Submitting a job with qsub
Batch vs interactive

The working pattern we are all familiar with is interactive - I type (or click) something, and the computer performs the associated action. Then I type (or click) the next thing.
You may recall this from the first tutorial:

The trouble with interactive environments

There is another reason why GUIs are less common in HPC environments: **point-and-click is necessarily interactive.** In HPC environments (as we'll see in session 3) work is scheduled in order to allow exclusive use of the shared resources. On a busy system there may be several hours wait between when you submit a job and when the resources become available, so a reliance on user interaction is not viable. In Unix, commands need not be run interactively at the prompt, you can write a sequence of commands into a file to be run as a script, either manually (for sequences you find yourself repeating frequently) or by another program such as the batch system.

The job might not start immediately, and might take hours or days, so we prefer a **batch** approach:

- plan the sequence of commands which will perform the actions we need
- write them into a script

I can now run the script interactively, which is a great way to save effort if I frequently use the same workflow, or ...

- submit the script to a batch system, to run on dedicated resources when they become available

Where does the output go?

- The batch system writes stdout and stderr from a job to files named "jobname.o12345" and "jobname.e12345"
- Which you can change, using qsub options
- While a job is running, it caches the stdout and stderr in a hidden directory: $HOME/.pbs_spool
- You can use redirection (See Tutorial 1) to send output of a specific command into a file

Writing and Submitting a job

There are two aspects to a batch job script:

- A set of PBS directives describing the resources required and other information about the job for Torque
- The script itself, comprised of commands to setup and perform the computations without additional user interaction

A simple example

A typical batch script on an NYU HPC cluster looks something like these:

<table>
<thead>
<tr>
<th>Using precompiled third-party software</th>
<th>Using self-developed or built software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
We'll work through them more closely in a moment.

You submit the job with qsub:

```
$ qsub myscript.q
```

And monitor its progress (as is discussed further in here) with:

```
$ qstat -u $USER
```

What just happened? Here's an annotated version of the first script:

```bash
#!/bin/bash
# This line tells the shell how to execute this script, and is unrelated
to PBS. In fact, by default PBS executes scripts with bash so this line
could be omitted

# at the beginning of the script, lines beginning with "#PBS" are read by
# Torque and used to set queueing options. You can comment out a PBS
# directive with a second leading #, eg:
#PBS -l nodes=2:ppn=4

# we need 1 node, with 1 process per node:
#PBS -l nodes=1:ppn=1

# we expect the job to finish within 5 hours. If it takes longer than 5
# hours, Torque can kill it:
#PBS -l walltime=5:00:00

# we expect the job to use no more than 2GB of memory:
#PBS -l mem=2GB

# we want the job to be named "jobname" rather than something generated
```
# from the script name. This will affect the name of the files where
# stdout and stderr are placed, and also the name of the job as reported
# by qstat:
#PBS -N jobname

# if the job fails, send me an email at this email address.
#PBS -M bob.smith@nyu.edu

# instead of separate files for stdout and stderr, merge both into the
# stdout file. It will be placed in the directory I submitted the job
# from and will have a name like jobname.o12345
#PBS -j oe

# once the first non-comment, non-PBS-directive line is encountered, Torque
# stops looking for PBS directives. The remainder of the script is executed
# as a normal Unix shell script

# first we ensure a clean running environment:
module purge
# and load the module for the software we are using:
module load stata/13

# next we create a unique directory to run this job in. We will record its
# name in the shell variable "RUNDIR", for better readability.
# Torque sets PBS_JOBID to the job id, something like 12345.crunch.local
# ${PBS_JOBID/.*} expands to the job id up to the first '.
# We take the job id (the '12345' in 'jobname.o12345')
# We make the run directory in our area under $SCRATCH, because at NYU HPC
# $SCRATCH is configured for the disk space and speed required by HPC jobs.
RUNDIR=$SCRATCH/my_project/run-${PBS_JOBID/.*}
mkdir $RUNDIR

# we will be reading data in from somewhere, so define that too:
DATADIR=$SCRATCH/my_project/data

# the script will have started running in $HOME, so we need to move into the
# unique directory we just created
cd $RUNDIR

# now start the Stata job:
stata -b do $DATADIR/data_0706.do
The second script has the same PBS directives, but this time we are using code we compiled ourselves. Starting after the PBS directives:

```bash
# first we ensure a clean running environment:
module purge

# and ensure we can find the executable:
SRCDIR=$HOME/my_project/code

# create a unique directory to run this job in, as per the script above
RUNDIR=$SCRATCH/my_project/run-${PBS_JOBID/.*/}
mkdir $RUNDIR

# the script will have started running in $HOME, but our job script (and in
# this example our input file) is in the directory we ran qsub from. Torque
# sets some environment variables with information about the job, including
# PBS_O_WORKDIR which is the directory the job was submitted from. So lets
# go there and copy the input file to the run directory on /scratch:
cd $PBS_O_WORKDIR
cp my_input_params.inp $RUNDIR

# go to the run directory to begin the run:
cd $RUNDIR

# load whatever environment modules the executable needs:
module load fftw/intel/3.3

# run the executable (sending the contents of my_input_params.inp to stdin)
$SRCDIR/my_exec.exe < my_input_params.inp

# blank line to finish
```

Submitting a job

Jobs are submitted with the `qsub` command:

```
$ qsub options job-script
```

The options tell Torque information about the job, such as what resources will be needed. These can be specified in the job-script as PBS directives, or on the command line as options, or both (in which case the command line options take precedence should the two contradict each other). For each option there is a corresponding PBS directive with the syntax:

```
#PBS option
```

For example, you can specify that a job needs 2 nodes and 8 cores on each node by adding to the script the directive:

```
#!/bin/bash
#PBS -l nodes=2:ppn=8
```

or as a command-line option to `qsub` when you submit the job:
$ qsub -l nodes=2:ppn=8 my_script.q

Options to manage job output:

- **-N jobname**
  Give the job a name. The default is the filename of the job script. Within the job, $PBS_JOBNAME expands to the job name.
- **-j oe**
  Merge stderr into the stdout file.
- **-o path/for/stdout**
  Send stdout to path/for/stdout. Can be a filename or an existing directory. The default filename is $PBS_JOBNAME.o$(PBS_JOBID)/*, e.g., myjob.o12345, in the directory from which the job was submitted.
- **-e path/for/stderr**
  Send stderr to path/for/stderr. Same usage as for stdout.
- **-M my_email_address@nyu.edu**
  Send email to my_email_address@nyu.edu when certain events occur. By default an email is sent only if the job is killed by the batch system.
- **-m b -m e -m a -m abe**
  Send email when the job begins (b), ends (e) and/or is aborted (a).

Options to set the job environment:

- **-S /path/to/shell**
  Use the shell at /path/to/shell to interpret the script. Default is your login shell, which at NYU HPC is normally /bin/bash.
- **-v , = , VAR1 = , VAR2 = , "some value" , VAR3**
  Pass variables to the job, either with a specific value (the VAR=value form) or from the submitting environment (without "=").
- **-V**
  Pass the full environment the job was submitted from.

Options to request compute resources:

- **-l walltime=walltime**
  Maximum wallclock time the job will need. Default is 1 hour. Walltime is specified in seconds or as hh:mm:ss.
- **-l mem=memory**
  Maximum memory per node the job will need. Default depends on queue, normally 2GB for serial jobs and the full node for parallel jobs. Memory should be specified with units, e.g., 500MB or 8GB.
- **-l nodes=num:ppn=num**
  Number of nodes and number of processors per node required. Default is 1 node and 1 processor per node. The :ppn=num can be omitted, in which case (at NYU HPC) you will get full nodes. When using multiple nodes the job script will be executed on the first allocated node.
- **-q queue**
  Submit to a specific queue. If not specified, Torque will choose a queue based on the resources requested.

A job submitted without requesting a specific queue or resources will go to the default serial queue (s48 on Mercer) with the default resource limits for that queue.

Requesting the resources you need, as accurately as possible, allows your job to be started at the earliest opportunity as well as helping the system to schedule work efficiently to everyone’s benefit.

Options for running interactively on the compute nodes:

- **-I**
  Don’t just submit the job, but also wait for it to start and connect stdout, stderr and stdin to the current terminal.
- **-X**
  Enable X forwarding, so programs using a GUI can be used during the session (provided you have X forwarding to your workstation set up).
- **-V**
  Pass the current environment to the interactive batch job.
- **exit**
  To leave an interactive batch session, type exit at the command prompt.

Options for delaying starting a job:

- **-W depend=afterok:jobid**
  Delay starting this job until jobid has completed successfully.
- **-a [MM/DD]hhmm**
  Delay starting this job until after the specified date and time. Month (MM) and day-of-month (DD) are optional, hour and minute are required.
Options for many similar jobs (array jobs and pbsdsh):

- `-t 1,10,50-100`
  Submit an array of jobs with array ids as specified. Array ids can be specified as a numerical range, a comma-separated list of numbers, or as some combination of the two. Each job instance will have an environment variable $PBS_ARRAYID
- `-t 1,10,50-100%5`
  As above, but the appended ’%5’ specifies the maximum number of array items (in this case, 5) which should be running at one time
- Submit a single "shepherd" job requesting multiple processes and from it start individual jobs with `pbsdsh`.

Next: Requesting resources