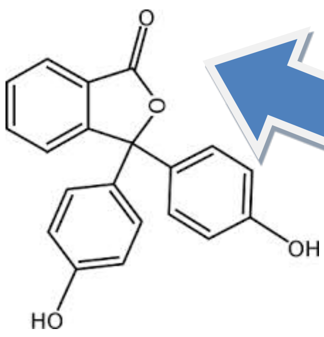


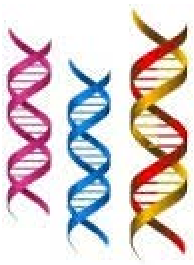
# Submitting and monitoring batch jobs using Torque & Maui

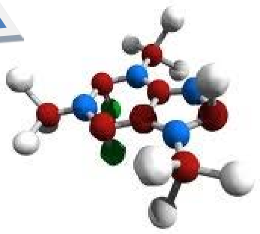
**Mark Meenan**

**University of Glasgow, IT Services**

**For help email IT-HPC-Support@glasgow.ac.uk**

**Ver 3.3.2**





# Getting Started

## Web Pages

The first thing you should do is look at the IT High Performance Compute Clusters web page for general information about the cluster. You will find a link to a basic user guide and registration form.

## Preparing your account for use

Once you get an account you will be asked to set a password. Then you should do a few tasks to set up your account for use.

You need to set up ssh as explained below, the reason for this is to allow inter-process communication on the cluster and it also is needed by TORQUE to copy the error and output files from the compute node to the headnode at the end of the job.

**Please run this configuration once you are logged into the system**
**You need to create an ssh key to allow the cluster to pass files back and forth**
Run the command to generate your key:

- **ssh-keygen -t rsa**

When prompted to enter a password just press return, to work with PBS/Torque it needs to be a passwordless ssh - this will create a number of files in ~/.ssh/.
Then you need to run

- **cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys**

There is sometimes problems created by the permissions on this authorized_keys file. So check the permissions on ~/.ssh/authorized_keys by running the command:

- **ls -l ~/.ssh/**

It should show the line (where -rw------- is the important part):

- -rw------- (some other fields) authorized_keys

If it does not run the following command:

- **chmod 600 ~/.ssh/authorized_keys**

I think it is also worth setting up mail forwarding (explained below). This means that any emails from the system will be forwarded to an account(s) you specify.

**Mail Forwarding Configuration**

It is recommended that you create a .forward file to forward mail to your email address - by default Torque sends mail to you when a job starts & finishes. eg.
touch ~/.forward
echo user@udcf.gla.ac.uk > ~/.forward
cat ~/.forward to check
Permissions are again important check that the permissions are -rw-r--r-- if not run the command
chmod 644 ~/.forward

There are also links to basic instructions on how to use Torque/PBS
and running MPI jobs.

# Resources available to a job

## Headline Specification

- *1020 cores*
- *7 GeForce GTX 1080 GPUs*
- *Up to 16GB RAM per core*

## The hardware is a mix of:

- *7 Dell PowerEdge 6145 with Quad 16 core AMD Opteron 6376 and 512GB RAM per node*
- *2 Avanti SYS-2028GR-TE each with 4 GeForce GTX 1080 GPUs*
- *2 Dell R440 with Dual 24 Thread Xeon 4116 and 384GB RAM per node*
- *1 Dell R930 with Quad 24 Thread Xeon 4830 and 1.5TB RAM*
- *1 Dell R840 with Quad 40 Thread Xeon 6138 and 256GB RAM*
- *1 Dell R750 with Dual 16 Core Xeon Gold 6346 and 768GB RAM*

The Avanti's, Dell R440's, Dell R930 and R840 have been purchased by research groups/projects and so have additional restrictions on access.

There are a number of resources available to your job, as part of the job submission process a temporary directory is created in /tmp using the $PBS_JOBID variable. You can refer to it in your scripts by using the path /tmp/$PBS_JOBID – this gives you some local disk space to use. This area is deleted when your job finishes.

There is your home area $HOME or ~ which is exported to all the compute nodes and there is /export/scratch/username again exported to all the compute nodes Some research groups have additional space under /export/projects/.

In addition there are the normal tools and libraries available on linux.

# Overview of the batch process

You submit a job to the cluster using qsub. The Maui and torque programs will then decide if your job can be run now or if it is to be queued. When the job is run it will be allocated one or more processors on one or more nodes, depending on the resources you have requested.

As your job starts a temporary folder is created on the execution nodes assigned to you at /tmp/$PBS_JOBID (where $PBS_JOBID is an environment variable you can reference in your code) . Your job will then run to completion, or until one of the resource limits is reached (typically cput or walltime).

As your job exits the temporary folder is deleted along with all its contents (ie it is up to your job to save its contents back to somewhere accessible). The clean up process will also kill any remaining processes belonging to you on the execution nodes. Two files are then copied back to your home area (in actual fact the directory from which you launched qsub). These files are in the format *jobname.*o*jobid* and *jobname.*e*jobid*. (eg STDIN.o17469 and STDIN.e17469) o is for output and e is for error

The output file will have something like

```
Prologue Args:
Job ID: 17469.headnode01.cent.gla.ac.uk
User ID: mjm4y
Group ID: mjm4y
MAchine: comp01

Epilogue Args:
Job ID: 17469.headnode01.cent.gla.ac.uk
User ID: mjm4y
Group ID: mjm4y
Job Name: script_name
Session ID: 18380
Resource List: neednodes=6:ppn=2,nodes=6:ppn=2,walltime=60:00:00
Resources Used: cput=00:00:00,mem=3676kb,vmem=114680kb,walltime=00:19:32
Queue Name: parallel
Account String:
Process's killed on comp01
tmp directory removed on comp01
Process's killed on comp02
tmp directory removed on comp02
Process's killed on comp05
tmp directory removed on comp05
Process's killed on comp06
tmp directory removed on comp06
Process's killed on comp07
tmp directory removed on comp07
Process's killed on comp08
tmp directory removed on comp08
```

The error file should contain any errors that your code produced. It is expected your job will write its output to somewhere you specify. Your home area $HOME and a scratch directory /export/scratch/username are provided through NFS to all the execution nodes. You may also have further space allocated in the /export/project directory to your research group.

# Useful Commands in PBS/TORQUE & Maui

## qsub

**/usr/local/bin/qsub**

This is the main job wrapper in PBS/Torque and you will use this to submit all your jobs, it is worth spending some time looking at the man pages from Adaptive Computing. We will discuss here the most used options -I, -l, -m, -M, -N.

When you use qsub you can pass options in 2 ways – on the command line and within your job script, where appropriate we will use both methods in the examples below.

### Queue structure

Before going any further it is important to understand the queue structure and default behaviour. This will then allow you to predict the resource allocation given to your job.

There are currently 10 queues – when you first submit your job it goes into ***feed*** which is a routing queue. The feed queue then allocates the job to one of the other queue's based on the resources asked for. The other 5 main queues are called ***short, long, verylong, parallel, and veryparrallel.*** In addition there are ***3 bioinf-stud*** queues and a ***gpu*** queue. Allocation to a main queue is on the basis of expected cputime (cput) or wall clock time (walltime) and the number of processors you request. (The ***bioinf-stud*** and ***gpu*** queues have additional restrictions based on group membership, they are included here for completeness but you can ignore it for now).

The queue your job is allocated to is determined by checking the resources you request against the queues in order (***short, long, verylong, parallel, veryparrallel*** and ***bioinf-stud).*** You job will be run on the first queue that matches your request.

NOTE: the bioinf-stud2 & bioinf-stud3 queues are currently disabled – they are enabled over the summer to facilitate student projects on the Bioinformatics Masters Course.

| **Queue Name** | **Max CPU Time** | **Max Walltime** | **Min-Max Number of Processors** | **Max jobs allowed in queue per user** |
|---|---|---|---|---|
| **Short** | 10 hours | 2 hours | 1-16 procs | 175 |
| **Long** | 500 hours | 50 hours | 1-16 procs | 50 |
| **Verylong** | - | - | 1-16 procs | 35 |
| **Parallel** | - | - | 17-32 procs | 10 |
| **Veryparrallel** | - | - | 17-128 procs | 2 |
| **Bioinf-stud** | - | - | 1-4 procs | 5 |
| **Bioinf-stud2** | - | - | 5-16 procs | 3 |
| **Bioinf-stud3** | - | - | 17-32 procs | 1 |
| **GPU** | - | 192 hours | 1-4 GPUs | 25 |
| **biobank** | - | - | 1-open procs | 25 |
| **bioinf-verylong** | - | - | 1-40 procs | 25 |
| **bioinf-parallel** | - | - | 17-40 procs | - |

If you reach the limit for the number of jobs allowed in the queue your job will remain in ***feed*** until space becomes available.

The queues also have some defaults – which are applied if that resource is not specified when submitting a job.

| **Queue Name** | **Default CPU Time** | **Default Walltime** | **Default processors** | **Max running jobs per user** |
|---|---|---|---|---|
| **Short** | 1 hour | 1 hour | 1 proc | 175 |
| **Long** | 24 hours | 24 hours | 1 proc | 16 |
| **Verylong** | 288 hours | 288 hours | 1 proc | 12 |
| **Parallel** | - | 288 hours | 17 procs | 4 |
| **Veryparrallel** | - | 288 hours | 33 procs | 1 |
| **Bioinf-stud** | 1 hour | 1 hour | 1 proc | 5 |
| **Bioinf-stud2** | 1 hour | 1 hour | 5 procs | 3 |
| **Bioinf-stud3** | 1 hour | 1 hour | 17 procs | 1 |
| **GPU** | - | 24 hours | 1 GPU | 2 |
| **biobank** | 288 hours | 288 hours | 1 proc | 25 |
| **bioinf-verylong** | 288 hours | 288 hours | 1 proc | 10 |
| **bioinf-parallel** | - | 288 hours | 17 proc | - |

Finally, due to job housekeeping scripts, **you are only allowed to have one running job on a physical node at any one time.**

Here are some examples to help explain what will happen when you submit a job:-

You submit a job with no request for resources. It will enter the feed queue where it will be routed to the short queue. The short queue will then apply its defaults of cput = 1 hour, walltime = 1 hour and processors =1. The job will then be allocated to a node and run until either the job finishes, OR it reaches 1 hour of cpu time OR it reaches 1 hour of real time.

You submit a job with a request for 2 hours walltime. It will enter the feed queue where it will be routed to the short queue. The short queue will then apply its defaults of cput = 1 hour and processors =1. The job will retain its requested 2 hours walltime. The job will then run until either the job finishes OR it reaches 1 hour of cpu time OR 2 hours of real time.

You submit a job with a request for 3 hours cput. It will enter the feed queue where it will be routed to the long queue. The long queue will then apply its defaults of walltime = 24 hours and processors =1. The job will retain its requested 3 hours cput. The job will then run until either the job finishes OR it reaches 3 hours of cpu time OR 24 hours of real time.

You submit a job with a request for 3 hours cput and 200 hours walltime. It will enter the feed queue where it will be routed to the verylong queue. The verylong queue will then apply its default of processors =1. The job will retain its requested 3 hours cput and 200 hours walltime. The job will then run until either the job finishes OR it reaches 3 hours of cpu time OR 200 hours of real time.

You submit a job with a request for 17 processors and 1 hour cput. It will enter the feed queue where it will be routed to the parallel queue (because you have requested 17 processors) the default walltime will be set to 288 hours so the job will run with an allocation of 17 processors. The job will run until it finishes OR reaches 1 hour cpu time or 288 hours of walltime.

## Qsub options

### -I

The first option qsub –I will open an interactive shell running on one of the compute nodes – this is most useful if you want to test something or want to compile some piece of software. When you use this you should also consider what resources you need and pass them with the –l option, otherwise your job will go into the short queue and have the associated defaults applied. This shell behaves exactly as any other shell and can be used as such – if you requested multiple nodes then it will run on the first node in the list (the one with the lowest number) and by default all your jobs will run on this node but you can access the other processors as long as your software is aware of mpi more on this in the mpi section later)

Some examples

*qsub –I*

This will open an interactive shell on 1 node with the default resource allocation of the short queue

*qsub –I –l cput=01:30:00,walltime=02:00:00,nodes=2:ppn=2*

This will open an interactive shell on a node and allocate it the resources of 1hr 30 minutes cputime, 2 hours walltime and 2 nodes each with 2 processors. Using qsub –I in a script makes no real sense

### -l

The –l option is to allow you to request specific resources from the cluster that override the defaults. A full list of the options you can request are detailed from Adaptive Computing. The most common resources requested are cput, walltime and nodes. We will describe them in turn.

**cput** is the amount of processor time you would like to request as opposed to **walltime** which is the actual clock time your job will take. **Nodes** is the number of processors you need, there are a number of ways of describing this, lets say that you need 8 processors – you can say nodes=8 this will return you 8 processors on between 1 and 8 physical compute nodes– now you probably want to run your job on one physical machine so you can say instead nodes=1:ppn=8 – this says you need 1 node with 8 processors free, this will give you access to 8 processors on one physical node. So some examples:-

*qsub –l cput=01:30:00, walltime=02:00:00 script.sh*

your job script.sh is submitted to the cluster and allocated to one node with a cputime of 1:30 hours and walltime of 2 hours

*qsub –l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2 script.sh*

as before but your job is allocated two processors on a particular node.

Now to put these options in a script you would do the following

Run *qsub script.sh*

The script script.sh would start with the qsub options

```
#!/bin/sh
#PBS -l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2
/path/to/programme_to_run
```

You can also set any environment variables you need within the script.

There are additional node labels we have set in our cluster to allow you to select particular nodes:- one set for RAM giving the number of GB of ram per processing core

twogpc eightgpc sixteengpc

and another for the OS

centos7 oracle9

these can be requested by adding them to the nodes section of the command line or script ie:-

*-l nodes=1:ppn=2:eightgpc*

Which will allocate 2 processors on one of the nodes with 8 GB of RAM per processor.

## -M & -m

The next two options –m and –M are to do with mailing job status reports –m asks for an email to be sent to the user that submits the job you give in one or more of three options, those options are b to be mailed when a job begins execution, e to be mailed when a job completes execution and a to be mailed if a job is aborted. The –M option is to allow you to override the default email address that the programme sends to. (Assume jobs have been submitted by mjm4y)

```
qsub –l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2 –m e script.sh
```

In this case you will be emailed to your user account on the cluster when the job finishes. The email will go to the user who submitted the job – ie mjm4y@headnode01.cent.gla.ac.uk

```
qsub –l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2 –m be script.sh
```

In this case you will be mailed when the job starts as well

```
qsub –l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2 –m abe script.sh
```

In this case you will also get a mail if the job is aborted.

```
qsub –l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2 –m abe  -M mjm4y@udcf.gla.ac.uk script.sh
```

In this final case, note that the email will be sent to mjm4y@udcf.gla.ac.uk rather than the default mjm4y@headnode01.cent.gla.ac.uk.

To put these options in a script we would amend the script above to the following

```
#!/bin/sh
#PBS -l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2
#PBS –m abe
#PBS –M mjm4y@udcf.gla.ac.uk
/path/to/programme_to_run
```

## -N

The final option I want to discuss briefly is –N this will give your job a name, which will then be used by PBS

```
qsub –l cput=01:30:00, walltime=02:00:00 –N my_job script.sh
```

You job will be submitted to the cluster and will be referred to by the name my_job in the qstat output and the output and error files.

This can be put in a script as before

```
#!/bin/sh
#PBS -l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2
#PBS –N my_job
/path/to/programme_to_run
```

Jobs will by default run in the directory from which it is launched. It is best therefore to give full paths to all executables and files within your job script rather than use relative paths.

There are a number of other options you may wish to look at (see the man page from Adaptive Computing) for example –o and –e  allows you to redirect output (by default the output and error files will be written to the directory you launch qsub from).

# Useful Commands in PBS/TORQUE & Maui

## qstat

**/usr/local/bin/qstat**

This gives the status of the work queue.

There are a number of possible options you can pass to this command which determines what the output is.

qstat  on its own gives the output below – Time Use is cput used to date. S is the status of the job where Q is queued, R is running, E is exiting and C is completed.

```
[root@headnode04 export]# qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
245017.headnode03          SUB10_G09.new    gfb2y           1776:03: R verylong
245070.headnode03          SUB10_G09.new    ms208c          1739:05: R verylong
245079.headnode03          SUB20_G09.new    ms208c          1932:05: R verylong
245089.headnode03          SUB2_G09.new     ms208c          1926:46: R verylong
245164.headnode03          SUB45_G09.new    gfb2y           1547:39: R verylong
245172.headnode03          SUB20_G09.new    gfb2y           1466:26: R verylong
245176.headnode03          SUB3_G09.new     gfb2y           1019:17: R verylong
245177.headnode03          SUB41_G09.new    gfb2y           975:43:3 R verylong
245204.headnode03          SUB8_G09.new     ms208c          1193:56: R verylong
245205.headnode03          SUB_G09.new      ms208c          1157:09: R verylong
245206.headnode03          SUB33_G09.new    ms208c          1081:30: R verylong
245351.headnode03          SUB38_G09.new    ms208c          293:49:2 R verylong
245352.headnode03          SUB5_G09.new     ms208c          293:57:1 R verylong
245358.headnode03          SUB7_G09.new     ms208c          225:48:3 R verylong
245370.headnode03          SUB1_G09.new     1107184o        1040:47: R verylong
245435.headnode03          SUB_G09.new      gfb2y           260:11:4 R verylong
245442.headnode03          SUB39_G09.new    ms208c          109:20:3 R verylong
245445.headnode03          SUB9_G09.new     ms208c          103:36:2 R verylong
245446.headnode03          SUB4_G09.new     gfb2y           218:19:1 R verylong
245463.headnode03          SUB28_G09.new    ms208c          41:51:14 R verylong
245464.headnode03          SUB32_G09.new    ms208c                 0 Q verylong
245465.headnode03          SUB14_G09.new    ms208c                 0 Q verylong
245466.headnode03          SUB23_G09.new    ms208c                 0 Q verylong
245467.headnode03          SUB30_G09.new    ms208c                 0 Q verylong
245486.headnode03          SUB1_G09.new     1002159f        953:47:5 R verylong
245531.headnode03          SUB25_G09.new    ms208c                 0 Q verylong
245589.headnode03          SUB21_G09.new    ms208c                 0 Q verylong
245740.headnode03          SUB23_G09.new    gfb2y           41:14:51 R verylong
245752.headnode03          SUB2_G09.new     gfb2y           38:30:03 R verylong
245755.headnode03          SUB32_G09.new    gfb2y           34:38:15 R verylong
245758.headnode03          SUB38_G09.new    gfb2y           33:52:04 R verylong
245759.headnode03          SUB25_G09.new    gfb2y           31:20:48 R verylong
245828.headnode03          SUB9_G09.new     gfb2y                  0 Q verylong
```

Passing the –a option to qstat gives more information. SessID is the process pid on the execution node, note this is not recorded for mpi jobs, NDS is the number of nodes requested and TSK is the number of processors Req'd Time & Elap Time refer to CPU time. (Note for jobs running on more than one node the Sess ID is given as 0)

```
[root@headnode04 export]# qstat -a

headnode03.cent.gla.ac.uk:
                                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
245017.headnode03.cent  gfb2y       verylong SUB10_G09.new      8598     1      8    --  3000:00:0 R 1776:03:1
245070.headnode03.cent  ms208c      verylong SUB10_G09.new     10008     1      8    --  3000:00:0 R 1739:05:2
245079.headnode03.cent  ms208c      verylong SUB20_G09.new      9468     1      8    --  3000:00:0 R 1932:05:3
245089.headnode03.cent  ms208c      verylong SUB2_G09.new      10190     1      8    --  3000:00:0 R 1926:46:4
245164.headnode03.cent  gfb2y       verylong SUB45_G09.new     18769     1      8    --  3000:00:0 R 1547:39:2
245172.headnode03.cent  gfb2y       verylong SUB20_G09.new     33090     1      8    --  3000:00:0 R 1466:26:4
245176.headnode03.cent  gfb2y       verylong SUB3_G09.new      57535     1      8    --  3000:00:0 R 1019:17:1
245177.headnode03.cent  gfb2y       verylong SUB41_G09.new      8962     1      8    --  3000:00:0 R 975:43:31
245204.headnode03.cent  ms208c      verylong SUB8_G09.new      51805     1      8    --  3000:00:0 R 1193:56:3
245205.headnode03.cent  ms208c      verylong SUB_G09.new       45770     1      8    --  3000:00:0 R 1157:09:5
245206.headnode03.cent  ms208c      verylong SUB33_G09.new     58638     1      8    --  3000:00:0 R 1081:30:1
245351.headnode03.cent  ms208c      verylong SUB38_G09.new     30180     1      8    --  3000:00:0 R 293:58:42
245352.headnode03.cent  ms208c      verylong SUB5_G09.new      27263     1      8    --  3000:00:0 R 293:57:19
245358.headnode03.cent  ms208c      verylong SUB7_G09.new      43759     1      8    --  3000:00:0 R 225:48:39
245370.headnode03.cent  1107184o    verylong SUB1_G09.new      59552     1      8    --  3000:00:0 R 1040:47:4
245435.headnode03.cent  gfb2y       verylong SUB_G09.new       27488     1      8    --  3000:00:0 R 260:11:46
245442.headnode03.cent  ms208c      verylong SUB39_G09.new      9191     1      8    --  3000:00:0 R 109:20:39
245445.headnode03.cent  ms208c      verylong SUB9_G09.new      55255     1      8    --  3000:00:0 R 103:42:25
245446.headnode03.cent  gfb2y       verylong SUB4_G09.new      51079     1      8    --  3000:00:0 R 218:19:15
245463.headnode03.cent  ms208c      verylong SUB28_G09.new     30234     1      8    --  3000:00:0 R  41:51:14
245464.headnode03.cent  ms208c      verylong SUB32_G09.new       --      1      8    --  3000:00:0 Q       --
245465.headnode03.cent  ms208c      verylong SUB14_G09.new       --      1      8    --  3000:00:0 Q       --
245466.headnode03.cent  ms208c      verylong SUB23_G09.new       --      1      8    --  3000:00:0 Q       --
245467.headnode03.cent  ms208c      verylong SUB30_G09.new       --      1      8    --  3000:00:0 Q       --
245486.headnode03.cent  1002159f    verylong SUB1_G09.new       2989     1      8    --  3000:00:0 R 953:47:54
245531.headnode03.cent  ms208c      verylong SUB25_G09.new       --      1      8    --  3000:00:0 Q       --
245589.headnode03.cent  ms208c      verylong SUB21_G09.new       --      1      8    --  3000:00:0 Q       --
245740.headnode03.cent  gfb2y       verylong SUB23_G09.new     57724     1      8    --  3000:00:0 R  41:14:51
245752.headnode03.cent  gfb2y       verylong SUB2_G09.new      30469     1      8    --  3000:00:0 R  38:40:50
245755.headnode03.cent  gfb2y       verylong SUB32_G09.new     23118     1      8    --  3000:00:0 R  34:38:15
245758.headnode03.cent  gfb2y       verylong SUB38_G09.new     12564     1      8    --  3000:00:0 R  34:00:35
245759.headnode03.cent  gfb2y       verylong SUB25_G09.new      3754     1      8    --  3000:00:0 R  31:28:09
245828.headnode03.cent  gfb2y       verylong SUB9_G09.new        --      1      8    --  3000:00:0 Q       --
```

You can also request only idle jobs with qstat –i

```
[root@headnode04 export]# qstat -i

headnode03.cent.gla.ac.uk:
                                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
245464.headnode03.cent  ms208c      verylong SUB32_G09.new       --      1      8    --  3000:00:0 Q       --
245465.headnode03.cent  ms208c      verylong SUB14_G09.new       --      1      8    --  3000:00:0 Q       --
245466.headnode03.cent  ms208c      verylong SUB23_G09.new       --      1      8    --  3000:00:0 Q       --
245467.headnode03.cent  ms208c      verylong SUB30_G09.new       --      1      8    --  3000:00:0 Q       --
245531.headnode03.cent  ms208c      verylong SUB25_G09.new       --      1      8    --  3000:00:0 Q       --
245589.headnode03.cent  ms208c      verylong SUB21_G09.new       --      1      8    --  3000:00:0 Q       --
245828.headnode03.cent  gfb2y       verylong SUB9_G09.new        --      1      8    --  3000:00:0 Q       --
245831.headnode03.cent  ms208c      verylong SUB24_G09.new       --      1      8    --  3000:00:0 Q       --
```

or running jobs with qstat –r

```
[root@headnode04 export]# qstat -r

headnode03.cent.gla.ac.uk:
                                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
245017.headnode03.cent  gfb2y       verylong SUB10_G09.new      8598     1      8    --  3000:00:0 R 1776:55:2
245070.headnode03.cent  ms208c      verylong SUB10_G09.new     10008     1      8    --  3000:00:0 R 1739:59:1
245079.headnode03.cent  ms208c      verylong SUB20_G09.new      9468     1      8    --  3000:00:0 R 1933:05:2
245089.headnode03.cent  ms208c      verylong SUB2_G09.new      10190     1      8    --  3000:00:0 R 1927:46:4
245164.headnode03.cent  gfb2y       verylong SUB45_G09.new     18769     1      8    --  3000:00:0 R 1548:39:1
245172.headnode03.cent  gfb2y       verylong SUB20_G09.new     33090     1      8    --  3000:00:0 R 1467:25:3
245176.headnode03.cent  gfb2y       verylong SUB3_G09.new      57535     1      8    --  3000:00:0 R 1020:17:1
245177.headnode03.cent  gfb2y       verylong SUB41_G09.new      8962     1      8    --  3000:00:0 R 976:43:28
245204.headnode03.cent  ms208c      verylong SUB8_G09.new      51805     1      8    --  3000:00:0 R 1194:56:2
245205.headnode03.cent  ms208c      verylong SUB_G09.new       45770     1      8    --  3000:00:0 R 1158:09:1
245206.headnode03.cent  ms208c      verylong SUB33_G09.new     58638     1      8    --  3000:00:0 R 1082:30:1
245351.headnode03.cent  ms208c      verylong SUB38_G09.new     30180     1      8    --  3000:00:0 R 294:54:54
245352.headnode03.cent  ms208c      verylong SUB5_G09.new      27263     1      8    --  3000:00:0 R 294:55:00
245358.headnode03.cent  ms208c      verylong SUB7_G09.new      43759     1      8    --  3000:00:0 R 226:48:03
245370.headnode03.cent  1107184o    verylong SUB1_G09.new      59552     1      8    --  3000:00:0 R 1041:42:5
245435.headnode03.cent  gfb2y       verylong SUB_G09.new       27488     1      8    --  3000:00:0 R 261:11:45
245442.headnode03.cent  ms208c      verylong SUB39_G09.new      9191     1      8    --  3000:00:0 R 110:20:37
245445.headnode03.cent  ms208c      verylong SUB9_G09.new      55255     1      8    --  3000:00:0 R 104:42:21
245446.headnode03.cent  gfb2y       verylong SUB4_G09.new      51079     1      8    --  3000:00:0 R 219:16:18
245463.headnode03.cent  ms208c      verylong SUB28_G09.new     30234     1      8    --  3000:00:0 R  42:51:12
245486.headnode03.cent  1002159f    verylong SUB1_G09.new       2989     1      8    --  3000:00:0 R 954:47:53
245740.headnode03.cent  gfb2y       verylong SUB23_G09.new     57724     1      8    --  3000:00:0 R  42:08:50
245752.headnode03.cent  gfb2y       verylong SUB2_G09.new      30469     1      8    --  3000:00:0 R  39:36:02
```

Passing the –n option gives further information as to the nodes the jobs are executing on. Torque notionally allocates jobs to each processor – the second line of each job tells you the nodes the job is running on.

```
[root@headnode04 export]# qstat -n

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                 Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
245017.headnode03.cent  gfb2y       verylong SUB10_G09.new      8598     1      8    --  3000:00:0 R 1777:06:5
   node072+node072+node072+node072+node072+node072+node072+node072
245070.headnode03.cent  ms208c      verylong SUB10_G09.new     10008     1      8    --  3000:00:0 R 1740:10:0
   node072+node072+node072+node072+node072+node072+node072+node072
245079.headnode03.cent  ms208c      verylong SUB20_G09.new      9468     1      8    --  3000:00:0 R 1933:17:2
   node065+node065+node065+node065+node065+node065+node065+node065
245089.headnode03.cent  ms208c      verylong SUB2_G09.new      10190     1      8    --  3000:00:0 R 1927:58:4
   node068+node068+node068+node068+node068+node068+node068+node068
245164.headnode03.cent  gfb2y       verylong SUB45_G09.new     18769     1      8    --  3000:00:0 R 1548:53:5
   node027+node027+node027+node027+node027+node027+node027+node027
245172.headnode03.cent  gfb2y       verylong SUB20_G09.new     33090     1      8    --  3000:00:0 R 1467:42:2
   node069+node069+node069+node069+node069+node069+node069+node069
245176.headnode03.cent  gfb2y       verylong SUB3_G09.new      57535     1      8    --  3000:00:0 R 1020:29:1
   node065+node065+node065+node065+node065+node065+node065+node065
245177.headnode03.cent  gfb2y       verylong SUB41_G09.new      8962     1      8    --  3000:00:0 R 977:00:47
   node071+node071+node071+node071+node071+node071+node071+node071
245204.headnode03.cent  ms208c      verylong SUB8_G09.new      51805     1      8    --  3000:00:0 R 1195:11:0
   node070+node070+node070+node070+node070+node070+node070+node070
245205.headnode03.cent  ms208c      verylong SUB_G09.new       45770     1      8    --  3000:00:0 R 1158:21:1
   node063+node063+node063+node063+node063+node063+node063+node063
245206.headnode03.cent  ms208c      verylong SUB33_G09.new     58638     1      8    --  3000:00:0 R 1082:46:5
   node067+node067+node067+node067+node067+node067+node067+node067
245351.headnode03.cent  ms208c      verylong SUB38_G09.new     30180     1      8    --  3000:00:0 R 295:04:19
   node032+node032+node032+node032+node032+node032+node032+node032
245352.headnode03.cent  ms208c      verylong SUB5_G09.new      27263     1      8    --  3000:00:0 R 295:08:40
   node031+node031+node031+node031+node031+node031+node031+node031
245358.headnode03.cent  ms208c      verylong SUB7_G09.new      43759     1      8    --  3000:00:0 R 227:04:58
   node069+node069+node069+node069+node069+node069+node069+node069
245370.headnode03.cent  1107184o    verylong SUB1_G09.new      59552     1      8    --  3000:00:0 R 1041:57:5
   node072+node072+node072+node072+node072+node072+node072+node072
245435.headnode03.cent  gfb2y       verylong SUB_G09.new       27488     1      8    --  3000:00:0 R 261:23:44
   node063+node063+node063+node063+node063+node063+node063+node063
245442.headnode03.cent  ms208c      verylong SUB39_G09.new      9191     1      8    --  3000:00:0 R 110:37:57
   node071+node071+node071+node071+node071+node071+node071+node071
245445.headnode03.cent  ms208c      verylong SUB9_G09.new      55255     1      8    --  3000:00:0 R 104:48:20
   node066+node066+node066+node066+node066+node066+node066+node066
245446.headnode03.cent  gfb2y       verylong SUB4_G09.new      51079     1      8    --  3000:00:0 R 219:31:12
   node062+node062+node062+node062+node062+node062+node062+node062
245463.headnode03.cent  ms208c      verylong SUB28_G09.new     30234     1      8    --  3000:00:0 R  43:03:11
   node064+node064+node064+node064+node064+node064+node064+node064
245464.headnode03.cent  ms208c      verylong SUB32_G09.new       --      1      8    --  3000:00:0 Q       --
    --
245465.headnode03.cent  ms208c      verylong SUB14_G09.new       --      1      8    --  3000:00:0 Q       --
    --
245466.headnode03.cent  ms208c      verylong SUB23_G09.new       --      1      8    --  3000:00:0 Q       --
    --
245467.headnode03.cent  ms208c      verylong SUB30_G09.new       --      1      8    --  3000:00:0 Q       --
    --
245486.headnode03.cent  1002159f    verylong SUB1_G09.new       2989     1      8    --  3000:00:0 R 954:59:52
   node066+node066+node066+node066+node066+node066+node066+node066
245531.headnode03.cent  ms208c      verylong SUB25_G09.new       --      1      8    --  3000:00:0 Q       --
    --
245589.headnode03.cent  ms208c      verylong SUB21_G09.new       --      1      8    --  3000:00:0 Q       --
    --
245740.headnode03.cent  gfb2y       verylong SUB23_G09.new     57724     1      8    --  3000:00:0 R  42:26:49
   node066+node066+node066+node066+node066+node066+node066+node066
245752.headnode03.cent  gfb2y       verylong SUB2_G09.new      30469     1      8    --  3000:00:0 R  39:46:50
   node064+node064+node064+node064+node064+node064+node064+node064
245755.headnode03.cent  gfb2y       verylong SUB32_G09.new     23118     1      8    --  3000:00:0 R  35:50:14
   node067+node067+node067+node067+node067+node067+node067+node067
245758.headnode03.cent  gfb2y       verylong SUB38_G09.new     12564     1      8    --  3000:00:0 R  35:05:33
   node070+node070+node070+node070+node070+node070+node070+node070
245759.headnode03.cent  gfb2y       verylong SUB25_G09.new      3754     1      8    --  3000:00:0 R  32:34:08
   node068+node068+node068+node068+node068+node068+node068+node068
245828.headnode03.cent  gfb2y       verylong SUB9_G09.new        --      1      8    --  3000:00:0 Q       --
    --
```

The next option I want to look at is the –f option (full) – I have chosen to give it a job number in the example below to limit the output, qstat –f on its own will give the information for every job in the queue – things to note here are the resources requested (Resource_List.*) and the resources used so far (resources_used.*)

Also the variable list & path lists can be useful in problem solving

```
[root@headnode04 export]# qstat -f 246074
Job Id: 246074.headnode03.cent.gla.ac.uk
    Job_Name = test2.sh
    Job_Owner = mjm4y@headnode04.cent.gla.ac.uk
    resources_used.cput = 00:00:00
    resources_used.mem = 4656kb
    resources_used.vmem = 251176kb
    resources_used.walltime = 00:07:07
    job_state = R
    queue = short
    server = headnode03.cent.gla.ac.uk
    Checkpoint = u
    ctime = Mon Dec  1 14:58:46 2014
    Error_Path = headnode04.cent.gla.ac.uk:/export/home/mjm4y/test2.sh.e246074

    exec_host = node058.hpc.gla.ac.uk/0+node058.hpc.gla.ac.uk/1+node057.hpc.gl
        a.ac.uk/0+node057.hpc.gla.ac.uk/1
    exec_port = 15003+15003+15003+15003
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Mon Dec  1 14:59:38 2014
    Output_Path = headnode04.cent.gla.ac.uk:/export/home/mjm4y/test2.sh.o24607
        4
    Priority = 0
    qtime = Mon Dec  1 14:58:46 2014
    Rerunable = True
    Resource_List.cput = 01:00:00
    Resource_List.neednodes = 2:ppn=2
    Resource_List.nodect = 2
    Resource_List.nodes = 2:ppn=2
    Resource_List.walltime = 01:00:00
    session_id = 8861
    substate = 42
    Variable_List = PBS_O_QUEUE=feed,PBS_O_HOME=/export/home/mjm4y,
        PBS_O_LOGNAME=mjm4y,
        PBS_O_PATH=/usr/kerberos/bin:/opt/PBS/bin:/usr/local/bin:/bin:/usr/bi
        n,PBS_O_MAIL=/var/spool/mail/mjm4y,PBS_O_SHELL=/bin/bash,
        PBS_O_LANG=en_US.UTF-8,PBS_O_WORKDIR=/export/home/mjm4y,
        PBS_O_HOST=headnode03.cent.gla.ac.uk,
        PBS_O_SERVER=headnode03.cent.gla.ac.uk
    euser = mjm4y
    egroup = users
    hashname = 246074.headnode03.cent.gla.ac.uk
    queue_rank = 2136
    queue_type = E
    etime = Mon Dec  1 14:58:46 2014
    submit_args = -l nodes=2:ppn=2 test2.sh
    start_time = Mon Dec  1 14:58:48 2014
    Walltime.Remaining = 3120
    start_count = 1
    fault_tolerant = False
    job_radix = 0
    submit_host = headnode04.cent.gla.ac.uk

[root@headnode04 export]#
```

The options –a, –i, -n  and –r can be combined with –u username to see only one users jobs

```
[root@headnode04 export]# qstat -n -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                                  Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
246074.headnode03.cent  mjm4y       short    test2.sh           8861     2      4    --   01:00:00 R  00:00:00
   node058+node058+node057+node057
[root@headnode04 export]#
```

There are a few more options which give useful summary information.

The –q option gives a readout detailing the queues on the system. Of note here are the current maximum cpu time(CPU Time)and maximum walltime (Walltime) for each queue. It also gives the number of jobs running (Run) and waiting (Que) in each queue. The Lm column should show the maximum number of running jobs, however it only shows the first 2 digits so the 1000 maximum running jobs for the short queue shows as 12 on this.

```
[root@headnode04 export]# qstat -q

server: node000.hpc.gla.ac.uk

Queue            Memory CPU Time Walltime Node  Run Que Lm  State
---------------- ------ -------- -------- ----  --- --- --  -----
ottomsc            --   50:00:00 24:00:00   --    0   0 10   E R
bioinf-stud2       --      --       --      --    0   0 10   E R
biobank            --      --       --      --    0   0 96   E R
bioinf-stud        --      --       --      --    0   0 40   E R
long               --   500:00:0 50:00:00   --   13   1 12   E R
veryparrallel      --      --       --      --    0   0  4   E R
bioinf-stud3       --      --       --      --    0   0  4   E R
verylong           --      --       --      --   15  28 12   E R
gpu                --      --   192:00:0    --    0   0  7   E R
parallel           --      --       --      --    0   1 17   E R
feed               --      --       --      --    0 941 --   E R
short              --   10:00:00 02:00:00   --    0   1 10   E R
bioinf-verylong    --      --       --      --    4   0 96   E R
bioinf-parallel    --      --       --      --    0   0  5   E R
                                               ----- -----
                                                  32   972
```

Another summary option is –Q. Max is maximum number of running jobs, Tot is the number currently running Ena & Str is enabled and started, should always be yes or there is a problem somewhere. Then the next columns represent the various possible states of jobs Que is waiting in the queue, Run is currently running, Hld is job held for some reason, Wat is the job waiting for something to happen before it can proceed, Trn is job transiting and Ext is job exiting.

```
[root@headnode04 export]# qstat -Q
Queue              Max    Tot   Ena   Str   Que   Run   Hld   Wat   Trn   Ext T   Cpt
----------------   ---   ----    --    --   ---   ---   ---   ---   ---   --- -   ---
ottomsc            100      0   yes   yes     0     0     0     0     0     0 E     0
bioinf-stud2        10      0   yes   yes     0     0     0     0     0     0 E     0
biobank             96      0   yes   yes     0     0     0     0     0     0 E     0
bioinf-stud         40      0   yes   yes     0     0     0     0     0     0 E     0
long               125     16   yes   yes     1    13     0     0     0     0 E     2
veryparrallel        4      0   yes   yes     0     0     0     0     0     0 E     0
bioinf-stud3         4      0   yes   yes     0     0     0     0     0     0 E     0
verylong           120     46   yes   yes    28    15     0     0     0     0 E     3
gpu                  7      0   yes   yes     0     0     0     0     0     0 E     0
parallel            17      1   yes   yes     1     0     0     0     0     0 E     0
feed                 0    941   yes   yes    24     0   917     0     0     0 R     0
short             1000      1   yes   yes     1     0     0     0     0     0 E     0
bioinf-verylong     96      4   yes   yes     0     4     0     0     0     0 E     0
bioinf-parallel      5      0   yes   yes     0     0     0     0     0     0 E     0
[root@headnode04 export]#
```

The final option I want to discuss just now is the –Qf option which gives a verbose description of each queue. The additional information this gives is the additional properties currently assigned to each queue – resources_max – the maximum possible for a job running on that queue, resourses_default – the resources assigned to a job entering that queue unless something different has been requested. – resources_min – the minimum resources request needed before a job will be assigned to this queue (if a particular resource is not specified at submit time then the minimum check is not applied).

```
[root@headnode04 export]# qstat -Qf
Queue: ottomsc
    queue_type = Execution
    max_queuable = 200
    max_user_queuable = 3
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 100
    resources_max.cput = 50:00:00
    resources_max.procct = 4
    resources_max.walltime = 24:00:00
    resources_min.procct = 1
    resources_default.cput = 01:00:00
    resources_default.procct = 1
    resources_default.walltime = 01:00:00
    acl_group_enable = True
    acl_groups = ottomsc
    acl_group_sloppy = True
    mtime = 1715596153
    enabled = True
    started = True

Queue: bioinf-stud2
    queue_type = Execution
    max_queuable = 20
    max_user_queuable = 3
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 10
    resources_max.procct = 16
    resources_min.procct = 5
    resources_default.cput = 01:00:00
    resources_default.procct = 5
    resources_default.walltime = 01:00:00
    acl_group_enable = True
    acl_groups = bioinf-stud
    acl_group_sloppy = True
    mtime = 1715596153
    enabled = True
    started = True

Queue: biobank
    queue_type = Execution
    max_queuable = 100
    max_user_queuable = 25
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 96
    resources_min.cput = 24:00:01
    resources_min.walltime = 24:00:01
    resources_default.cput = 288:00:00
    resources_default.procct = 1
    resources_default.walltime = 288:00:00
    acl_group_enable = True
    acl_groups = biobank
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.mem = 0b
    resources_assigned.nodect = 0
    enabled = True
    started = True
```

```
Queue: bioinf-stud
    queue_type = Execution
    max_queuable = 100
    max_user_queuable = 5
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 40
    resources_max.procct = 4
    resources_default.cput = 01:00:00
    resources_default.procct = 1
    resources_default.walltime = 01:00:00
    acl_group_enable = True
    acl_groups = bioinf-stud
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.nodect = 0
    enabled = True
    started = True

Queue: long
    queue_type = Execution
    max_queuable = 400
    max_user_queuable = 50
    total_jobs = 16
    state_count = Transit:0 Queued:1 Held:0 Waiting:0 Running:13 Exiting:0 Com
        plete:2
    max_running = 125
    resources_max.cput = 500:00:00
    resources_max.procct = 16
    resources_max.walltime = 50:00:00
    resources_default.cput = 24:00:00
    resources_default.procct = 1
    resources_default.walltime = 24:00:00
    acl_group_enable = True
    acl_groups = research-staff,res-studs
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.nodect = 15
    enabled = True
    started = True

Queue: veryparrallel
    queue_type = Execution
    max_queuable = 10
    max_user_queuable = 2
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 4
    resources_max.procct = 128
    resources_min.procct = 17
    resources_default.nodes = 2
    resources_default.procct = 33
    resources_default.walltime = 288:00:00
    acl_group_enable = True
    acl_groups = research-staff,res-studs
    acl_group_sloppy = True
    mtime = 1715596153
    enabled = True
    started = True

Queue: bioinf-stud3
    queue_type = Execution
    max_queuable = 10
    max_user_queuable = 1
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 4
    resources_max.procct = 32
    resources_min.procct = 17
    resources_default.cput = 01:00:00
    resources_default.procct = 17
    resources_default.walltime = 01:00:00
    acl_group_enable = True
    acl_groups = bioinf-stud
    acl_group_sloppy = True
    mtime = 1715596153
    enabled = True
    started = True

Queue: verylong
    queue_type = Execution
    max_queuable = 250
    max_user_queuable = 35
    total_jobs = 46
    state_count = Transit:0 Queued:28 Held:0 Waiting:0 Running:15 Exiting:0 Co
       mplete:3
    max_running = 120
    resources_max.procct = 16
    resources_min.cput = 24:00:01
    resources_min.walltime = 24:00:01
    resources_default.cput = 288:00:00
    resources_default.procct = 1
    resources_default.walltime = 288:00:00
    acl_group_enable = True
    acl_groups = research-staff,res-studs
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.nodect = 15
    enabled = True
    started = True

Queue: gpu
    queue_type = Execution
    max_queuable = 200
    max_user_queuable = 25
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
       lete:0
    max_running = 7
    resources_max.procct = 4
    resources_max.walltime = 192:00:00
    resources_default.procct = 1
    resources_default.walltime = 24:00:00
    acl_group_enable = True
    acl_groups = gpu-users
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.nodect = 0
    enabled = True
    started = True

Queue: parallel
    queue_type = Execution
    max_queuable = 50
    max_user_queuable = 10
    total_jobs = 1
    state_count = Transit:0 Queued:1 Held:0 Waiting:0 Running:0 Exiting:0 Comp
       lete:0
    max_running = 17
    resources_max.procct = 32
    resources_min.procct = 17
    resources_default.nodes = 2
    resources_default.procct = 17
    resources_default.walltime = 288:00:00
    acl_group_enable = True
    acl_groups = research-staff,res-studs
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.nodect = 0
    enabled = True
    started = True

Queue: feed
    queue_type = Route
    total_jobs = 941
    state_count = Transit:0 Queued:24 Held:917 Waiting:0 Running:0 Exiting:0 C
       omplete:0
    mtime = 1715596153
    route_destinations = short,long,bioinf-stud,bioinf-stud2,bioinf-stud3,
       bioinf-verylong,bioinf-parallel,biobank,verylong,parallel,veryparrallel,
        ottomsc
    enabled = True
    started = True

Queue: short
    queue_type = Execution
    max_queuable = 1500
    max_user_queuable = 175
    total_jobs = 1
    state_count = Transit:0 Queued:1 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 1000
    resources_max.cput = 10:00:00
    resources_max.procct = 16
    resources_max.walltime = 02:00:00
    resources_default.cput = 01:00:00
    resources_default.procct = 1
    resources_default.walltime = 01:00:00
    acl_group_enable = True
    acl_groups = research-staff,res-studs
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.nodect = 0
    enabled = True
    started = True

Queue: bioinf-verylong
    queue_type = Execution
    max_queuable = 300
    max_user_queuable = 25
    total_jobs = 4
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:4 Exiting:0 Comp
        lete:0
    max_running = 96
    resources_max.procct = 40
    resources_min.cput = 24:00:01
    resources_min.walltime = 24:00:01
    resources_default.cput = 288:00:00
    resources_default.procct = 1
    resources_default.walltime = 288:00:00
    acl_group_enable = True
    acl_groups = bioinf-staff,bioinf-students,bioinf-stud
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.mem = 365072220160b
    resources_assigned.nodect = 4
    enabled = True
    started = True

Queue: bioinf-parallel
    queue_type = Execution
    max_queuable = 20
    total_jobs = 0
    state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Comp
        lete:0
    max_running = 5
    resources_max.procct = 40
    resources_min.procct = 17
    resources_default.procct = 17
    resources_default.walltime = 288:00:00
    acl_group_enable = True
    acl_groups = bioinf-staff,bioinf-students,bioinf-stud
    acl_group_sloppy = True
    mtime = 1715596153
    resources_assigned.nodect = 0
    enabled = True
    started = True

[root@headnode04 export]#
```

# Useful Commands in PBS/TORQUE & Maui

## showq

### /opt/maui/bin/showq

The command showq comes as part of the maui scheduler package. This can be useful if there are jobs waiting on the queue & you want to get an idea of when your job may start particularly useful if you require a number of processors – the order it lists active jobs is starting with the job that will finish first. It also gives the percentage of the cluster that is currently being used – in this case 50.84% of the processors and 100.00% of the nodes. There is an option which can be passed to showq:-   -b will give you a list of blocked jobs  with a reason why the job is blocked.

```
[root@headnode03 home2]# showq
ACTIVE JOBS--------------------
JOBNAME            USERNAME      STATE  PROC   REMAINING            STARTTIME

552792                gfb2y    Running     8     5:20:42  Sun May 26 21:53:57
552793                gfb2y    Running     8    11:53:53  Mon May 27 04:27:08
552794                gfb2y    Running     8    12:03:33  Mon May 27 04:36:48
552799                gfb2y    Running     8    19:54:14  Mon May 27 12:27:29
552796                gfb2y    Running     8    20:55:12  Mon May 27 13:28:27
552800                gfb2y    Running     8    20:55:12  Mon May 27 13:28:27
552801                gfb2y    Running     8    21:44:17  Mon May 27 14:17:32
552686                gfb2y    Running     8    22:03:55  Mon May 27 14:37:10
552795                gfb2y    Running     8    22:13:44  Mon May 27 14:46:59
552784                gfb2y    Running    10  1:00:01:12  Mon May 27 16:34:27
552808                hx10z    Running     1  1:04:23:40  Mon May 27 20:56:55
552813                tss7r    Running    16  1:14:13:30  Tue May 28 09:46:45
552839[17]         2630252a    Running     8  1:22:26:29  Tue May 28 16:59:44
552839[18]         2630252a    Running     8  1:22:28:33  Tue May 28 17:01:48
552839[21]         2630252a    Running     8  1:22:45:36  Tue May 28 17:18:51
552839[22]         2630252a    Running     8  1:22:55:25  Tue May 28 17:28:40
552839[23]         2630252a    Running     8  1:23:14:32  Tue May 28 17:47:47
552839[57]         2630252a    Running     8  1:23:30:33  Tue May 28 18:03:48
552839[24]         2630252a    Running     8  1:23:43:59  Tue May 28 18:17:14
552839[25]         2630252a    Running     8  1:23:46:03  Tue May 28 18:19:18
552814             2762262d    Running    20  2:03:23:50  Tue May 28 09:57:05
549874                gfb2y    Running     8 13:01:03:16  Fri Apr 19 17:36:31
552639               jw269k    Running    11 16:08:10:33  Sun May 26 08:43:48
552646               jw269k    Running    11 16:11:21:12  Sun May 26 11:54:27
551791                gfb2y    Running     8 23:19:39:21  Tue Apr 30 12:12:36
551873                gfb2y    Running     8 23:19:52:40  Tue Apr 30 12:25:55
551935                gfb2y    Running     8 27:03:31:14  Fri May  3 20:04:29
552810             2762262d    Running    20 32:12:32:50  Mon May 27 23:06:05
552002                gfb2y    Running     8 38:22:45:14  Wed May 15 15:18:29
552495                gfb2y    Running     8 50:13:17:57  Mon May 27 05:51:12
551957                gfb2y    Running     8    INFINITY  Sat May  4 15:09:56
552673             2188196m    Running    16    INFINITY  Wed May 22 12:46:54

    32 Active Jobs     297 of 1027 Processors Active (28.92%)
                        18 of   25 Nodes Active      (72.00%)

IDLE JOBS----------------------
JOBNAME            USERNAME      STATE  PROC     WCLIMIT            QUEUETIME

552839[26]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[27]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[28]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[29]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[30]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[31]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[32]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[33]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[34]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
```

```
552839[35]         2630252a       Idle     8  2:00:00:00  Tue May 28 14:07:37
552839[36]         2630252a       Idle     8  2:00:00:00  Tue May 28 15:06:56
552839[37]         2630252a       Idle     8  2:00:00:00  Tue May 28 15:12:28
552839[39]         2630252a       Idle     8  2:00:00:00  Tue May 28 15:25:22
552839[40]         2630252a       Idle     8  2:00:00:00  Tue May 28 15:29:33
552839[42]         2630252a       Idle     8  2:00:00:00  Tue May 28 15:44:08
552839[43]         2630252a       Idle     8  2:00:00:00  Tue May 28 15:48:39
552839[44]         2630252a       Idle     8  2:00:00:00  Tue May 28 16:05:04
552839[45]         2630252a       Idle     8  2:00:00:00  Tue May 28 16:07:05
552839[46]         2630252a       Idle     8  2:00:00:00  Tue May 28 16:07:45
552839[49]         2630252a       Idle     8  2:00:00:00  Tue May 28 16:59:51
552839[50]         2630252a       Idle     8  2:00:00:00  Tue May 28 17:01:51
552839[51]         2630252a       Idle     8  2:00:00:00  Tue May 28 17:08:53
552839[52]         2630252a       Idle     8  2:00:00:00  Tue May 28 17:17:56
552839[53]         2630252a       Idle     8  2:00:00:00  Tue May 28 17:18:46
552839[55]         2630252a       Idle     8  2:00:00:00  Tue May 28 17:28:29
552839[56]         2630252a       Idle     8  2:00:00:00  Tue May 28 17:47:55
552839[58]         2630252a       Idle     8  2:00:00:00  Tue May 28 18:16:54
552839[59]         2630252a       Idle     8  2:00:00:00  Tue May 28 18:19:15

28 Idle Jobs

BLOCKED JOBS----------------
JOBNAME            USERNAME      STATE  PROC     WCLIMIT            QUEUETIME

545726                gy11w       Idle     1    12:00:00  Tue Mar  5 07:27:17
550513             2860910l  BatchHold    20  4:03:00:00  Mon Apr 15 15:25:59
550531             2630252a  BatchHold     1     1:00:00  Mon Apr 15 19:39:31

Total Jobs: 63   Active Jobs: 32   Idle Jobs: 28   Blocked Jobs: 3
```

Showq –b gives the list of blocked jobs along with a reason why the job is blocked.

```
[root@headnode03 home2]# showq -b
           JobName      User              Reason

            245464    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245465    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245466    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245467    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245531    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245589    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245828     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245831    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245833    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245895     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245899     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245900     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245903     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245919    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245904     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245908     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245915     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245994     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            245997     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246003     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246010     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246011     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246015     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246022     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246024    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246025    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246026    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246029    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246035    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246038    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246039    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246047    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246048    ms208c violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246051     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246052     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
            246053     gfb2y violates active HARD MAXJOB limit of 12 for class verylong user (R: 1, U: 12)
[root@headnode04 home2]#
```

# Useful Commands in PBS/TORQUE & Maui

## Additional useful Torque commannds

### tracejob

#### /usr/local/bin/tracejob

This command will trawl the log files and return the results. Important options are –n the number of days to look back into the log (ie tracejob –n 20 13049    look back 20 days in the logs), the default is one day.

```
[test@headnode01 test]$ tracejob 13049

Job: 13049.headnode01.cent.gla.ac.uk

12/02/2005 13:34:06  S    enqueuing into feed, state 1 hop 1
12/02/2005 13:34:06  S    dequeuing from feed, state QUEUED
12/02/2005 13:34:06  S    enqueuing into verylong, state 1 hop 1
12/02/2005 13:34:06  S    Job Queued at request of test@headnode01.cent.gla.ac.uk,
owner = test@headnode01.cent.gla.ac.uk, job name =
                          serandite_poten, queue = verylong
12/02/2005 13:34:08  S    Job Modified at request of root@headnode01.cent.gla.ac.uk
12/02/2005 13:34:08  S    Job Run at request of root@headnode01.cent.gla.ac.uk
12/02/2005 13:34:08  S    Job Modified at request of root@headnode01.cent.gla.ac.uk
[test@headnode01 test]$
```

### qdel

#### /usr/local/bin/qdel

This command takes a jobid as a parameter and removes that job from the queue, terminating job execution if required.

```
-bash-3.2$ qstat -a -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                      Req'd    Req'd       Elap
Job ID                 Username    Queue    Jobname          SessID  NDS   TSK    Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247224.headnode03.cent  mjm4y       short    one.sh           22776     1      1    --   01:00:00 R       --
247225.headnode03.cent  mjm4y       short    one.sh            6419     1      1    --   01:00:00 R       --
247226.headnode03.cent  mjm4y       short    one.sh           20545     1      1    --   01:00:00 R       --

-bash-3.2$ qdel 247225

-bash-3.2$ qstat -a -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                      Req'd    Req'd       Elap
Job ID                 Username    Queue    Jobname          SessID  NDS   TSK    Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247224.headnode03.cent  mjm4y       short    one.sh           22776     1      1    --   01:00:00 R       --
247225.headnode03.cent  mjm4y       short    one.sh            6419     1      1    --   01:00:00 C  00:00:00
247226.headnode03.cent  mjm4y       short    one.sh           20545     1      1    --   01:00:00 R       --
```

There is another option –p which is available to administrators to purge a job which cannot be deleted by a user. If you are having difficulty deleting a job, mail me at
 it-hpc-support@glasgow.ac.uk.

## pbsnodes

### /usr/local/bin/pbsnodes

You would run this command with the –a option and you will get a list of all the nodes with details of their configuration, you can also give it a node and the output will be restricted to that node. The output consists of the state of the node (free – available for a job, jobexclusive – all slots currently allocated to running jobs and down – unavailable. If there are jobs running on the node the optional line jobs = is shown with the jobid(s) of the running jobs. np = gives the number of job slots and status gives details of the arch, OS, memory, number of cpu's and current load on the machine.  The properties line gives a list of any properties we have assigned to a node, these can be used when you request resources for a job. Ie   qsub –l nodes=1:eightgpc will request a node with the property "eightgpc" (eg node059 below)

```
-bash-3.2$ /usr/local/bin/pbsnodes -a node059.hpc.gla.ac.uk node060.hpc.gla.ac.uk node061.hpc.gla.ac.uk
node059.hpc.gla.ac.uk
     state = free
     np = 64
     properties = centos7,eightgpc,cpunode,short,long,staging,bio7
     ntype = cluster
     jobs =
0/551935.headnode03.cent.gla.ac.uk,1/551935.headnode03.cent.gla.ac.uk,2/551935.headnode03.cent.gla.ac.uk,3/551935.h
eadnode03.cent.gla.ac.uk,4/551935.headnode03.cent.gla.ac.uk,5/551935.headnode03.cent.gla.ac.uk,6/551935.headnode03.
cent.gla.ac.uk,7/551935.headnode03.cent.gla.ac.uk
     status =
rectime=1716386591,varattr=,jobs=551935.headnode03.cent.gla.ac.uk,state=free,netload=75913472486358,gres=,loadave=8
.94,ncpus=64,physmem=528167776kb,availmem=564158048kb,totmem=586761052kb,idletime=3124807,nusers=2,nsessions=2,sess
ions=52512 61439,uname=Linux node059.hpc.gla.ac.uk 3.10.0-1160.95.1.el7.x86_64 #1 SMP Mon Jul 24 13:59:37 UTC 2023
x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003

node060.hpc.gla.ac.uk
     state = free
     np = 64
     properties = centos7,eightgpc,cpunode,short,long,staging,bio7
     ntype = cluster
     jobs =
0/551873.headnode03.cent.gla.ac.uk,1/551873.headnode03.cent.gla.ac.uk,2/551873.headnode03.cent.gla.ac.uk,3/551873.h
eadnode03.cent.gla.ac.uk,4/551873.headnode03.cent.gla.ac.uk,5/551873.headnode03.cent.gla.ac.uk,6/551873.headnode03.
cent.gla.ac.uk,7/551873.headnode03.cent.gla.ac.uk
     status =
rectime=1716386606,varattr=,jobs=551873.headnode03.cent.gla.ac.uk,state=free,netload=134143534046165,gres=,loadave=
7.93,ncpus=64,physmem=528167776kb,availmem=565146984kb,totmem=586761052kb,idletime=3112126,nusers=2,nsessions=2,ses
sions=16091 60767,uname=Linux node060.hpc.gla.ac.uk 3.10.0-1160.95.1.el7.x86_64 #1 SMP Mon Jul 24 13:59:37 UTC 2023
x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003

node061.hpc.gla.ac.uk
     state = free
     np = 64
     properties = centos7,eightgpc,cpunode,short,long,staging,bio7
     ntype = cluster
     jobs =
0/551791.headnode03.cent.gla.ac.uk,1/551791.headnode03.cent.gla.ac.uk,2/551791.headnode03.cent.gla.ac.uk,3/551791.h
eadnode03.cent.gla.ac.uk,4/551791.headnode03.cent.gla.ac.uk,5/551791.headnode03.cent.gla.ac.uk,6/551791.headnode03.
cent.gla.ac.uk,7/551791.headnode03.cent.gla.ac.uk,8/552677.headnode03.cent.gla.ac.uk,9/552677.headnode03.cent.gla.a
c.uk,10/552677.headnode03.cent.gla.ac.uk,11/552677.headnode03.cent.gla.ac.uk,12/552677.headnode03.cent.gla.ac.uk,13
/552677.headnode03.cent.gla.ac.uk,14/552677.headnode03.cent.gla.ac.uk,15/552677.headnode03.cent.gla.ac.uk,16/552677
.headnode03.cent.gla.ac.uk,17/552677.headnode03.cent.gla.ac.uk,18/552677.headnode03.cent.gla.ac.uk,19/552677.headno
de03.cent.gla.ac.uk,20/552677.headnode03.cent.gla.ac.uk,21/552677.headnode03.cent.gla.ac.uk,22/552677.headnode03.ce
nt.gla.ac.uk,23/552677.headnode03.cent.gla.ac.uk
     status = rectime=1716386591,varattr=,jobs=551791.headnode03.cent.gla.ac.uk
552677.headnode03.cent.gla.ac.uk,state=free,netload=61309435597730,gres=,loadave=22.75,ncpus=64,physmem=528167776kb
,availmem=574136480kb,totmem=590664540kb,idletime=602943,nusers=3,nsessions=3,sessions=19327 49203
51820,uname=Linux node061.hpc.gla.ac.uk 3.10.0-1160.88.1.el7.x86_64 #1 SMP Tue Mar 7 15:41:52 UTC 2023
x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003
```

# Useful Commands in PBS/TORQUE & Maui

## Other Useful Maui Commands

## showstart

#### /opt/maui/bin/showstart

This command gives an indication of when a job that is in the queue will start, it takes the jobid as a parameter and returns output which will give an estimate of the likely start & end time of your job (based on walltime). The example I have given below is for a job that is already running so it gives a negative time to indicate that it is in the past. It also gives the expected finish time (this is based on resources requested and not any qualitative assessment of how well the job is running).

```
[test@headnode01 bin]$ ./showstart 12727
job 12727 requires 10 procs for 12:12:00:00
Earliest start in      -4:12:49:21 on Mon Nov 28 00:05:35
Earliest completion in  7:23:10:39 on Sat Dec 10 12:05:35
Best Partition: DEFAULT
```

## checkjob

#### /opt/maui/bin/checkjob

This allows you to check the status of a job that is currently running. Pass the command a parameter of the jobid you wish to check.

```
[test@headnode01 bin]$ ./checkjob 12727

checking job 12727

State: Running
Creds:  user:test  group:test  class:verylong  qos:DEFAULT
WallTime: 4:12:54:53 of 12:12:00:00
SubmitTime: Mon Nov 28 00:01:38
  (Time Queued  Total: 00:03:57  Eligible: 00:00:02)

StartTime: Mon Nov 28 00:05:35
Total Tasks: 10

Req[0]  TaskCount: 10  Partition: DEFAULT
Network: [NONE]  Memory >= 0  Disk >= 0  Swap >= 0
Opsys: [NONE]  Arch: [NONE]  Features: [NONE]
NodeCount: 6
Allocated Nodes:
[comp06:2][comp07:1][comp09:1][comp10:2]
[comp11:2][comp12:2]


IWD: [NONE]  Executable:  [NONE]
Bypass: 0  StartCount: 1
PartitionMask: [ALL]
Flags:       RESTARTABLE

Reservation '12727' (-4:12:50:23 -> 7:23:09:37  Duration: 12:12:00:00)
PE:  10.00  StartPriority:  1

[test@headnode01 bin]$
```

## canceljob

### /opt/maui/bin/canceljob

This command takes a list of jobid’s as parameters and cancel’s those jobs

```
[test@headnode01 bin]$ ./canceljob 13050 13051 13052


job '13050' cancelled
job '13051' cancelled
job '13052' cancelled

[test@headnode01 bin]$
```

## pbstop

### /usr/bin/pbstop

This is quite a complicated command that shows the state of the cluster – it was written some time ago and the default behaviour gives confusing output so I would suggest running it with the options “-n -c 64 -m 1” this will give a section on each node showing what processors are in use. It refreshes every 20 seconds, you can use page up and page down to scroll through the output and should press q to quit.

```
pbstop  -n -c 64 -m 1
```

*see the manual pages for full information*

```
man pbstop
```

# Advanced Commands in PBS/TORQUE

## qsub – advanced part one

Earlier we looked at the most commonly used options –I, -l, -m, -M, -N. To this I will add –W which allows you to specify some additional attributes, and –t which introduces array jobs.

### Qsub options

#### -t

This allows you to submit a group of similar/identical jobs as one and, optionally, restrict the number of them running at any one time. You follow –t with a numeric range and/or discrete numbers e.g

> -t 0-5
> will run 6 array tasks with array ids - 0,1,2,3,4,5
>
> -t 3,6,15
> will run 3 array tasks with array ids – 3,6,15
>
> -t 1-4,10,12-13
> will run 7 array tasks with array ids - 1,2,3,4,10,12,13

You can, optionally, restrict the array job to a maximum number of running tasks at any one time by adding %n to the end of the command – where n is the maximum number you want running. E.g.

> -t 0-5%2
> Will run 6 array tasks but no more than 2 at any one time

Here are the commands as you would put them in a script

```
#!/bin/bash

#PBS -l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2:centos7
#PBS –m abe
#PBS –M mjm4y@udcf.gla.ac.uk
#PBS –N TestArray
#PBS –t 0-20%5
/path/to/programme_to_run
```

So this script submits an array job of 21 tasks but only allows a maximum of 5 to run at any one time. All 20 tasks will have the same Job ID but with a suffix of the array id

Ie 12345[0], 12345[1], ... 12345[19],12345[20]

On the execution node your task will run with the environment variable $PBS_JOBID equal to, for example, 12345[9].headnode03.cent.gla.ac.uk.

You have an additional environment variable - $PBS_ARRAYID - which you can use to distinguish between the different job tasks with the job array from within your script.

Here is a simple example I put together to illustrate what you might want to do. I have created a control file to control what each element of the job array does.

Here is the contents of arrayjob.sh

```
#! /bin/bash

#PBS -l cput=01:30:00,walltime=02:00:00,nodes=1:ppn=2:centos7
#PBS -m abe
#PBS -M mjm4y@udcf.gla.ac.uk
#PBS -N TestArray
#PBS -t 1-4%2

JOB=`grep $PBS_ARRAYID /export/home/mjm4y/array-job-test/array-jobs-files | awk '{print $2}'`
LOG=`grep $PBS_ARRAYID /export/home/mjm4y/array-job-test/array-jobs-files | awk '{print $3}'`
/export/home/mjm4y/array-job-test/$JOB > /export/home/mjm4y/array-job-test/$LOG
```

Here we are submitting a four task array which has a maximum of 2 running tasks. It uses the PBS_ARRAYID to query an input file for which job to run and the logfile to put the results in.

The control file array-job-files consists of the following:-

```
1 one.sh first.log
2 two.sh second.log
3 three.sh third.log
4 four.sh fourth.log
```

You would submit arrayjob.sh as below (it picks up all the PBS commands from the top of the script).

```
qsub arrayjob.sh
```

For each line in the control file the first entry is the array id, the second is the script to run and the third is the name of the log file. Task 1 of the array job runs one.sh and puts the output into the file first.log, task 2 runs two.sh and the output goes into second.log etc.

## qstat  advanced - arrays

If you try to query the running jobs with qstat you will see that the job array is listed as a single job, however you are probably interested in the individual tasks of the job array.  To see the individual tasks you use the –t option, this can be combined with the options –r, -i. –e, -a, -n and will expand the array to show individual tasks. For example here is the output of qstat –a and qstat –at for 2 array jobs of 4 tasks each

```
[root@headnode04 ~]# qstat -a

headnode03.cent.gla.ac.uk:
                                                                                         Req'd    Req'd      Elap
Job ID                    Username    Queue    Jobname          SessID  NDS    TSK    Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ----------------------------------------
246447[].headnode03.ce  mjm4y      short    TestArray         --     1     2        --       01:30:00 R       --
246448[].headnode03.ce  mjm4y      short    TestArray         --     1     2        --       01:30:00 R       --
```

```
[root@headnode04 ~]# qstat -at

headnode03.cent.gla.ac.uk:
                                                                                            Req'd    Req'd      Elap
Job ID                    Username    Queue    Jobname          SessID      NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ------------------------------------------
246447[1].headnode03.c  mjm4y      short    TestArray-1     31763    1     2        --       01:30:00 R      00:23:17
246447[2].headnode03.c  mjm4y      short    TestArray-2      6968    1     2        --       01:30:00 R      00:23:16
246447[3].headnode03.c  mjm4y      short    TestArray-3       --      1     2        --       01:30:00 H      --
246447[4].headnode03.c  mjm4y      short    TestArray-4       --      1     2        --      01:30:00 H      --
246448[1].headnode03.c  mjm4y      short    TestArray-1      5377    1     2        --      01:30:00 R      00:23:10
246448[2].headnode03.c  mjm4y      short    TestArray-2     22858    1     2        --      01:30:00 R      00:23:10
246448[3].headnode03.c  mjm4y      short    TestArray-3       --      1     2        --      01:30:00 H      --
246448[4].headnode03.c  mjm4y      short    TestArray-4       --      1     2        --      01:30:00 H      --
```

Note there is a new status – H – which stands for job held, in this case because we have stated that there will be a maximum of two tasks from the array running at any one time, so for each job array there are two running tasks and two held tasks – as the running jobs finish the held jobs will start.

For  qstat –f you need to give the job id  & array id. Ie qstat –f   26447[3]

## tracejob advanced - arrays

With arrays you need to approach tracejob a little differently if you use it as before then you get all the information about all the tasks in one section and it is not clear which line belongs to which task

```
-bash-3.2$ tracejob 247213

Job: 247213[].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into feed, state 1 hop 1
12/16/2023 13:02:09  S    dequeuing from feed, state QUEUED
12/16/2023 13:02:09  S    enqueuing into short, state 1 hop 1
12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Run at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Run at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
```

Instead what you can do is specify which task you would like to check as follows – I have done this for all four tasks (the array is currently running)

```
-bash-3.2$ tracejob 247213[1]

Job: 247213[1].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Run at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk


-bash-3.2$ tracejob 247213[2]

Job: 247213[2].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Run at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk


-bash-3.2$ tracejob 247213[3]

Job: 247213[3].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1


-bash-3.2$ tracejob 247213[4]

Job: 247213[4].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
```

Note if you give the job id with the brackets but no array id – as follows you get info on the array job as a whole and nothing about the actual tasks

```
-bash-3.2$ tracejob 247213[]

Job: 247213[].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into feed, state 1 hop 1
12/16/2023 13:02:09  S    dequeuing from feed, state QUEUED
12/16/2023 13:02:09  S    enqueuing into short, state 1 hop 1
```

Then when the array job is complete you would see this:-

```
-bash-3.2$ tracejob 247213[]

Job: 247213[].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into feed, state 1 hop 1
12/16/2023 13:02:09  S    dequeuing from feed, state QUEUED
12/16/2023 13:02:09  S    enqueuing into short, state 1 hop 1
12/16/2023 13:10:31  S    dequeuing from short, state COMPLETE
```

i.e. nothing is reported for the array tasks – here is the output for an array task that has completed.

```
-bash-3.2$ tracejob 247213[1]

Job: 247213[1].headnode03.cent.gla.ac.uk

12/16/2023 13:02:09  S    enqueuing into short, state 2 hop 1
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Run at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:02:10  S    Job Modified at request of root@headnode03.cent.gla.ac.uk
12/16/2023 13:03:50  S    Exit_status=0 resources_used.cput=00:00:00
resources_used.mem=3980kb resources_used.vmem=252752kb
                          resources_used.walltime=00:01:40
12/16/2023 13:03:50  S    on_job_exit valid pjob: 247213[1].headnode03.cent.gla.ac.uk
(substate=50)
12/16/2023 13:08:52  S    dequeuing from short, state COMPLETE
```

## qdel advanced - arrays

If you want to delete an array job or individual tasks in an array you need to change the jobid you pass to qdel.

Here is an example where we submit an array job as before – we can see it running and we try to delete it with qdel 247214

```
-bash-3.2$ qsub arrayjob2.sh
247214[].headnode03.cent.gla.ac.uk


-bash-3.2$ qstat -at -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                 Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247214[1].headnode03.c  mjm4y       short    TestArray-1       17634     1      2    --   01:30:00 R       --
247214[2].headnode03.c  mjm4y       short    TestArray-2        1174     1      2    --   01:30:00 R       --
247214[3].headnode03.c  mjm4y       short    TestArray-3          --     1      2    --   01:30:00 H       --
247214[4].headnode03.c  mjm4y       short    TestArray-4          --     1      2    --   01:30:00 H       --


-bash-3.2$ qdel 247214


-bash-3.2$ qstat -at -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                 Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247214[1].headnode03.c  mjm4y       short    TestArray-1       17634     1      2    --   01:30:00 R       --
247214[2].headnode03.c  mjm4y       short    TestArray-2        1174     1      2    --   01:30:00 R       --
247214[3].headnode03.c  mjm4y       short    TestArray-3          --     1      2    --   01:30:00 H       --
247214[4].headnode03.c  mjm4y       short    TestArray-4          --     1      2    --   01:30:00 H       --
```

As you can see the arrayjob is still running – instead we need to pass qdel an option which tells it that this is an array job – qdel 247214[]

```
-bash-3.2$ qdel 247214[]

-bash-3.2$ qstat -at -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                 Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247214[1].headnode03.c  mjm4y       short    TestArray-1       17634     1      2    --   01:30:00 C  00:00:00
247214[2].headnode03.c  mjm4y       short    TestArray-2        1174     1      2    --   01:30:00 C  00:00:00
247214[3].headnode03.c  mjm4y       short    TestArray-3          --     1      2    --   01:30:00 C       --
247214[4].headnode03.c  mjm4y       short    TestArray-4          --     1      2    --   01:30:00 C       --
```

Here we see that all tasks in the array have been stopped.

However you may only want to stop one task in your array and let the others continue, to do this we need to tell qdel which task we want to delete – so let’s submit another arrayjob

```
-bash-3.2$ qsub arrayjob2.sh
247215[].headnode03.cent.gla.ac.uk


-bash-3.2$ qstat -at -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                 Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
---------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247215[1].headnode03.c  mjm4y       short    TestArray-1       17732     1      2    --   01:30:00 R       --
247215[2].headnode03.c  mjm4y       short    TestArray-2        1271     1      2    --   01:30:00 R       --
247215[3].headnode03.c  mjm4y       short    TestArray-3          --     1      2    --   01:30:00 H       --
247215[4].headnode03.c  mjm4y       short    TestArray-4          --     1      2    --   01:30:00 H       --
```

Now if we let qdel know which task to delete then it will only delete that task - qdel 247215[2]

```
-bash-3.2$ qdel 247215[2]

-bash-3.2$ qstat -at -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK    Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247215[1].headnode03.c  mjm4y       short    TestArray-1       17732     1      2    --   01:30:00 R       --
247215[2].headnode03.c  mjm4y       short    TestArray-2        1271     1      2    --   01:30:00 C  00:00:00
247215[3].headnode03.c  mjm4y       short    TestArray-3        1367     1      2    --   01:30:00 R       --
247215[4].headnode03.c  mjm4y       short    TestArray-4          --     1      2    --   01:30:00 H       --
```

Notice that task two has finished and task three has started.

Finally you can use the –t option to list a selection of tasks to delete. Let us start with an array of 12 tasks with 4 running

```
-bash-3.2$ qsub arrayjob4.sh
247230[].headnode03.cent.gla.ac.uk
```

After some time – note the first two tasks have competed – the next four are running and 6 are held.

```
-bash-3.2$ qstat -at -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK    Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247230[1].headnode03.c  mjm4y       short    TestArray-1       23011     1      2    --   01:30:00 C  00:00:00
247230[2].headnode03.c  mjm4y       short    TestArray-2        6661     1      2    --   01:30:00 C  00:00:00
247230[3].headnode03.c  mjm4y       short    TestArray-3       20692     1      2    --   01:30:00 R  00:00:00
247230[4].headnode03.c  mjm4y       short    TestArray-4       31132     1      2    --   01:30:00 R  00:00:00
247230[5].headnode03.c  mjm4y       short    TestArray-5       23105     1      2    --   01:30:00 R  00:00:00
247230[6].headnode03.c  mjm4y       short    TestArray-6        6755     1      2    --   01:30:00 R  00:00:00
247230[7].headnode03.c  mjm4y       short    TestArray-7          --     1      2    --   01:30:00 H       --
247230[8].headnode03.c  mjm4y       short    TestArray-8          --     1      2    --   01:30:00 H       --
247230[9].headnode03.c  mjm4y       short    TestArray-9          --     1      2    --   01:30:00 H       --
247230[10].headnode03.  mjm4y       short    TestArray-10         --     1      2    --   01:30:00 H       --
247230[11].headnode03.  mjm4y       short    TestArray-11         --     1      2    --   01:30:00 H       --
247230[12].headnode03.  mjm4y       short    TestArray-12         --     1      2    --   01:30:00 H       --
```

So if we decide that we want to kill a specific selection of the tasks we can do it like this

```
-bash-3.2$ qdel -t 4,5,10-11 247230[]

-bash-3.2$ qstat -at -u mjm4y

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK    Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
247230[1].headnode03.c  mjm4y       short    TestArray-1       23011     1      2    --   01:30:00 C  00:00:00
247230[2].headnode03.c  mjm4y       short    TestArray-2        6661     1      2    --   01:30:00 C  00:00:00
247230[3].headnode03.c  mjm4y       short    TestArray-3       20692     1      2    --   01:30:00 R  00:00:00
247230[4].headnode03.c  mjm4y       short    TestArray-4       31132     1      2    --   01:30:00 C  00:00:00
247230[5].headnode03.c  mjm4y       short    TestArray-5       23105     1      2    --   01:30:00 C  00:00:00
247230[6].headnode03.c  mjm4y       short    TestArray-6        6755     1      2    --   01:30:00 R  00:00:00
247230[7].headnode03.c  mjm4y       short    TestArray-7       23204     1      2    --   01:30:00 R       --
247230[8].headnode03.c  mjm4y       short    TestArray-8       31231     1      2    --   01:30:00 R       --
247230[9].headnode03.c  mjm4y       short    TestArray-9          --     1      2    --   01:30:00 H       --
247230[10].headnode03.  mjm4y       short    TestArray-10         --     1      2    --   01:30:00 C       --
247230[11].headnode03.  mjm4y       short    TestArray-11         --     1      2    --   01:30:00 C       --
247230[12].headnode03.  mjm4y       short    TestArray-12         --     1      2    --   01:30:00 H       --
```

Note tasks 4,5,10 and 11 have completed but the others are still running or waiting to run.

## qsub – advanced part two

There is another qsub option you may find useful –W which is for additional attributes. You can look at the qsub manual pages for a full list of attributes but I am going to cover three here:- depend, stagein & stageout

**depend**

depend allows you to automate some of the decision you might take about the order that you run your jobs or what to do after a job or group of jobs has finished. It lets you decide to run jobs before, after or along with other jobs.

qsub –W depend=dependency_list

Where the dependency list can be one or more of a list of dependencies on other jobs. There are 5 groups of dependencies:-

you want your job to run after another job or jobs;
before another job or jobs;
at the same time as another job or jobs;
after a job array;
before a job array

Here we submit a job after another job:-

qsub –W depend=after:12345   script1.sh

qsub –W depend=afterok:12345   script2.sh

qsub –W depend=afternotok:12345   script3.sh

qsub –W depend=afterany:12345   script4.sh

These mean submit a job and give it a dependency on job 12345, in the first case the new job can start after job 12345 has started execution. In the second case the job can start only after the successful completion of job 12345. In the third case the job starts only if job 12345 fails. In the final case the job will start after job 12345 finishes, with or without errors.

Here is an example in practice:-

Submit four jobs each writes to a different logfile – I also submitted a final job that combines the four logfiles. I put a dependency on that job to only start after the successful completion of the other four jobs

Contents of cleanup.sh

```
#! /bin/bash
cat /export/home/mjm4y/depend-job-test/*log  >> /export/home/mjm4y/depend-job-test/combined.log
```

```
-bash-3.2$ qsub one.sh
246482.headnode03.cent.gla.ac.uk

-bash-3.2$ qsub two.sh
246483.headnode03.cent.gla.ac.uk

-bash-3.2$ qsub three.sh
246484.headnode03.cent.gla.ac.uk

-bash-3.2$ qsub four.sh
246485.headnode03.cent.gla.ac.uk

-bash-3.2$ qsub -W depend=afterok:246482:246483:246484:246485 cleanup.sh
246486.headnode03.cent.gla.ac.uk
```

```
-bash-3.2$ qstat -a

headnode03.cent.gla.ac.uk:
                                                                         Req'd    Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory   Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ ------ --------- - ---------
246482.headnode03.cent  mjm4y       short    one.sh             1365     1      1    --   01:00:00 R       --
246483.headnode03.cent  mjm4y       short    two.sh             9110     1      1    --   01:00:00 R       --
246484.headnode03.cent  mjm4y       short    three.sh           6659     1      1    --   01:00:00 R       --
246485.headnode03.cent  mjm4y       short    four.sh           24051     1      1    --   01:00:00 R       --
246486.headnode03.cent  mjm4y       short    cleanup.sh          --      1      1    --   01:00:00 H       --
```

Notice the cleanup job is in the Held state – once the other four jobs successfully complete the cleanup job will run.

You can also submit a job – but require subsequent jobs to run before it starts. When you submit it you need to let Torque know there will be a dependency so you would submit it like this:-

qsub –W depend=on:2 cleanup.sh

This says that there is a dependency on 2 subsequent jobs that need to run before this job starts.

You then submit the subsequent jobs as follows (assume the job above is jobid 12345)

qsub –W depend=before:12345 one.sh

qsub –W depend=beforeok:12345 one.sh

qsub –W depend=beforenotok:12345 one.sh

qsub –W depend=beforeany:12345 one.sh

The options are the same as for the after dependencies, however in this case 2 jobs much reach their dependency before the original job starts.

To run a group of jobs together you need to start the first with the synccount dependency then start the others with the syncwith dependency

The fist job is started as follows

```
qsub –W depend=synccount:2 script1.sh
```

This requires another two jobs to be grouped with this one and then the three jobs will start at the same time

```
qsub –W depend=syncwith:12345  script2.sh
qsub –W depend=syncwith:12345  script3.sh
```

The final two groups are about dependencies on arrays. The first is a group that start after an array has started. Note [count] this is an optional attribute particular to arrays and I will discuss this later.

```
qsub –W depend=afterstartarray:12345[][count]  script.sh
```

```
qsub –W depend=afterokarray:12345[][count]  script.sh
```

```
qsub –W depend=afternotokarray:12345[][count]  script.sh
```

```
qsub –W depend=afteranyarray:12345[][count]  script.sh
```

Let us look at this in practice.  If we were to submit our arrayjob.sh as before in “qsub advanced part one”

Then if we wanted to run our cleanup.sh job again we can run it with an array dependency as follows.

```
qsub -W depend=afterokarray:12345[] cleanup.sh
```

This means that cleanup.sh will start after all tasks of array 12345 have successfully completed.  You can use the other three variants as before to start the job when the all tasks in the array have started, not completed successfully, or just all completed – with or without errors.

Now we will discuss the [count] option – this allows you to run a job after a set number of the tasks of the job array have satisfied the condition. So if we wanted cleanup.sh to run after at least two of the array job tasks have finished we would submit it like this:-

```
 qsub -W depend=afterokarray:12345[][2] cleanup.sh
```

Finally we should be able to submit an array job that requires other jobs to run before it starts   with beforestartarray,beforeokarray etc. However I could not get it to work, I tried many different combinations and no of them worked – I can get it to accept the jobs but the job that depends on the later job never starts and remains in the Hold state.

**stagein**
**stageout**

I am going to look at these two related options together. These options are for copying a file to the execution node before your job starts (stagein) and copying a file back after your job completes (stageout). They can be used independently as well as together.

```
qsub –W stagein=localfile:remotefile
qsub –W stageout=localfile:remotefile
```

Local in this case means on the execution node.

Why might you want to do this when you have a shared filestore?

If you have a job that is greatly IO intensive (reads/writes to a file a lot) then you will get MUCH better performance with a local file rather than a remotely accessed (NFS) file.

The benefit must outweigh the overhead of copying the file(s).

Let us look at adding them to a batch script – The filesystem /tmp on each node is 95GB – this is shared with all users currently running jobs on that execution node. Note the use of localhost here, this is a way of specifying the node your job is actually running on.

```
#!/bin/bash

#PBS -l cput=01:30:00, walltime=02:00:00, nodes=1:ppn=2:centos7
#PBS –m abe
#PBS –M mjm4y@udcf.gla.ac.uk
#PBS –N TestFileStaging
#PBS –W stagein=/tmp/myinputfile@localhost:/path/to/input
#PBS –W stageout=/tmp/myoutputfile@localhost:/path/to/outputfile

/path/to/programme_to_run   /tmp/myinputfile /tmp/myoutputfile
```

Warning:-  Avoid generic names for your input and output files – like out.log or input. If you do this you run the risk of choosing the same name as another user who may be running jobs on the same execution node. I would suggest using your username somewhere in the file – eg mjm4y-input.file or mjm4y-output.log.

When your job ends the stagein & stageout files are deleted from /tmp.

# MPI

MPI is using multiple processors across multiple nodes, MPI stands for Message Passing Interface. This is less of an issue now that we have nodes with 64 cores and 512GB RAM – most jobs will be able to run on the one physical node and therefore not need to consider message passing between nodes.

There is an important environment variable $PBS_NODEFILE which is a file containing the nodes that your job has been assigned. You use this to pass the list of nodes to the mpi programme.

I will explain the use of pbsdsh and mpirun.

pbsdsh is supplied as part of torque and so already knows what nodes your job has assigned. You can use this to launch multiple copies of your programme. Eg pbsdsh script.sh will launch a copy of script.sh on each processor you have been assigned, gathering the output and error files

Example 1

qsub parallel-simple.sh

parallel-simple.sh contains

```
#!/bin/sh
#PBS -l cput=55:30:00, walltime=100:00:00, nodes=2:ppn=2:centos7
#PBS –m abe
#PBS –M mjm4y@udcf.gla.ac.uk

/usr/local/bin/pbsdsh   script.sh &
```

This will launch 4 copies of script.sh – 2 copies on each of the two nodes you have been assigned
It is upto you to write the code if you require them to communicate between the processes.

A more complicated example uses mpirun is shown next

Example 2

qsub parallel-complex.sh

parallel-complex.sh contains

```
#!/bin/sh
#PBS -l cput=55:30:00, walltime=100:00:00, nodes=2:ppn=16:centos7
#PBS –m abe
#PBS –M mjm4y@udcf.gla.ac.uk

module load openmpi
mpirun -machinefile $PBS_NODEFILE -np 32 $HOME/thread.sh &
```

We use mpirun to control the process between nodes. The environment variable $PBS_NODEFILE contains a file with the list of nodes assigned to your job.
This job will then run on 16 processors on each of 2 nodes – the mpirun command will handle message passing between the nodes.

If you are using an mpi aware programme you should find that it can communicate between the processes on different nodes.

If you want to handle your own message passing then you need to be aware of the $PBS_NODEFILE variable – this variable contains a list of the nodes you have been allocated – a particular node will appear twice if you have been allocated both slots on a node. To take a look at this try the following – qsub –I –l nodes=2:ppn=2. then in the shell you are presented with type cat $PBS_NODEFILE – the output is 4 lines, each line is the hostname of a node and each node is repeated twice (because you have asked for 2 processors on each node).

There is a temporary folder created on each node that you are using at /tmp/$PBS_JOBID they are deleted when your job ends.

For more detailed instructions please check the MPI basic instructions manual, or contact us if in doubt.

## GPU Servers

The HPC has a couple of GPU servers that have been provided by The College of Science & Engineering primarily for their researchers.

For those researchers with access the job needs to be submitted directly to the Queue rather than to the generic routing queue as you would with normal jobs we do this with another qsub option -q. Normally it is better to allow Torque/Maui to decide which queue is best for you according to the resources you request – however routing to the gpu queue based on resource requests is not currently working as expected as the full gpu options were not compiled into torque when the HPC was put together. The following sections show how to submit jobs to run on the GPU servers. You should use the ppn statement here as a request for the number of GPU's you need on the node – eg nodes=1:ppn=1 for one GPU or nodes=1:ppn=2 for two GPU's.

### Qsub options

-q

This allows you to submit a job directly to a particular queue. In this case we use it to submit directly to the gpu queue. You follow the –q with the name of the queue you want your job to run on.

```
        -q queue_name
```

Here are examples at the command line

```
qsub –q gpu -l nodes=1:ppn=1,walltime=24:00:00 /path/to/programme_to_run
```

or in a wrapper script

```
#!/bin/bash
#PBS -l nodes=1:ppn=1,walltime=24:00:00
#PBS -q gpu

/path/to/programme_to_run
```

### Other considerations

The GPU servers have multiple GPU's in them – you have to ensure that your job only uses the gpu you have been assigned and not any of the others. These may well have other jobs running on them and so if you ran your job on the same GPU you would not get to use the full potential of the GPU and would possibly adversely affect the other researcher's job. Luckily there is an environment variable you can use to control this called CUDA_VISIBLE_DEVICES – this is a comma separated list of the indices of the GPU's you can use eg 2,0 – in your script you need to set this with indices of the GPU((s) you have been allocated. When your job starts on one of the GPU servers Torque creates a file with this information and you need to use this to set the environment variable. You just need to copy this information to the CUDA_VISIBLE_DEVICES environment variable.

In the bash shell you would do this with the following line in your script

```
export CUDA_VISIBLE_DEVICES=`cat /tmp/$PBS_JOBID/gpu`
```

You can put this in either your wrapper script or at the start of your own script – here is the earlier example expanded to include this.

```
#!/bin/bash
#PBS -l nodes=1:ppn=1,walltime=24:00:00
#PBS -q gpu

export CUDA_VISIBLE_DEVICES=`cat /tmp/$PBS_JOBID/gpu`

/path/to/programme_to_run
```

# Gotcha’s & what to do about them

**You check a program on the headnode & it works – you submit it to the cluster & it fails**

The headnode has many more packages on it than the compute nodes, I would advise checking a program by running qsub –I and running the program within the shell you are given.

**You do not get the error & output files written to your filespace**

Check that you launched qsub from a directory you have write permission to. Check that you have properly done the ssh setup steps.

**You do not get the mail messages sent to you when jobs begin and end**

Have you passed the –m option? Do you have mail forwarding setup or pass the –M option to qsub

**Your MPI job fails, unable to talk to other nodes**

Some mpi aware code communicates using rsh – the HPC cluster only uses ssh. You will probably have to pass an environment variable to your programme of the sort –

```
export  RSH_COMMAND=/usr/bin/ssh
```

for castep the command is

```
export P4_RSHCOMMAND=/usr/bin/ssh
```

some other programs assume one mpi implementation or another for instance cct & par assume that you are using lamd but you can pass an option at runtime – no-lamd which turns off this assumption.

**Path problems**

I would recommend using absolute paths at all time to avoid working directory problems – it appears that the working directory in a script does not always change when you issue a cd command.

**Memory problems**

You can sometimes get memory problems if your job needs a lot of memory. Our compute nodes currently have between 4GB of RAM (on some 4 processor nodes) upto 512GB of RAM on our 64 processor nodes – There is a property on each of the nodes that described the amount of RAM available per processing core, it is one of:-

twogpc – 2GB per processing core
eightgpc – 8GB per processing core
sixteengpc – 16GB per processing core

You can use this in the –l options when you request resources for your job (see the qsub notes for details of this)

**Array Job problems - (Bad Job Array Request)**

The range of valid array id numbers is 0 to 99999. You have to submit your array with id's within this range.

**Array Job problems - (Bad Job Array Request MSG=Requested array size too large, limit is 5000)**

For performance and reliability reasons there is a maximum size of array allowed on the cluster at the time of writing this is 5000, although this may well change – the current number is included in the error message.  The solution is to submit your job in batches of that size or smaller – you can use discrete ranges as long as they are within the 0-99999 range

i.e.

```
qsub –t 0-4999 script.sh
qsub –t 5000-9999 script.sh
qsub –t 10000-14999 script.sh etc
```

You should write your submission script in such a way as to only submit a batch as the earlier one finishes. (e.g package Raccoon 2 handles much of this, although it will try to submit a job with array ids greater than 99999).

**R – Installing New Packages**

If you try to install new packages in R it will normally try to connect to the internet to download it. The cluster nodes are on private address space and so cannot reliably connect out with campus.

Instructions on compiling R packages from source are available under the Manuals section on R-Project website, contact us if in doubt.