Starting and managing jobs with PBS

This documentation provides information for **how to use PBS Pro** to submit and manage interactive jobs and batch jobs on NCAR systems.

The basic PBS commands are the same on each cluster, but refer to these system-specific pages for details that are unique to each of them, including hardware specifications, software, and job-submission queues and procedures:

- Derecho (in development)
- Casper - casper.ucar.edu
- Cheyenne - cheyenne.ucar.edu

Page contents

Submitting jobs

PBS Pro is used to schedule both interactive jobs and batch compute jobs. Detailed examples of how to start both types of jobs are included in the documentation (see links above) for each individual system.

Commands for starting interactive jobs are specific to individual systems. The basic command for starting a batch job, however, is the same.

To submit a batch job, use the **qsub** command followed by the name of your PBS batch script file.

```
qsub script_name
```

Propagating environment settings

Some users find it useful to set environment variables in their login environment that can be temporarily used for multiple batch jobs without modifying the job script. This practice can be particularly useful during iterative development and debugging work.

PBS has two approaches to propagation:

1. Specific variables can be forwarded to the job upon request.
2. The entire environment can be forwarded to the job.

In general, the first approach is preferred because the second may have unintended consequences.

These settings are controlled by **qsub** arguments that can be used at the command line or as directives within job scripts. Here are examples of both approaches:

```
# Selectively forward runtime variables to the job (lower-case v)
qsub -v DEBUG=true,CASE_NAME job.pbs
```

When you use the selective option (lower-case v), you can either specify only the variable name to propagate the current value (as in CASE_NAME in the example), or you can explicitly set it to a given value at submission time (as in DEBUG).

```
# Forward the entire environment to the job (upper-case V)
qsub -V job.pbs
```

Do not use full propagation when peer-scheduling jobs. Doing so will cause libraries and binaries to be inherited via variables like PATH and LD_LIBRARY_PATH. These inherited settings WILL cause applications to break, and may render the job completely unusable.

Managing jobs

Here are some of the most useful commands for managing and monitoring jobs that have been launched with PBS.

Most of these commands will only modify or query data from jobs that are active on the same system. That is, run each command on Derecho if you want to interact with a job on Derecho.

Run any command followed by **-h** to get help, as in **qhist -h**.

qdel

Run **qdel** with the job ID to kill a pending or running job.

```
qdel jobID
```

Kill all of your own pending or running jobs. (Be sure to use *backticks* as shown.)

```
qdel `qselect -u $USER`
```

qhist

Run **qhist** for information on finished jobs.

```
qhist -u $USER
```

Your output will include jobs that finished on the current day unless you specify the number (N) of days to include.

```
qhist -u $USER -d N
```

Your output will be similar to this, with Mem(GB) and CPU(%) indicating approximate total memory usage **per job** and average CPU usage **per core per job**:

```
Job ID  User     Queue  Nodes NCPUs Finish  RMem(GB)  Mem(GB) CPU(%) Elap(h)
2426690 stormyk  regular    1     1 05-1527        -     0.3   75.0    0.09
2426693 stormyk  regular    1     1 05-1527        -     0.1   90.0    0.09
2426541 stormyk  regular    1     1 05-1523        -     0.1   83.0    0.03
2426542 stormyk  regular    1     1 05-1524        -     0.1   70.0    0.04
2426683 stormyk  regular    1     1 05-1523        -     0.1    0.0    0.02
2426444 stormyk  regular    1     1 05-1522        -     0.1   19.0    0.02
2426435 stormyk  regular    1     1 05-1522        -     0.1   13.0    0.02
```

The following variation will generate a list of jobs that finished with non-zero exit codes to help you identify jobs that failed.

```
qhist -u $USER -r x0
```

qstat

Run this to see the status of all of your own unfinished jobs.

```
qstat -u $USER
```

Your output will be similar to what is shown just below. Most column headings are self-explanatory – NDS for nodes, TSK for tasks, and so on.

In the status (S) column, most jobs are either queued (Q) or running (R). Sometimes jobs are held (H), which might mean they are dependent on the completion of another job. If you have a job that is held and is not dependent on another job, CISL recommends killing and resubmitting the job.

```
                                                  Req'd  Req'd   Elap
Job ID          Username Queue   Jobname SessID NDS TSK Memory Time  S Time
------          -------- -----   ------- ------ --- --- ------ ----- - ----
657237.chadmin apatelsm economy ens603   46100 60  216   --   02:30 R 01:24
657238.chadmin apatelsm regular ens605     --   1   36   --   00:05 H   --
657466.chadmin apatelsm economy ens701    5189 60  216   --   02:30 R 00:46
657467.chadmin apatelsm regular ens703     --   1   36   --   00:10 H   --
```

Following are examples of qstat with some other commonly used options and arguments.

Get a long-form summary of the status of an unfinished job. (Use this only sparingly; it places a high load on PBS.)

```
qstat -f jobID
```

Get a single-line summary of the status of an unfinished or recently completed job (within 72 hours).

```
qstat -x jobID
```

Get information about unfinished jobs in a specified execution queue.

```
qstat queue_name
```

See job activity by queue (e.g., pending, running) in terms of numbers of jobs.

```
qstat -Q
```

Display information for all of your pending, running, and finished jobs.

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

Query jobs running on one system by specifying the system as shown here. (Only these options are supported when running **qstat** in this cross-server mode: **-x**, **-u**, **-w**, **-n**, **-s**)

```
qstat -w -u $USER @derecho
```