Using BlueCrystal

COMS30005: An Introduction to High Performance Computing

September 28, 2017

This handout is designed as a cheat-sheet, to help with using BlueCrystal Phase 3, the University of Bristol's supercomputer. It contains key commands you will need to use BlueCrystal. There are also some helpful tips and advice for best practice, but these are not required. You will need to be able to use BlueCrystal as part of this course in order to complete the assignments.

To use BlueCrystal you will need to use a terminal. OS X and all Linux distributions come with one. On Windows you can try 'Windows Subsystem for Linux'[1].

Other BlueCrystal guides are available at `https://www.acrc.bris.ac.uk/acrc/resources.htm`.

Remember, BlueCrystal is a shared resource for the whole University, and as taught students you get access to a limited portion. Be mindful of other users and be good HPC citizens.

1 Connecting to BlueCrystal

You must be *inside* the University firewall to access BlueCrystal directly. This means you are using a lab machine, or are on Eduroam.

```
ssh username@bluecrystalp3.bris.ac.uk
```

1.1 Change your password

Run this on BlueCrystal:

```
passwd
```

Follow the prompts for typing in your current and preferred new password.

1.2 X forwarding

You might want to use graphical tools remotely. Add the `-X` flag when connecting:

```
ssh -X username@bluecrystalp3.bris.ac.uk
```

Note that to use X forwarding on MacOS you might need to install XQuartz and on Windows you might need to install Xming.

1.3 Outside the firewall (helpful, not required)

You can get inside the firewall by connecting via `snowy` first.

On your own machine, you could create a ssh alias which bounces through `snowy` automatically.

```
# ~/.ssh/config
Host bluep3
    HostName bluecrystalp3.bris.ac.uk
    User username
    ProxyCommand ssh -X username@snowy.cs.bris.ac.uk nc %h %p 2> /dev/null
```

You can then connect to BlueCrystal as:

```
ssh bluep3
```

Note that now when you connect to BlueCrystal via proxy you will need to enter your password for `snowy` first and then the password for BlueCrystal Phase 3.

[1] `https://msdn.microsoft.com/en-gb/commandline/wsl/about` and `https://blogs.windows.com/buildingapps/2016/03/30/run-bash-on-ubuntu-on-windows/`

1.4 Passwordless access (helpful, not required)

You can set up SSH keys so that you can connect to BlueCrystal without typing your password in every time.

1. Make a new key on your machine.

   ```
   ssh-keygen
   ```

2. Copy the contents of the *public key* file to BlueCrystal's `~/.ssh/authorized_keys` file. Run this on your local machine:

   ```
   ssh-copy-id username@bluecrystalp3.bris.ac.uk
   ```

   If you are using `snowy` as a proxy you can also copy the public key there so that you do not have to type in the `snowy` password. Run this on your local machine:

   ```
   ssh-copy-id username@snowy.cs.bris.ac.uk
   ```

   If your system does not provide `ssh-copy-id`, you may have to copy the public key manually. We recommend copy/paste and **not** `scp` as this file might already contain important keys.

2 Moving files

Use the `scp` command to move files between your workstation (laptop/lab machine) and BlueCrystal. If you set up an SSH alias you can replace the remote URL with your alias.

2.1 Workstation TO BlueCrystal

```
scp file.txt username@bluecrystalp3.bris.ac.uk:~

scp -r directory username@bluecrystalp3.bris.ac.uk:~
```

2.2 BlueCrystal TO Workstation

```
scp username@bluecrystalp3.bris.ac.uk:~/file.txt .

scp -r username@bluecrystalp3.bris.ac.uk:~/directory .
```

2.3 Common mistakes

- Don't forget the colon `:` or the period/full stop `.` characters
- The order is important
- These must be run in your **local** terminal, not on BlueCrystal

3 Editing text

Use whichever text editor you prefer. Nano, vim, emacs and gedit are our suggestions, and they are all available on BlueCrystal. You are expected to be able to open, edit and save files.

3.1 Nano

Nano is a simple, command line text editor.

3.1.1 Opening a file

```
nano filename
```

3.1.2 Saving a file

```
<Ctrl-O>
<Enter>
```

3.1.3 Exit

```
<Ctrl-X>
```

3.2 Gedit

This is a graphical editor so connect to BlueCrystal with X forwarding (see §1.2). Open a file with:

```
gedit filename
```

If you get this error, `Gtk-WARNING **: cannot open display:`, you forgot to connect with X forwarding.

4 Modules

BlueCrystal has lots of extra software available, available through Environment Modules. You will need to load the modules you need in your job submission script so the job runs correctly. Or you can put them into your `.bashrc` so they are loaded when you log in and for **every** job you run.

4.1 View available software

```
module avail
```

4.2 Load/unload some software

```
module load languages/gcc-4.8.4
module unload languages/gcc-4.8.4
```

4.3 List loaded modules

```
module list
```

4.4 Loading modules by default

If you want certain modules loaded every time you log in or start a job, you can add `module load` commands to your `.bashrc`. Open the file in a text editor and add the commands at the very end of the file.

```
# ~/.bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions
module load languages/gcc-4.8.4
```

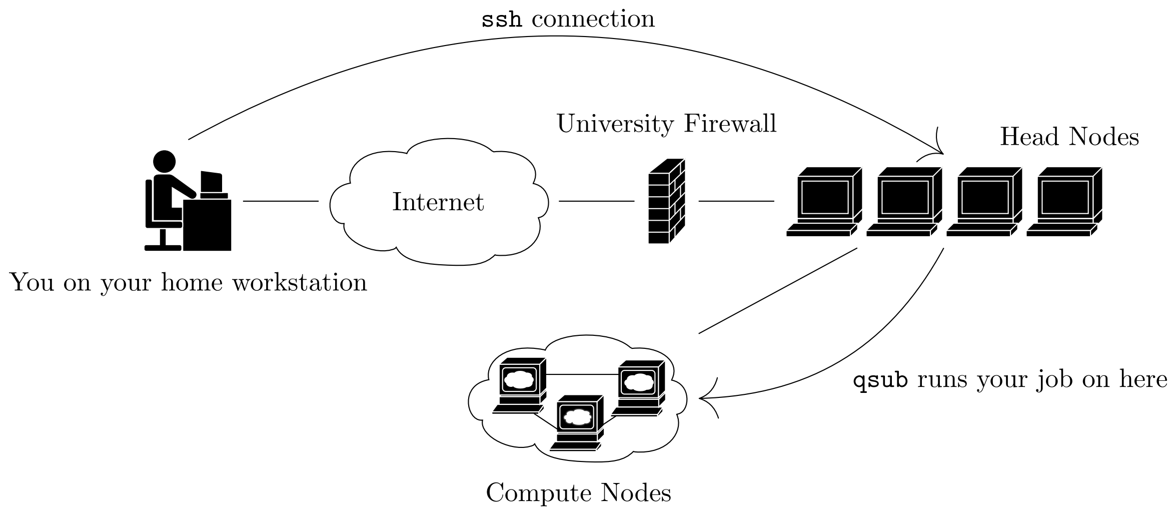
4.5 Searching modules

```
module avail 2>&1 | grep intel
```

5 Using the queue

BlueCrystal Phase 3 is made up of 4 headnodes and 341 compute nodes. **Do not run code on the headnode**, it slows it down for everyone else (**very** short testing runs are fine).

Your jobs may wait at peak times, so you would be wise to not leave everything the last minute. Expect BlueCrystal to go down during the term, for maintenance or extraordinary circumstances like power cuts.



5.1 Submitting jobs

5.1.1 Interactive jobs

Get some time on a compute node where you have bash access and run commands manually. Useful for debugging. You submit the job and wait until the job goes through.

```
qsub -lnodes=1:ppn=16 -qteaching -I
```

5.1.2 Batch jobs

You can write a script to run commands for you, and submit it to the queue for later execution. You can then log out, so this is useful for long running jobs. We provide lots of examples of these in the exercises.

Important:

- Do not submit lots of small jobs to the queue, run the binary many times in the script.
- Do not submit lots of jobs to the queue without a 'sleep' command in between of at least a few seconds.

An example job submission script should contain in the minimum:

```
#!/bin/bash
#PBS -qteaching
#PBS -lnodes=1:ppn=16
#PBS -lwalltime=00:15:00
#PBS -N myjob

cd $PBS_O_WORKDIR

./a.out
```

Submit the job script to the queue:

```
qsub script.sh
```

5.2 View your submitted jobs

```
qstat -u $USER
```

The `S` column gives the job status. `R` = running. `C` = complete. `Q` = queued.

Important: Do not use `watch` to check the status continually. This slows down scheduling for everyone.

5.3 Deleting jobs

```
qdel jobid
```

6 Version control (helpful, not required)

We **strongly recommend** you use version control when completing your coursework. You will be making lots of code changes, and being able to roll back is very helpful. We use `git` for this, but you can use whatever you know. More detailed tutorial at `https://git-scm.com/docs/gittutorial`.

We won't be providing git support.

Warning

Do not put your coursework **publicly** on GitHub. It is *your* responsibility to protect your code. If someone else copies your code, you are *both* reported for plagiarism.

GitHub offers free, *private* repositories for Students. Private repos are fine to use.

6.1 Starting a repository with git

1. Change to your code directory.

   ```
   cd my-coursework
   ```

2. Initialise a new repository.

   ```
   git init
   ```

3. Add your files.

   ```
   git add .
   ```

4. Your first commit

   ```
   git commit -m "Initial commit"
   ```

6.2 Working with git

Change your code a little at a time. Compile and test your code between changes. We recommend you commit your changes to git whenever you've completed a change.

6.2.1 List changed files

```
git status
```

6.2.2 Commit changes

1. Add a changed file to git

   ```
   git add file.c
   ```

2. Commit the change

   ```
   git commit -m "message explaining what you did"
   ```

6.2.3 View history

View your commits:

```
git log
```

To look at what is included in a commit:

```
git show <hash>
```

6.2.4 Undoing changes

You've made a load of changes to a file, and you just need to roll back to the last committed version.

```
git checkout filename
```