type=	When to email user	--mail-type=ALL
-c, --cpus-per-task=	Number of cores (per task)	-c 22
-n, --ntasks=	Number of tasks	-n 528
--ntasks-per-node=	Number of tasks per node	--ntasks-per-node=44
--exclusive	Need whole node(s)	--exclusive
--array=	Job array	--array=0-9999%88

SLURM maintain a website https://slurm.schedmd.com/ with a wealth of information for users. There is also a summary of SLURM commands, options and environment variables at https://slurm.schedmd.com/pdfs/summary.pdf

Lustre

The ARCHER site has guidance on how to improve your I/O on Lustre:
http://www.archer.ac.uk/documentation/data-management/lustre.php

The University of Tennessee has a helpful guide to Lustre striping:
https://www.nics.tennessee.edu/computing-resources/file-systems/lustre-striping-guide