Casper job script examples

These PBS batch script examples work for executables generated by any compiler and MPI library installed on Casper. (The defaults are Intel and OpenMPI, respectively.) Remember to substitute your own job name and project code, and customize the other directives and commands as necessary. That includes the commands shown for setting your **TMPDIR** environment variable as recommended here: Storing temporary files with TMPDIR.

The examples are similar to PBS examples for running jobs on Cheyenne. For help with any of them, contact the NCAR Research Computing help desk.

When your script is ready, submit your batch job from a Casper login node by using the **qsub** command followed by the name of your script file.

```
qsub script_name
```

You can also submit your batch script from a Cheyenne login node to Casper by using the **qsubcasper** command. All other steps are the same.

```
qsubcasper script_name
```

On Cheyenne, we recommend the use of the **-k oed** flag to enable direct writing of job output and error logs to the submission directory of the job (instead of copying them to this directory at the end of the job). In the Casper version of PBS, we have made log direct-write the default behavior, and so this flag is no longer needed. If you move a workflow between Cheyenne and Casper, it is appropriate to keep that flag in your script for consistent behavior.

Page contents

Batch script to run an MPI GPU job

For bash users

```
#!/bin/bash -l
#PBS -N mpi_job
#PBS -A project_code
#PBS -l select=2:ncpus=4:mpiprocs=4:ngpus=4:mem=40GB
#PBS -l gpu_type=v100
#PBS -l walltime=01:00:00
#PBS -q casper
#PBS -j oe

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Provide CUDA runtime libraries
module load cuda

### Run program
mpirun ./executable_name
```

For tcsh users

```
#!/bin/tcsh
### Job Name
#PBS -N mpi_gpu_job
### Charging account
#PBS -A project_code
### Request two resource chunks, each with 4 CPUs, GPUs, MPI ranks, and 40 GB of memory
#PBS -l select=2:ncpus=4:mpiprocs=4:ngpus=4:mem=40GB
### Specify that the GPUs will be V100s
#PBS -l gpu_type=v100
### Allow job to run up to 1 hour
#PBS -l walltime=01:00:00
### Route the job to the casper queue
#PBS -q casper
### Join output and error streams into single file
#PBS -j oe

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Provide CUDA runtime libraries
module load cuda

### Run program
mpirun ./executable_name
```

Batch script to run a pure OpenMP job

For bash users

```
#!/bin/bash -l
#PBS -N OpenMP_job
#PBS -A project_code
#PBS -l select=1:ncpus=8:ompthreads=8
#PBS -l walltime=00:10:00
#PBS -q casper
#PBS -j oe

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run program
./executable_name
```

For tcsh users

```
#!/bin/tcsh
#PBS -N OpenMP_job
#PBS -A project_code
#PBS -l select=1:ncpus=8:ompthreads=8
#PBS -l walltime=00:10:00
#PBS -q casper
#PBS -j oe

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run program
./executable_name
```

Batch script to run a hybrid MPI/OpenMP job

For bash users

```
#!/bin/bash -l
#PBS -N hybrid_job
#PBS -A project_code
#PBS -l select=2:ncpus=8:mpiprocs=2:ompthreads=4
#PBS -l walltime=00:10:00
#PBS -q casper
#PBS -j oe

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run program
mpirun ./executable_name
```

For tcsh users

```
#!/bin/tcsh
#PBS -N hybrid_job
#PBS -A project_code
#PBS -l select=2:ncpus=8:mpiprocs=2:ompthreads=4
#PBS -l walltime=00:10:00
#PBS -q casper
#PBS -j oe

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run program
mpirun ./executable_name
```

Batch script to run a job array

Job arrays are useful for submitting and managing collections of similar jobs – for example, running the same program repeatedly on different input files. PBS can process a job array more efficiently than it can process the same number of individual non-array jobs.

This example uses environment variable **$PBS_ARRAY_INDEX** as an argument in running the jobs. This variable is set by the scheduler in each of your array subjobs, and spans the range of values set in the **#PBS -J** array directive.

For bash users

```
#!/bin/bash -l
#PBS -N job_array
#PBS -A project_code
### Each array subjob will be assigned a single CPU with 4 GB of memory
#PBS -l select=1:ncpus=1:mem=4GB
#PBS -l walltime=00:10:00
#PBS -q casper
### Request 10 subjobs with array indices spanning 2010-2020 (input year)
#PBS -J 2010-2020
#PBS -j oe

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run program
./executable_name data.year-$PBS_ARRAY_INDEX
```

For tcsh users

```
#!/bin/tcsh
#PBS -N job_array
#PBS -A project_code
### Each array subjob will be assigned a single CPU with 4 GB of memory
#PBS -l select=1:ncpus=1:mem=4GB
#PBS -l walltime=01:00:00
#PBS -q casper
### Request 10 subjobs with array indices spanning 2010-2020 (input year)
#PBS -J 2010-2020
#PBS -j oe

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run program
./executable_name data.year-$PBS_ARRAY_INDEX
```

If you need to include a job ID in a subsequent **qsub** command, be sure to use quotation marks to preserve the **[]** brackets, as in this example:

```
qsub -W "depend=afterok:317485[]" postprocess.pbs
```

Batch script for running dependent jobs

It is possible to schedule a job to run based on the status of other jobs by setting one or more job dependencies. The dependency option requires knowledge of the job ids from the original job or jobs, and so the option is typically used when running the qsub command, rather than as a batch script directive.

Follow this example, making sure to use **backticks** as shown.

Let's say you have you have three scripts to submit and run consecutively:

1. **pre.pbs**: a preprocessing job
2. **main.pbs**: a computation job
3. **post.pbs**: a post-processing job

Place the first job on hold. If it starts before the dependent jobs are submitted, the dependent jobs might never start.

bash users

```
JID1=`qsub -h pre.pbs`
```

tcsh users

```
set JID1=`qsub -h pre.pbs`
```

Make starting of the second job dependent on successful completion of the first.

bash users

```
JID2=`qsub -W depend=afterok:$JID1 main.pbs`
```

tcsh users

```
set JID2=`qsub -W depend=afterok:$JID1 main.pbs`
```

Make starting of the post-processing job dependent on successful completion of the second job.

```
qsub -W depend=afterok:$JID2 post.pbs
```

Release the first job to initiate the sequence.

```
qrls $JID1
```

The **JID** (job ID) in the second command is the one associated with the first job. In this case, the dependent job will start only if the first job completes successfully (with a zero exit code).

The following shows another way to run dependent jobs. The batch script in this example submits a dependent job from within the original job if the executable runs successfully.

For bash users

```
#!/bin/bash -l
#PBS -N dep_job1
#PBS -A project_code
#PBS -l select=1:ncpus=4
#PBS -l walltime=00:10:00
#PBS -q casper
#PBS -j oe

export TMPDIR=/glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run first program
./executable_name

### Run job 2 if program runs successfully
if [[ $? == 0 ]]; then
   qsub dep_job2
fi
```

For tcsh users

```
#!/bin/tcsh
#PBS -N dep_job1
#PBS -A project_code
#PBS -l select=1:ncpus=4
#PBS -l walltime=00:10:00
#PBS -q casper
#PBS -j oe

setenv TMPDIR /glade/scratch/$USER/temp
mkdir -p $TMPDIR

### Run first program
./executable_name

### Run job 2 if program runs successfully
if ($status .eq. 0) then
   qsub dep_job2
endif
```

This approach could allow you to set more complex conditions based on specific exit codes, whereas the PBS dependency system described above only allows for pass/fail distinctions. However, native PBS dependencies enable checks across multiple jobs, which would be difficult using the second approach.