Table of Contents

SBCS-Informatics

This is a README explaining how to use the Apocrita HPC resource from an SBCS perspective, it contains:

- Instructions on how to submit jobs and use Apocrita generally
- An introduction to the command line and Unix
- A slightly more advanced section with tips and tricks for the more seasoned user who has gotten to know Apocrita and Unix better

Click on the menu on the left to navigate through the website. This document is also available as a single pdf download.

Contributing

There are two main ways of contributing content or asking questions about this document.

1. Using GitHub, clicking the Edit link will take you to the repository of this document. There, you can either make a GitHub issue or a pull request.
2. You can find contact details here if you would rather send an email.

Using Unix and Apocrita

This section will help new users to get started using Apocrita. Here you can find information regarding:

- How to log in
- How to submit jobs
- Handy information on the system
- Common Unix commands and how to use them to navigate

The information here can also be seen as the first place to look when a question about Apocrita comes up. If there are any unanswered questions after that feel free to create a GitHub issue or use the contact page.

Much of this information has been gathered from the official Apocrita docs, you can find them here.

1. What is Apocrita?

Cluster Structure

Apocrita is a High Performance Computing (HPC) cluster that consists of many interconnected computers, called **nodes**. The user logs into one specific **head node**, from which the rest of the cluster can be accessed. On Apocrita these are called **frontends**. Importantly, ***no jobs are run directly on the frontends***. Instead, whenever the user wants to run a job, the details of the job is submitted to a software that handles scheduling of work on the cluster and it is dispatched to a work node when one is available. There are also **utility servers** which the user can interactively log in to. More information on utility servers can be found below and in the advanced section.

Apocrita and all the other peripheral machines are run by the ITS Research team. They have a documentation site available here.

Nodes

- 150 normal use nodes each with two 6-core processors, i.e. 12 cores, and 24GB RAM
- 11 so called **fat nodes** each with four 12-core processors, i.e. 48 cores, and 512GB RAM. These nodes are used for heavier calculations, in particular ones which require a lot of memory.

For more information on the specifications on Apocrita nodes, see the advanced section.

Utility servers

There are a number of very specific so called utility servers that are part of the HPC resources but not the general Apocrita system. Users can SSH to these machines and use them interactively for their analysis. These servers were all purchased separately with different funding, keep this in mind as groups who have contributed to the acquisition of these machines have priority. With that said, feel free to use these resources when they are available, you can find more information in the advanced section.

Galaxy

There is a Galaxy server set up on Apocrita through which you can run some general (and some more specific) analyses in a graphical web interface. For more information on Galaxy, see the specifications here.

Data

Each user has a specific account on Apocrita and have a data quota of 50GB in their home directory. Often this is nowhere near enough to run the analysis needed, therefore there is a so called **scratch** space for temporary files. This is where most people do their work but it's important to note that that **scratch area is not backed up**. In the future a time-limit on files may be implemented, automatically deleting old files in the scratch space.

Home

- Is tiny
- Is backed up
- Is for your personal scripts and installations, small files you use often and want to keep safe

Scratch

- Is **NOT** backed up
- Is communal, you're not the only user here - keep this in mind and write only in your directories

- Is temporary - **NEVER** store files on a scratch space, they will be deleted sooner or later

Available scratch spaces include a small personal space in `/data/scratch/btw000` and a larger communal space in `/data/omicsScratch/`.

A new scratch space is in the works which will include automated deletion protocols. These protocols will be implemented on all communal scratch spaces so please keep this in mind and be aware.

Lab specific storage

- Each lab has a 1TB space, some labs have purchased additional storage
- Ask your supervisor about access to your lab space
- This is usually for medium term storage of data etc., but each lab handles their area internally

The Hive Archive

There is an archive available for long term storage of data. It is called Hive and can be accessed from anywhere on a Queen Mary network. It is only available with ssh key authentication, so in order to log in you need to create one or more ssh keys. You need to create a key for each machine youre logging in from, i.e. if you want access from both your desktop computer and Apocrita you will need to create two keys, one on your desktop and one while logged into Apocrita. You cannot copy your own key to the Hive, instead you have to send an email to its-research-support@qmul.ac.uk with your public keys and Apocrita username.

- Log in with your Apocrita username, key authentication only
- hive-master.hpc.qmul.ac.uk
- 120TB storage available
- Backed up by mirroring to an identical setup
- Accessible via SFTP, rsync, scp
- For storing files you cannot delete for one reason or another
- Please remember to **compress** your files before transferring them to the archive

System

Apocrita is running Scientific Linux 6.2.

2. Connecting to Apocrita

Get an account on the cluster

To get an Apocrita account you need to make a request to ITS Research. Once this has been set up you can continue to log in with the details provided.

Logging in from Mac (or other Unix systems)

OS X (and indeed all Linux distributions) have an application called Terminal. On your Mac you can find it in Applications -> Utilities.

1. Open Terminal
2. Enter the following

   ```
   ssh -X [youruserID]@login.hpc.qmul.ac.uk
   ```

   e.g.

   ```
   ssh -X btw000@login.hpc.qmul.ac.uk
   ```

3. Enter your password when prompted (It will not appear as you type, this is by design, your keystrokes are being registered.)

To avoid having to type your password each time you log in You can also set up SSH keys that will be used when you log in from your computer instead of a password. There are instructions on how to set up SSH keys in the advanced section.

Logging in from Windows

The simplest way of doing things on Windows is using a program called MobaXterm.

1. Download the newest version of MobaXterm
2. Create a session
3. Enter the address to Apocrita which is `login.hpc.qmul.ac.uk`
4. You can also enter username and password to save them here
5. Connect and enter your username and/or password when prompted

If you would rather use PuTTY.

1. Download PuTTY
2. In the host name window enter the server address `login.hpc.qmul.ac.uk`
3. Enter the correct port number (22) and check on correct connection type (SSH)
4. Now go to the data tab under connection and in auto-login username box enter your username (e.g. btw000)
5. Go back to the session tab and in the saved sessions box enter a name for this log in (e.g. apocrita)
6. Save!
7. Double click the session name and enter your password where prompted

Display

If you're planning on using any graphical interface, e.g. plotting in R, you need the X11 forwarding flag, `-X`. This instructs the terminal forward display items to your computer, that will allow you to see your plots or other graphical user interfaces.

You might as well always have this option set - to simplify things. MobaXterm uses X11 forwarding by default.

```
ssh -X btw000@login.hpc.qmul.ac.uk
```

Keep your terminal as you log out or lose network connection

When you shut your laptop down the connections it has made with Apocrita (and the rest of the outside world) will be interrupted. This can happen randomly at other times as well and the implication is that anything you were running in your terminal on Apocrita will be shut down too, along with the connection. You will not need to do this if you submit jobs through the scheduling system on Apocrita.

Screen

Usefully, the program `screen` allows you to keep a process running on a server without the need to be consistently connected to the network. It sets up a session that will stay active even if you quit your terminal, you can just connect to the session later and you will find yourself exactly where you left off.

```
screen -ls                  # list all your sessions
screen -r <session_id>      # reconnect to a session
<CTRL>+A, <CTRL>+D          # pressing these two key bindings will detach you from the current screen session
<CTRL>+D                    # <CTRL>+D without <CTRL>+A first will terminate the session instead
man screen                  # manual for screen - explaining every option
```

Disconnect a command from the terminal

When a terminal is shut down, a so called hang-up signal is sent to any process running in that terminal telling it to quit. There is a way you can make your process ignore that specific signal with `nohup`. It is very simple to use; you add `nohup` before your command as such:

```
bwa mem -t 10 -p inputreference.fa inputfasta.fastq > outputfile.sam &              #if this is your command
nohup bwa mem -t 10 -p inputreference.fa inputfasta.fastq > outputfile.sam &         #do this instead
nohup nice bwa mem -t 10 -p inputreference.fa inputfasta.fastq > outputfile.sam &    #it's a good idea to add a nice va
```

An ampersand, &, at the end of a command will send it to the background, however this is not enough as it will still be sent the hang-up signal when the terminal is closed, you need the `nohop` addition.

The last example includes the `nice` command as well. `nice` tells the computer that your process should be run with a lower priority, a good idea if you are leaving your process unattended! This does not mean it will take any longer than normal as long as there is available CPUs on the machine.

Keep in mind that nohup will redirect the output if it is a set to a terminal - use `man nohup` for more information. `nohup` does not prevent other signals and will not make your process unkillable!

Queen Mary
University of London

3. Launching a Job on Apocrita

When you log in to Apocrita you will be accessing a **frontend**. It is very important that you do not simply run your scripts here. These nodes are there to accommodate logins to the cluster, not workload. Instead, there is a scheduling program running on the cluster which takes instructions to your job and distributes the total load over all the worker nodes. You can also use the utility servers interactively by SSHing to them from the frontend.

Interactive use

These modes will log you in directly to a machine, allowing you to test, develop and try different things out in a single session. This is useful if for example you are trying a new tool out and need to test different parameters or if you are unsure about the syntax of the program.

Using the queueing system on Apocrita

`qlogin`

These commands will connect you to one of the worker nodes with the requested resources available for you to use. This will allow you to run your scripts/jobs interactively on a command line. This can be useful if you want to have more freedom in what you are doing than a script allows, but should be avoided for longer jobs as the frontends are rebooted fairly frequently which may kill your job. These commands take the arguments on how what resources you need. For more details, have a look at the ITS Research website

Here is an example of logging onto a machine using qlogin:

```
ssh -X btw000@login.hpc.qmul.ac.uk                    #log in to Apocrita
qlogin -l h_rt=3600                                   #request a session 3600s (1h) long
qlogin -l h_rt=10:0:0,h_vmem=4G                       #request 10h and 4G memory
exit                                                  #just type exit to logout
```

Do keep in mind that the frontends are rebooted reasonably frequently to be able to handle all the traffic on them. For this reason, it's best to keep interactive jobs fairly short. Use the utility servers or `qsub` described below for longer runs.

SSHing to a utility server

frontend5, frontend6 and SM11

Once you have logged into a frontend of Apocrita, all you need to do to log in to one of these servers is to SSH to that particular name. These machines have all been bought by different groups of academics and priority remains with these groups. This only means that you need to check carefully before you start to work here, if it is free everyone is allowed to make use of the resource.

```
ssh -X frontend5
ssh -X frontend6
ssh -X sm11
```

These machines are all different, to find out how and why, head to the advanced section. It is **VERY** important that you do not overload these servers, it is up to the user to figure out what the load on the machine is and determine if there is room for their job. Use commands such as `top`, `htop` `free` etc. to make sure.

If you do not know exactly what this means or how to do it properly, ask your colleagues or set out to find Adrian, the cluster guy.

Non-interactive use

Submitting a job script to the queue

This is the easiest way to run your job on the Apocrita cluster. Even so, it is not as simple as running your command. Remember, don't run jobs on the frontends. This method uses a command called `qsub` which takes a script as input. Inside the script you define the resources your job needs.

You need to write the script with the instructions for your job, below you find the simplest version of such a script. Use this as a template and add options as needed. There are several more options you can add to the header of your script which for example allows for more cores to be used.

```
#!/bin/sh
#$ -cwd             # Set the working directory for the job to the current directory
#$ -V
#$ -l h_rt=24:0:0    # Request 24 hour runtime
#$ -l h_vmem=1G      # Request 1GB RAM
./code                                          # Your code goes here
/data/home/btw000/scripts/my_fave_script.sh     # It could be a separate script
bwa mem -t 10 -p inref.fa infa.fastq > out.sam  # Or it could be a program installed on apocrita
echo "Hello" > world.txt                        # Or just bash code to do whatever
```

After you have written and saved the script you feed it to `qsub` as such

```
qsub job_script.sh
```

For more details, have a look at the ITS research website, or if you feel like you can take it, the qsub man page.

Monitoring your job

Use `qstat` to show the status of all your jobs in the queue on Apocrita, and `qdel` to delete a job from the queue.

To monitor a job running in an interactive session or on a utility server use a program like `top`. It lists all processes currently running on the machine and allows you to sort them by various criteria like CPU or memory usage. There is an arguably better version of the same type of tool called `htop`. You will need to load it with `module load htop/1.0.3` before you can use it though. Both `top` and `htop` have an argument `-u` taking a username so that you can inspect only your own processes if you want, `htop -u btw000` will show only btw000's processes.

4. Basic Unix Navigation

This section contains some extremely basic commands that you can run while logged onto Apocrita (or in your own Unix terminal if you are a Linux/Mac user.) There are much better guides available online if you want to learn about Linux and the command line. One such is Learn Linux the Hard Way]. That link will take you to one which, although long, could be very useful. Some parts of it are even interactive. IBM has a technical library for Linux as well, it's a bit heavier and set up as several tutorials but very informative.

Where am I?

Files are organised in your allocated server space into folders, or 'directories', just like on your regular computer. You can check your current working directory (i.e. where you are) with the 'print working directory' command, `pwd`.

Checking the contents of directory

It's easy to check the contents of the directory you're currently in, using the list command, `ls`. Typing `ls` alone will list the filenames. However, if you require extra information you can add 'flags' after the command to give the computer further instructions:

```
ls          #prints filenames in a list
ls -a       #prints all filenames, including hidden files
ls -l       #'long' list, displays info including permissions.
ls -h       #prints the sizes of files in units you can read
ls -tr      #t for time sorted, r for reverse
```

The above flags (amongst others) can be combined together in one string for convenience. You may use one `-` (to indicate a flag) followed by all required characters, or separate them individually:

```
ls -lhatr          #has the same effect as
ls -l -h -a -t -r
```

This bit will print a list of files with all of the above information.

Changing directories

Navigate through directories using the 'change directory' command, `cd`. It will assume you're looking for a directory name that is within your current working directory

Change directory

```
cd directory_name/
```

Go back "up" one directory:

```
cd ../
cd ../../                   #you can go several at a time
cd ../../anthr_dir/subdir2  #as many up and down as you want
```

Return to home directory

```
cd ~/
```

or just

```
cd
```

> *Note: On Mac, it's possible to click and drag the desired location of a directory or file from the finder (by the icon) to the terminal. Just type* `cd` *followed by a space, click & drag, hit enter. You can do this from any starting point.*

5. Playing with Files

Creating a new directory

The `mkdir` command will create a directory inside your current location.

```
mkdir newdirectoryname
```

Moving files in local machine

The `mv` command will move a file or directory from its current location and place it elsewhere. The syntax is, `mv current_location new_location`. It can also be used to rename files.

If the new location is a file, the file is renamed:

```
mv oldfile newfile
```

or if the target is a directory, the file is moved:

```
mv oldfile newdirectory/
```

Delete files or directories

Delete files with the 'remove' command, `rm`:

`rm filename`

or for a directory `rmdir`, the directory has to be empty:

`rmdir directory/`

You can use `rm` to remove directories with the `-r` option, it removes files in a directory recursively. **Be careful** when using `rm`, once you hit enter, the files are gone. After adding or removing files and directories, you can check and make sure it worked using the `ls` command.

Viewing different parts of an existing file

`less` shows a small portion of the file, `more` shows a larger portion (according to the manual, it will display your file 'one screen's worth of lines at a time'). `head` displays the first ten lines, and `tail` displays the last ten lines. Both `head` and `tail` have several useful options like how many lines to display.

```
less filename.txt         #look through file with less
tail -n35 filename.txt    #print last 35 lines of file
```

Searching for a pattern within a file

The `grep` function searches for a particular pattern of characters. The syntax for the grep function is: `grep pattern file-to-look-in`. `grep` will by default print the whole line in which it found the pattern. There are options to have it print only the

match, line numbers, or just a count of how many times the pattern was found.

e.g.

```
grep "scaffold" genome.fasta    #prints all lines with "scaffold" in them
grep ">" genome.fasta -c        #counts the number of ">" characters in the file
```

6. Copying & Downloading Files

Files can be copied in an around your computer, in and around your space on the servers, and between the two. The basic copy function, `cp`, is for local to local copy (i.e. your within computer or wihtin the server). To go between the two, use `rsync` or the slightly more simple `scp`.

The basic syntax is `cp /path/to/source/file /path/to/destination`.

Between a remote server and local machine

The most reliable and easiest way to copy and sync files between different machines is rsync.

```
rsync -avP ~/my-work username@remote_host:/path/to/destination/
```

This bit of code will copy the directory "my-work" to the destination at the remote host.

To copy files from Apocrita to the Hive, use the following while logged into Apocrita:

```
rsync -avP ~/my-work btw000@hive-master.hpc.qmul.ac.uk:/SBCS-BloggsLab/btw000/
```

rsync has many options, find out more by reading the manual (`man rsync`) or look at an online guide.

scp is very similar, but it doesn't have the syncing capabilities of rsync, it is therefore best used to copy single files. To use `scp` to copy from server to directory on local machine:

```
scp btw000@frontend1.apocrita.hpc.qmul.ac.uk:/home/btw000/myoutput.pdf .
```

> *Note, "`.`" means "current directory".*

Using a graphical user interface (GUI)

Cyberduck (on Mac) or FileZilla (on Windows). These two are GUI programs which offer file-sharing via a straightforward drag-and-drop or by menu navigation.

Both programs have a facility for attaching your account and passwords to them so they'll log in automatically. Make sure you check in the preferences/options/settings to set the file transfer protocol to "SFTP", as that's not usually the default.

1. Open Cyberduck
2. Create a new connection specifying
    - SFTP protocol
    - server login.hpc.qmul.ac.uk
    - username and password
3. Connect
4. Manage your files on Apocrita through the interface provided

> *Note that while these systems are convenient for moving and organising your files, they will not let you schedule jobs or run anything on the server. Alas, it's back to the command line to do anything using the Apocrita computational resources.*

Mounting your Apocrita file system on your computer

You can use some third party software to easily mount your Apocrita file system on your computer, that way you can

manipulate your files in your operating system, and programs on your personal computer will be able to read and write files on Apocrita.

Here is a good guide for every OS.

Downloading files from the web

`wget` is short for "web-get", and will download a target file into your current working directory, e.g.

```
wget http://ftp.gnu.org/gnu/wget/wget-1.5.3.tar.gz
```

Here wget is used to download an old version of itself. It is often much easier to download a new piece of software or data straight to the server using `wget`, instead of first downloading it to your local machine and then `scp`-ing the package over. Just copy the download link of the file when you're browsing and paste it to `wget`.

7. Dealing with Compressed Files

As disk space is always limited and factoring in the large number of users, it is vital that everyone takes care storing their data properly on Apocrita. Always compress and archive files that aren't used.

Files

In genomics it is very common to have big files which contain sequence information or mapping files that can tell software and user where each sequence read fits against a certain reference. These files are essentially text files like any other and are often human-readable (for your convenience). Unfortunately, that means these types of files take up an enormous amount of space, but there are ways to mitigate this issue. Compression really just means that files are manipulated in such a way that information is denser, making them smaller, although this also means that they cannot be viewed/read unless you reverse the process. The simpler the file, the smaller it gets after compression because there wasn't much information to begin with. It is important to **compress anything that you are not currently using**, as this will save a lot of space on the cluster.

> *Remember that some tools and applications are able to work with compressed files. You might not have to unzip that sequence_reads.fastq.gz file you've got.*

Directories

In some cases, it's not a single large file that is causing storage issues. Some programs create a large system of folders filled with lots and lots of tiny files. This can add up quickly and because of the way file systems work there is a "minimum size" that a file can occupy on disk (if you want to know more about block size). The solution to this problem is to make an archive of the directory. In a Linux/Unix environment the most common archiver is a program called `tar`. `tar` was created to handle problems with block size and writes a single new file, often called tarball, containing everything in the directory. This is not compressed so what you often see is compressed tar archives where the tarball has been run through `gzip`, these files often have the extensions `.tar.gz` or `.tgz`. You should do this as well.

Archiving

Use the `tar` command to create, and extract, archives of folders.

```
tar -c directory/ > directory.tar
```

`-c` is for create. This creates a new file called `directory.tar` but the original directory is still there. You can now compress the `.tar` file and remove the original directory.

Compressing

Gzip is the go-to program to use for compressing files on any Unix system. Here is how simple it is to use:

```
gzip file
```

This zips the file up and gives it the .gz extension, note that this replaces the file with the compressed version.

Use `tar` to convert your analysis folder into an archive, followed by `gzip` to compress it making it as small and easy to handle as possible.

Extracting compressed files, e.g. .zip .tar.gz .tgz

The command for unzipping a file depends on the type of archive it is (i.e. its extension). `tar` can decompress several types of files as well, so you do not have to `gunzip` your `archive.tar.gz` file before using `tar`. `tar` handles it all by itself.

```
unzip file.zip          #for .zip
gunzip file.gz          #for .gz
tar -zxvf file.tar.gz   #for .tar.gz
tar -zxvf file.tgz      #for .tgz
tar -jxvf file.tar.bz   #for .tar.bz
```

> *Notable is that a file may have any extension, it is actually just a part of the file name. However, using proper extensions is a way of letting the user know what kind of file it is. When you move, archive and unzip files etc., make sure that you keep correct extensions on your files, or maybe you wont remember how to open them next time.*

8. Available Software

What is installed on Apocrita?

Software installed on the cluster is handled through something called the `module` command. Each tool installed is a module which can be loaded into your session. `module` has a number of subcommands which you can see by running just `module`. These are the most commonly used:

```
module avail                    #prints a long list of available modules
module avail py                 #prints all available modules starting with "py"
module load python/3.4.3        #loads python 3.4.3
module unload python/3.4.3      #unloads python to return to to sweet 2.X
module clear                    #unloads all loaded modules
```

All `module` commands work with tab-completion, that way you can type `module load java` and press TAB twice to see all available java versions.

The only problem with these modules is that they are not always up to date. You may have to ask for a new version to be installed if you want to be on the bleeding edge.

Installing new software

If you need a tool that is not currently available there are essentially two ways to get new software on Apocrita. Either request an installation by ITS Research, or install it yourself in your home directory.

Requesting installation by ITS Research

If you are looking into a piece of software that you think/know other users will have much use of it is best to have it installed for everyone, available through the `module` commands. The easiest way to create an application ticket with ITS Research is by filling in this form or sending an email to **its-research-support@qmul.ac.uk** explaining what software you want installed and providing relevant links to the software webpage.

Compiling software yourself

If you are in a rush, or if the software is experimental, etc. it might be better to compile it yourself and keep it in your home directory.

Software for Unix systems is generally distributed as archives. To install a software package:

Download it (perhaps with `wget`). Then decompress it (perhaps with `unzip` or `tar`). Then check the installation instructions, usually in an `INSTALL` or `README` file. A lot of the time this will involve running a script called `configure` followed by typing `make` in the directory.

Keep in mind that by default most software will try to install in directories in which you do not have permissions to write. In almost all cases there are options to instead install the tool in an alternate location. Use these options to install software in your home directory. Often this includes using the `--prefix` parameter in the `configure` script.

Some readme files are more useful than others. If there is no help in the readme, but there is a file called `makefile` in the directory, just type `make` and that should work. What happens usually in this case is that the compiled software is created in the same directory.

Queen Mary
University of London

9. R

You can run R in the Unix command line. To do this, simply type `R`. If you have X11 forwarding on in your ssh session, using the `-X` option, you will be able to see graphical devices that R opens when you for example plot something

This essentially turns the command window into the R console you're familiar with. From here, you can do all of the same things as you can in regular R. There are a few things worth noting:

Reading files

Alas, the command line is completely mouseless, so the `file.choose()` command for reading data using `read.table()` or `read.csv()` is no longer available. Instead, type in the name of the file you want in inverted commas. Fortunately, R will look for the file in the working directory you were in when you started the console, so it may actually be even more straightforward. However, if you're not in the same directory, you'll need to type the path the same way as you would when you're copying, moving, or navigating files & folders.

```
mydata <- read.table("data.txt", header=T)
otherdata <- read.csv("work-monthly/RData/data.csv")
```

Creating pdf file of a plot

Unless you're using X11 forwarding, you won' t be able to see any plots you've made in R. Even if you are, once you've made the plot you want for your paper, you'll still need to save it. R can do this by opening its own workspace inside a pdf file, called a 'device'. Other devices include the Quartz & X11 windows you're already familiar with looking at your plots on.

First, call the `pdf()` function to tell it what the file should be called, then make your plot as you like it. To stop working in that file call the `dev.off()` function. You can then download it from the server to view it.

```
pdf("plot.pdf")
plot(object)
dev.off()
```

More information on R's ability to use devices (it can do more than just PDFs, and also have several open at once) can be found in the help files, `?pdf` etc.

10. Galaxy

There is a Galaxy server set up on Apocrita.

Login

https://galaxy.hpc.qmul.ac.uk/ - This is the link to the web interface.

To log in, use your entire QMUL email address i.e. j.doe@qmul.ac.uk as username and your **Apocrita** password.

SFTP

You can use SFTP to transfer large files to the Galaxy server. Use a GUI program to connect to the server using SFTP and transfer your files. The hostname is the same as the website, galaxy.hpc.qmul.ac.uk. Use the same login information as for the web interface.

Specifics

The actual server runs on frontend2 but it uses DRMAA to submit jobs to the cluster queue. It will run jobs as the "galaxy" user, that means that the user starting the job is not the one that executes it on the cluster, but the username is listed in the job name in the queue.

For an unknown reason the command line version of sftp may or may not work for transferring files. There seems to be some issue with permissions of files sometimes. Use Cyberduck if you can. Contact Adrian Lärkeryd or ITS-Research if you're having issues with the file transfer.

This server is still very much experimental and things change. If something is not working or you need a new tool installed do not hesitate to contact Adrian.

Repeatexplorer

Is set to use almost an entire fat node on Apocrita. Unfortunately, it doesn't allow for dynamically changing the resource requirement, thus even a small job will request a lot of resources from the cluster and be stuck in the SGE queue for longer than necessary. The main requirement of this tool seems to be memory and the biggest nodes on the cluster have 512GB of RAM. It is possible that some very large jobs could exceed that limit and be killed.

11. Singularity Pilot

ITS Research have installed Singularity for us to test in SBCS. Singularity is a containerisation platform which enables full control of the environment in which a program runs, without having to rely on libraries and software installed on the Apocrita filesystem.

IF YOU WANT TO TRY SINGULARITY DURING THE PILOT PHASE, PLEASE CONTACT ADRIAN

Containers

There are many flavours of containers and each has benefits and downsides. Docker is the most ubiquitous and the platform with the most features. However, there are some serious security flaws which prevents us from using Docker on the HPC. Singularity does however support a very simple import of Docker containers from the repository Docker Hub.

Quick-start import a Docker container from Docker Hub

If you have ever found a docker container you might want to use on Apocrita, the following will explain how to make it run on the cluster through Singularity. You do not need a Docker Hub user account but you do need to know the Docker Hub user and container names.

The only thing you need is access to a machine with Singularity installed where you have administrator permissions. This can be a Virtual Machine (VM) with Linux and Singularity installed in it or your local Linux machine if you have one. Here are instructions on how to get singularity running on a Mac and on Linux. If you are unsure about what this means or how to create a VM like that, please get in contact.

1. Create a definition file

Create a new text file and enter the following. Exchange the name of the Docker container from Docker Hub in the second line after `From:` to your preferred container. Leave everything else as it is. The Docker Hub user in this example is bioconductor and the container name is release_sequencing. The identifier for this container is thus `bioconductor/release_sequencing`.

`my_definition_file.def`:

```
Bootstrap: docker
From: bioconductor/release_sequencing
IncludeCmd: yes

%post
        mkdir /data
```

2. Build the image

To create the image and import the docker container run the following commands. Depending on the size of the Docker container, you may have to increase the size of your image. This is the step which requires administrative permissions on a Linux machine with Singularity installed.

```
sudo singularity create --size 1024 my_image_name.img
# Create empty image of size 1024MB

sudo singularity bootstrap my_image_name.img my_definition_file.def
# Run bootstrap on that image
```

3. Copy the image to Apocrita and run!

The image is ready to be transferred to Apocrita for execution on the cluster.

```
scp image_name.img btw000@login.hpc.qmul.ac.uk:~/
# Using scp to copy the image to your home directory on Apocrita
```

Once the image is on the Apocrita file system, create your submission script and add a singularity call to it to run your newly created container.

You can test your container first in a qlogin session, here is how it works:

```
qlogin
module load singularity
singularity exec my_image_name.img command_name argument1 argument2

#singularity exec my_image_name.img    <- This is a Singularity call
#command_name argument1 argument2   <- This is your call to the software
```

That's it! You have executed your first Singularity image.

Advanced Singularity

Creating a custom definition file

Unlike Docker, Singularity does not have a daemon running as root to control the containers running on the system. Everything is running as your user. However, in the steps that build the image file you need sudo access. For this reason, those steps cannot be completed on the Apocrita file system and must be carried out on another machine. This could for example be your desktop computer if you have a linux machine, a Virtual Machine you have set up yourself, or the vagrant VM that ITSR provides. There are official instructions on how to run singularity on a Mac and on Linux. After you are done creating the Singularity image, you can copy it to Apocrita and run it.

There are three fundamental steps to creating and running a container with Singularity.

1. Writing a bootstrap definition file
2. Creating the empty image and running the bootstrap (ie installing the things described in the definition file inside the image file)
3. Running the container

Definition file

This is a description file which tells singularity what to put inside the image. There are a few ways of doing it, but the most fundamental way is to have it download a version of ubuntu or centos, then install your tools/libraries through yum/apt-get/git etc. You can also import Docker images in the bootstrap step.

Here is an example of a fairly simple .def file (This particular one installs bowtie2-2.3.0 from sourceforge):

The first three lines specify the most common operating system, Ubuntu Xenial, and point to where it can be downloaded (There are other flavours of Linux you can build the image upon, if required see the Singularity documentation). Everything in the %post block runs once when the image is bootstrapped. A %test section is included as well to make sure everything worked out. There are other sections you can include here too, see the Singularity documentation.

```
Bootstrap: debootstrap
OSversion: xenial
MirrorURL: http://archive.ubuntu.com/ubuntu/

# Requires 1024MB image size

%post
    # Create the /data directory
    mkdir /data

    apt-get update

    # This is needed to be able to add the universe repo
    # which is needed for apt to find libtbb-dev

    apt-get install -y software-properties-common
    add-apt-repository universe

    # Running normal apt-get update and isntalling the tools needed
    apt-get update
    apt-get install -y vim wget unzip build-essential libtbb-dev

    # Downloading from sourceforge and installing
    wget https://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.3.0/bowtie2-2.3.0-source.zip

    unzip bowtie2-2.3.0-source.zip
    cd bowtie2-2.3.0/
    make
    make install

%test
    bowtie2 -h
```

Bootstrap

To create an image which contains the things you detailed in the definition file, you need to first create an empty image and then run the bootstrap. These two steps are the only ones that require sudo access. The default image size is 768MB, which is enough for many lightweight toolchains, but you can decide the image size yourself. Unfortunately it cannot be automatically grown by the bootstrap command, so if it runs out of space it will just crash. If this happens, delete the image and create a new one with larger size and try again. Sometimes its hard to tell what size you will need, I have created images varying in size from 768MB to 10GB.

```
sudo singularity create --size 1024 image_name.img
# Create empty image of size 1024MB

sudo singularity bootstrap image_name.img definition_file.def
# Run bootstrap on that image
```

Execution

This is the step in which you will run the container and the tools you have packaged within. This step does not require sudo access, but it does take settings from a global configuration file which ITSR has control over. The image is immutable once created so you will not be able to install any more tools or anything like that, you also cannot change/store data in the image once its been created. All of that will have to happen in the bootstrap step, so if you want to change the version of the tool installed, you need to create a new image and bootstrap it with a different definition file.

```
singularity exec image_name.img command [arguments ...]

# For example, to print bowtie2 help
singularity exec ubuntu-bowtie2-2.3.0.img bowtie2 -h
```

You can also shell into the container, either by using the singularity shell command or by exec invoking bash/sh:

```
singularity shell image_name.img

singularity exec image_name.img bash
```

Singularity on Apocrita

Singularity is installed on Apocrita and ITSR have allowed access to those willing to try it out. However, because singularity needs sudo access to create the empty image and for the bootstrap step, those two steps need to be completed outside Apocrita. Execution of a container does not require sudo access and thus it is possible to create the image on your own machine, copy it over to Apocrita and then run your packaged tool through singularity as your Apocrita user.

/data mount

In order to have access to your data on Apocrita, you need to create the /data directory in your image during the bootstrap step. After doing that, Singularity will be able to mount all your data directories automatically when you run the container. If you do not have the /data directory made in your bootstrap step the container will not have any access to your data files on Apocrita.

You do this in the %post section of your bootstrap definition file. It does not matter where, but I usually put it in the beginning to remember.

MPI

Singularity does support MPI over the cluster scheduler out of the box. In order to get it running there are a couple of things you need to do though.

Building the image

In order to take advantage of MPI on the cluster a version of MPI must be installed in the singularity image, as well as a way for the MPI jobs to communicate.

- The MPI version must be the same as the one you load on Apocrita
  - Loading the module openmpi/1.6.5 on Apocrita means you need to install Open MPI 1.6.5 in the bootstrap step of your image
- Installing openssh-server (using either yum or apt-get depending on your flavour) in the bootstrap step should enough to handle communications

Executing the image on the queue

Requesting an MPI environment on the cluster can be done in several ways. The new hardware provision will provide a large number of compute cores with high performance interconnect, which is perfect for MPI. It is still not clear whether these machines will be on a separate queue or if everything will be handled by the scheduler internally.

This is an example of a submission script which utilises MPI on the old provision. The number of parallel slots must match the -np argument supplied to mpirun!

```
#!/bin/sh
```

```
#$ -cwd              # Set the working directory for the job to the current directory
#$ -V                #
#$ -pe parallel 4    # Request 4 parallel slots
#$ -l h_rt=1:0:0     # Request 1 hour runtime
#$ -l h_vmem=1G      # Request 1GB RAM per core

module load use.dev
module load singularity
module load openmpi/1.6.5/gcc/4.7.2 # OpenMPI 1.6.5 is installed in the image

mpirun -np 4 singularity exec ./ubuntu-mrbayes-3.2.6.img mb ./examples/hymfossil.nex

# The image is called ubuntu-mrbayes-3.2.6.img
# and the command running within the container is mb ./examples/hymfossil.nex
```

Issues

Singularity is running with surprising smoothness, but not everything worked right away.

MPI - SOLVED

Had to install correct version of OpenMPI and install openssh-server to get it working. More details in the MPI section above.

Image size - WORKAROUND

Singularity does not automatically create an image of the correct size for your container, you need to specify the image size in the create step and if its too small the bootstrap step will fail. On the other hand you do not want to specify a huge image size because you will need to copy the image around afterwards!

There is a command to grow the image size - unfortunately I have found that once your bootstrap step has failed once, it will not work to run the bootstrap again on the same image. This is a minimal annoyance as you can just delete the image and create a new one with larger size, which is what I have been doing when I run into space issues in the bootstrap step.

I have tried to get into the habit of adding image size specifications in the .def file so that a future user will know what image size is needed to successfully complete the bootstrap step.

/data mount - SOLVED

Singularity will automatically mount your home directory, but it did not automatically mount the rest of /data, which is where almost all users have their data spread out. ITSR changed the config file for Singularity to automatically mount all of /data instead of just /data/home/username, that way all data that users have access to should be available in the container. Very simple and quick fix.

debootstrap not installed - SOLVED

This was a minor issue, but the host you are creating your images on has to have the tool debootstrap installed if you are going to base your images on ubuntu. This has to be installed with yum on a centos machine and may have to be installed even on ubuntu machines. `sudo yum install debootstrap` should do the trick. You might have to install yum on ubuntu to create centos images? (Not tested)

.def file too large - WORKAROUND

Found that one of my def files was too large, singularity complains in the bootstrap step if there is too many characters in the %post section. I recompiled Singularity on the machine where I build the images after editing some of the C code. It did work but I had to run the binary from the src directory, running `sudo make install` did not change anything...! More info on this github issue.

Software tries to write in installation directory - UNSOLVED

When installing Trinity into a singularity container, Trinity comes with several of its dependencies in the bundle. One of them is jellyfish. jellyfish, when executed, tries to write a file in its installation directory - which is cannot do due to the read only file system. I do not know if this can be resolved at all.

12. Handy Shortcuts

- Get your quota information `qmquota -s`
- Make a soft link to a file `ln -s /path/to/source/file.txt ./destination_file.txt`
- Find username or information on a user with `finger <search-term>`
- Peek into the head of a gzipped file with `zcat zippedupfile.gz | head`
- You can use `bc` to do precision calculations like so `bc -l <<< "sqrt(2)-5/7+18"`

Apocrita job stats

ITS-R has a webpage for statistics where you can see the load on the cluster but more importantly you can see your completed jobs.

stats.hpc.qmul.ac.uk

On the left hand menu there is a View your job detail in which you can see all the jobs ran on the cluster recently.

Advanced

This section contains more advanced information as well as tips and tricks for users who know their way around. Make sure you know what a command found here does before running it. If you are curious about something and want to know if it can be applied to your situation or research, contact us.

1. Apocrita Specifications

Apocrita cluster nodes

Thin nodes

- 150 Nodes
- Dual 6-core Intel Westmere (E5645) - 2.4GHz
- Memory 24GB

Hyperthreading is currently disabled on the thin nodes, giving us 12 cores and 24GB RAM, 2GB RAM per core.

Fat Nodes

- 11 Nodes
- Four 12-core AMD Bulldozer (6234) - 2.4GHz
- Memory 512GB

Fat nodes total 48 cores and 512GB RAM, ~10.67GB RAM per core.

Frontends

- frontend1.apocrita.hpc.qmul.ac.uk - This is a legacy server, you should not connect to frontend1 anymore.
- frontend8.apocrita.hpc.qmul.ac.uk - This is the one you should be using to connect to Apocrita. `login.hpc.qmul.ac.uk` currently points to frontend8.
- frontend2.apocrita.hpc.qmul.ac.uk - Galaxy runs on frontend2.
- There are other servers with the naming convention frontend# which in fact are **utility servers**, listed below.

Other information on Apocrita

- OS - Scientific Linux 6.2
- Interconnect - Gigabit Ethernet
- Queueing system - Sun Grid Engine 8.0.0e
- Compilers - Intel, Solaris Studio, Open64, Portland
- Parallel libraries - OpenMPI

Utility servers (SSH-able nodes)

sm11

- Purchased by Nichols, Leitch, Rossiter, Buggs, Wurm, Chelala, Clayton and Le Comber

A "set free" fat node that SBCS users can ssh directly into. The GPFS file system is mounted and it should work like any other part of Apocrita, just that the Sun Grid Engine doesn't schedule jobs here so that it is free for users to handle themselves. Check whether or not the machine is free (top, htop) before you start something, and maybe use a nice value.

To log in, just type `ssh sm11` when logged into a frontend. You can also make an ssh shortcut in your config file, explained below.

- Four 12-core AMD Opteron(TM) Processor 6234 - 2.4GHz
- Mem 512GB
- 16TB local scratch disk

Prometheus

- Purchased with NERC money by Nichols & Wurm.

This machine is not connected to Apocrita. It runs Ubuntu 14.04 and is administrated by SBCS users. A user has to be created for anyone who wants to use this hardware, see below for contact details.

- 2 10-core Intel Xeon CPU E5-2680 v2s clocked at Min:1199.953Mhz Max:2538.265Mhz with hyperthreading ON
- Kernel 3.18.9-031809-generic x86_64
- Mem 512GB
- 13TB local scratch disk
- Runs Ubuntu
- Docker installed

Contact: a.larkeryd@qmul.ac.uk, y.wurm@qmul.ac.uk, r.a.nichols@qmul.ac.uk

frontend5 and frontend6

- Purchased with NERC money by Nichols & Wurm.

These two identical machines are similar to sm11 but are running newer hardware. For this reason, these two machines are slightly more powerful than sm11. The only difference between the machines is that frontend5 has hyperthreading (HT) turned on and frontend6 has it turned off. If you are aware of your tools running better on HT on/off, you can use the appropriate machine. Please feel free to use the /tmp catalogue on these machines for your temporary analysis files, but remember to move off it as soon as you are done as your files may be deleted during reboots etc.

- Four 10-core Intel Xeon E5-4640 v2 - 2.20GHz. This means 40 threads with HT off and 80 with HT on.
- Mem 630GB
- 13TB local scratch disk located on /tmp - do NOT store things here, it is ephemeral will be deleted.
- cachecade using a 100GB SSD. This is a caching of the local spinning disks onto the SSD. In principle it should increase performance on /tmp.

GPU

There are no GPUs on Apocrita, if you need this for your analysis you should talk to your PI about acquisition. ITSR have been looking into the interest of such a purchase.

2. SSH Keys

Keys

You can set up a pair of SSH keys for a more secure as well as password-less login to Apocrita. This is done by having a private key on your local machine, and a matching public key on the remote server, when you try to log in these two match up and let you in without having to type the user password. The private key should never be shared with anyone as it will allow that person access to your login. This is why you should always **protect your private key** with a passphrase.

Key Generation

1. Open a terminal window (on Windows, use MobaXterm)
2. Enter `ssh-keygen` and hit enter
3. You will see `Enter file in which to save the key (/home/username/.ssh/id_rsa):` on the screen. Just hit enter here which will save the keys in their default location.
4. The program will now ask you for a passphrase. Please enter one (it is possible to create a key without it but don't, it's to protect from someone getting hold of your private key.)
5. There is a message telling you that the key pair has been created, the public key is now located in /home/username/.ssh/id_rsa.pub and the private key is /home/username/.ssh/id_rsa.
6. You are now ready to copy the **public key** to Apocrita

Key for Hive

To connect to the Hive archive from Apocrita you need to create a key pair on Apocrita. You need to do this while logged in to Apocrita:

1. Log in to Apocrita `ssh btw000@login.hpc.qmul.ac.uk`
2. Run the command `ssh-keygen`
3. You will see `Enter file in which to save the key (/data/home/btw000/.ssh/id_rsa):` on the screen. Just hit enter here which will save the keys in their default location.
4. The program will now ask you for a passphrase. Please enter one (it is possible to create a key without it but don't, it's to protect from someone getting hold of your private key.)
5. There is a message telling you that the key pair has been created, the public key is now located in /data/home/btw000/.ssh/id_rsa.pub and the private key is /data/home/btw000/.ssh/id_rsa.
6. Open the **public key** with for example `less ~/.ssh/id_rsa.pub`. Don't touch the **private key**!
7. Send the **public key**, everything that is in the file you just opened, to Adrian.

Public Key Copy

The process is different depending on which operating system you are using.

Windows and Linux

Here it's very simple, open a terminal (or MobaXterm window) and type in

```
ssh-copy-id btw000@login.hpc.qmul.ac.uk
```

using your own username. That's it. Now try your connection

```
ssh -X btw000@login.hpc.qmul.ac.uk
```

Mac

> *If you have Homebrew installed on your Mac you can use it to install `ssh-copy-id` and go from there.*

One of the few times having a mac will make you suffer extra work. If you do not have Homebrew, you should install it, but if you do not want to do that, you will have to manually copy your **public key** to a file located in your home directory on Apocrita.

1. Open a terminal and go to your home directory with `cd`
2. Use `scp` to copy the public key to Apocrita with this command `scp .ssh/id_rsa.pub btw000@login.hpc.qmul.ac.uk:~/`
3. Login to Apocrita `ssh btw000@login.hpc.qmul.ac.uk`

4. `cat ~/id_rsa.pub >> ~/.ssh/authorized_keys`
5. You can now log out and try the connection again to see if your keys work!

Shortcuts

Another convenience tip is to add shorthand names for your ssh logins. The address to the Apocrita head node is quite long and arguably tedious to write. So, this being computer science, of course there is a setup that will allow you to simply type something like `ssh apocrita` on the command line to connect.

1. `cd ~/.ssh`
2. Open or create the config file `nano config`
3. Add the following, you may call the Host whatever you like, here I'm using "apocrita":

```
Host apocrita
 Hostname login.hpc.qmul.ac.uk
 IdentityFile ~/.ssh/id_rsa
 User btw000
 ServerAliveInterval 300
 ForwardX11 yes
```

4. Save and quit the editor
5. Change permissions of the file `chmod 600 ~/.ssh/config`
6. Try `ssh apocrita`

Another example, using apocrita (our newly created shortcut to frontend1) as a proxy to connect to sm11. This way all you need to connect to sm11 is `ssh sm11`.

```
Host sm11
    Hostname sm11
    User btw000
    ProxyCommand ssh apocrita nc %h %p
    ForwardX11 yes
```

You probably want X11 forwarding (-X option in ssh) and there are other options.

3. `zsh` defaults

These are instructions to set up your shell with some useful `zsh` settings like better auto completion and history etc. It also contains a small set of useful aliases.

All you need to do

Save your old `.bash_profile` and `.zshrc` files

These files control settings which should be set up as you log in. They are located in your home directory and control bash and zsh repsectively (You may not have a `.zshrc` file if you haven't tried that shell before). You will need to remove them in order for this to work, but it is highly recommended that you save them in case you want to go back. If you are controlling bash through a `.bashrc` file, it will not be changed during this process. You might still want to make a backup of that one and remove it in order to have a clean setup to begin with.

```
cd
mv .bash_profile .bash_profile.BAK        # this file runs every time you log in with bash
mv .bashrc .bashrc.BAK                    # this file is a complement to .bash_profile
mv .zshrc .zshrc.BAK                      # this file runs every time you log in with zsh
```

Run the small installation

```
source /data/SBCS-Informatics/zsh_extensions
```

You should find yourself in a helpful shell. Have fun.

What happened was that a new `.bash_profile` was created which launches `zsh`. This in turn reads a new `.zshrc` file. You may have had some settings set in your old `.bash_profile`, if you want to keep those settings just copy it all to the end of the newly created `.zshrc` file.

Features

Prompt directory information

`zsh` will show you an abbreviated version of the full path to where you are, giving you better sense of the file system.

```
/data/scratch/btw000/folder1         # pwd - this is the directory
[btw000@frontend1 folder1]$          # prompt with default bash settings
btw000@frontend1 /d/s/b/folder1>     # here is how it looks with the new settings
```

Better tab completion

Case insensitive tab completion and you can tab between the different options.

```
btw977@frontend1 ~> cd /data/sbcs    # pressing TAB twice
sbcs/                SBCS-ClareLab/       SBCS-EizaguirreLab/  SBCS-GreyLab/        SBCS-LeitchLab/      SBCS-OsmanLab/
SBCS-BessantLab/     SBCS-ClaytonLab/     SBCS-ElphickLab/     SBCS-HirstLab/       SBCS-MarinakisLab/   SBCS-RossiterL
SBCS-BuggsLab/       SBCS-CrespoOtero/    SBCS-EvansLab/       SBCS-HurdLab/        SBCS-McElligottLab/  SBCS-RubanLab/
SBCS-ChassLab/       SBCS-DiTommasoLab/   SBCS-ForniliLab/     SBCS-Irys/           SBCS-MSc-BioInf/     SBCS-Stollewer
SBCS-ChittkaLab/     SBCS-DuffyLab/       SBCS-GoldupLab/      SBCS-LeComberLab/    SBCS-NicholsLab/     SBCS-WurmLab/
```

`zsh` 5.0.7

The default version of zsh that every SBCS user has (unless explicitly changed) is version 4.3.10 which doesn't contain all the functionality used here. Because of that a check is made when logging in and if the version of `zsh` isn't 5.0.7, it is loaded.

Oh My Zsh

Oh My Zsh is a large package of settings, plugins and themes for zsh which is used as foundation for these defaults. It has a ton of options but it is kept fairly simple here, including the theme. It does allow for much more customisation which you can do for yourself.

Per directory history

This plugin is enabled to give you a command line history that is specific to the directory you are in. That way when you go back to the folder where you carried out your analysis all those months ago you can just press the up-arrow on your keyboard and it will go through what last happened in this particular directory (instead of the failed installation of that python package you were wrestling with yesterday.) Please keep in mind that this is not retroactive!

Customising further

Perhaps you want some more fancy features than what is provided here? There are a multitude of themes that change appearance of your shell, or maybe it's syntax highlighting you crave. The setup followed above inserted a sourcing of our general `.zshrc` file in your local version. That file is located in your home directory.

If you are adding things like `module load` commands, aliases, changing your $PATH variable or something similar, the easiest way is to just add it to the end of your new `.zshrc` file which is in your home.

You can also copy all the contents of the general SBCS-zshrc (/data/SBCS-Informatics/SBCS-zshrc) to your own `.zshrc`. That way you can change anything you like! Add plugins, try the random theme, or even build your own functions. You can of course write your own .zshrc and use other packages.

There are other shells available as well. `fish` for example, which claims to be "Finally, a command line shell for the 90s". However, `fish` does have slightly different syntax than `bash` and `zsh`, the two of which are very similar. Feel free to keep looking, there are other more obscure things out there.

Going back

To return to your previous settings, remove the two new files `.bash_profile` and `.zshrc` and rename your backed up files to their original names.

```
cd
rm .bash_profile                        # remove the new bash config
rm .zshrc                               # remove the new zsh config
mv .bash_profile.BAK .bash_profile      # reinstate your old bash profile
mv .bashrc.BAK .bashrc                  # if you were using bashrc
mv .zshrc.BAK .zshrc                    # if you were using zshrc
```

4. Oneliners and tricks

- Long list of bioinformatic and non-bio oneliners
- Get all sbcs users: `ldapsearch -x cn=sbcs | grep memberUid | sort`.
- All compute nodes have local scratch disks at $TMPDIR which equates to /tmp/jobid.queueid - It is purged at the end of the job so copy everything back when your data processing has finished.

Put an already running process under nohup

If you have started a process in an interactive session and want to put it in a state where it will keep running when you close the terminal window or lose connection, this is your solution.

1. Press CTRL+Z to suspend the process
2. `bg %1` to start the process in the background
3. `disown %1` to make it run disconnected from the terminal

Contact

Please send an email if you need anything. If you have a question, or more information, you would like to add to the documentation, feel free to use the EDIT link up top and create a GitHub issue or pull request.

Adrian Lärkeryd

- a.larkeryd@qmul.ac.uk