


# UG3.4.x: Batch Scheduler PBS

(updated Jan 2018)

**Attention**  
**PBS is no more available on HPC clusters in Cineca, since Jan 2018**

In this page:

- [Running applications using PBS](#)
  - [PBS commands](#)
  - [The User Environment](#)
- [PBS Resources](#)
  - [PBS job script](#)
  - [qsub attributes](#)
- [Examples](#)
- [Chaining multiple jobs](#)
- [High throughput Computing with PBS](#)
- [Further documentation](#)

**Portable Batch System** (or simply PBS) is the name of a computer software that performs job scheduling. Its primary task is to allocate computational tasks, i.e., batch jobs, among the available computing resources. At present, PBS is the scheduling system of [GALILEO](#), [PICO](#) and [MARCONI](#). The pdf version of the "[PBSpro 13 user manual](#)" is on this portal.

## Running applications using PBS

With PBS you specify the tasks to be executed; the system takes care of running these tasks and returns the results to you. If the available computers are full, then PBS holds your work and runs it when the resources are available.

With PBS you **create a batch job** which you then **submit to PBS**. A batch job is a file (a shell script under UNIX) containing the set of commands you want to run. It also contains directives which specify the characteristics (attributes) of the job, and resource requirements (e.g. number of processors and CPU time) that your job needs.

Once you create your PBS job, you can reuse it if you wish. Or, you can modify it for subsequent runs.

For example, here is a simple PBS batch job to run a user's application by setting a limit (one hour) to the maximum wall clock time, requesting 1 node with 1 cpus:

```
#!/bin/bash
#PBS -A <account_no>                (only for account based usernames)
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=1
#
./my_application
```

PBS provides two user interfaces: a command line interface (CLI) and a graphical user interface (GUI). The CLI lets you type commands at the system prompt. Only the CLI is presented here, if you are interested in the GUI, please refer to the official documents.

PBS has been configured differently on the various systems reflecting the different system features. Please refer to the specific system page for more detailed information.

## PBS commands

The main user's commands of PBS are reported in the table below: please consult the man pages for more information.

qsub	Submit a job
qstat	Status job, queue, Server
qdel	Delete job

Submit a job:

```
> qsub [opts] job_script
> qsub -I [opts] -- /bin/bash (interactive job)
```

The second command is related to a so-called **"Interactive job"**: sets job's interactive attribute to TRUE. The job is queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected to the terminal session in which qsub is running. When the job begins execution, all input to the job is taken from the terminal session. Use CTRL-D to close the session.

Displaying Job Status:

```
> qstat                               (lists all jobs)
> qstat -u $USER (lists only jobs submitted by you)

> qstat <job_id> (only the specified job)

> qstat -f <job_id>          (full display of the specified job)
```

Displaying Queue Status:

```
> qstat -Q
> qstat -Qf <queuenam>      (Long format of the specified queue)
```

Delete a job:

```
> qdel <jobID>.io01          (Galileo)
> qdel <jobID>.node001       (Pico)

> qdel <jobID>.r000u17101    (Marconi)
> qdel <jobID>               (all clusters)
```

More information about these commands are available with the **man** command.

## The User Environment

There are a number of environment variables provided to the PBS job. Some of them are taken from the user's environment and carried with the job. Others are created by PBS.

All PBS-provided environment variable names start with the characters **PBS\_**. Some are then followed by a capital O (**PBS\_O\_**) indicating that the variable is from the job's originating environment (i.e. the user's).

The following short example lists some of the more useful variables, and typical values (for instance on Galileo):

```
PBS_JOBNAME=jobb
PBS_ENVIRONMENT=PBS_BATCH
PBS_JOBID=453919.io01
PBS_QUEUE=shared

PBS_O_WORKDIR=/gpfs/scratch/usercin/aer0
PBS_O_HOME=/plx/usercin/aer0
PBS_O_QUEUE=route
PBS_O_LOGNAME=aer0
PBS_O_SHELL=/bin/bash
PBS_O_HOST=node166.galileo.cineca.it
PBS_O_MAIL=/var/spool/mail/aer0
PBS_O_PATH=/cinca/bin:/galileo/cineca/sysprod/pbs/default/bin: ...
```

There are a number of ways that you can use these environment variables to make more efficient use of PBS. For example **PBS\_ENVIRONMENT** can be used to test if we were running under PBS. Another commonly used variable is **PBS\_O\_WORKDIR** which contains the name of the directory from which the user submitted the PBS job

(NOTE: **PBS executes the job scripts in the \$HOME directory by default!**)

## PBS Resources

A job requests resources through the PBS syntax; PBS matches requested resources with available resources, according to rules defined by the administrator; when resources are allocated to the job, the job can be executed.

There are different types of resources, i.e. server level resources like walltime, and chunked resources like number of cpus or nodes. Other resources may be added to manage access to software resources for example, particularly when resources are limited and lack of their availability leads to aborting the jobs when they are scheduled for execution. Details may be found in documentation (module help) of the interested applications.

The syntax of the request depends on which type is concerned:

```
#PBS -l <resource>=<value>          (server level resources, e.g. walltime)
#PBS -l select=[N:]chunk[+[N:]chunk ...] (chunk resources, e.g. cpus, gpus, mpiprocs)
```

For example:

```
#PBS -l walltime=10:00
#PBS -l select=1:ncpus=1
```

Moreover, resources can be required in one of two possible ways:

- 1) using PBS directives in the job script
- 2) using options of the **qsub** command

## PBS job script

A PBS job script consists of:

- An optional shell specification
- PBS directives
- Tasks -- programs or commands

Once ready, the job must be submitted to PBS:

```
> qsub [options] <name of script>
```

The **shell** to be used by PBS, if given, is defined in the first line of the job script:

```
#!/bin/bash
```

**PBS directives** are used to request resources or set attributes. A directive begins with the default string **#PBS**. One or more directives can follow the shell definition in the job script.

The **tasks** can be programs or commands. This is where the user specifies an application to be run.

## PBS directives: num. of processors

The number of cpus required for a serial or parallel MPI/OpenMP/mixed job must be required with the "select" directive:

```
#PBS -l select=NN:ncpus=CC:mpiprocs=TT:ngpus=GG:nmics=MM[+N1....]
```

where:

- **NN**: number of nodes (max depending on the queue)
- **ncpus=CC**: number of physical cores *per node*
- **mpiprocs=TT**: number of MPI tasks *per node*
- **ngpus=GG**: number of physical gpus *per node* (if available on the system)
- **nmics=MM**: number of physical Intel MICs *per node* (if available on the system)

for example:

```
#PBS -l select=1:ncpus=1          --> serial job
#PBS -l select=2:ncpus=8:mpiprocs=8 --> MPI job (2 nodes and 8 procs per node)
#PBS -l select=2:ncpus=8:mpiprocs=1 --> mixed job (2 MPI tasks and 8 threads/task)
#PBS -l select=1:ncpus=16:mpiprocs=16:ngpus=2
                                   --> MPI job with cuda (16 MPI tasks and 2 GPUs)
#PBS -l select=2:ncpus=8:mpiprocs=8+1:ncpus=5:mpiprocs=5
                                   --> 21 MPI tasks on three nodes: 8+8+5
```

Please note that **hyper-threading is disabled**, so it is warmly recommended to set a value for mpiprocs less or equal to ncpus. If you specify a higher number of mpiprocs you will overload the physical cores and slow down your own execution.

On different systems there could be different values of cores/node and accelerator devices. Check the system page of the Guide for more information.

## PBS directives: processing time

Resources as the computing time, must be requested in this form:

```
#PBS -l walltime=<value>
```

where **<value>** expresses the actual elapsed time (wall-clock) in the format **hh:mm:ss**

for example:

```
#PBS -l walltime=1:00:00 (one hour)
```

Please note that there are specific limitations on the maximum walltime on a system. Check the system page of the Guide for more information.

## PBS directives: memory allocation

The default memory depends on the system you are working with. You can specify the requested memory with the "mem" directive up to to maximum memory available on the nodes. Even though the maximum amount of allocatable memory is higher, we suggest to limit it to **120 GB on Galileo, 122GB on Pico, 118GB on Marconi-A1, and 86GB (cache mode) / 101GB (flat mode) for Marconi-A2.**

```
#PBS -l select=NN:ncpus=CC:mpiprocs=TT:mem=24GB
```

Please note: if you are requiring a larger memory with respect to the "main amount" on the system, the number of "effective cores" and the cost of your job could increase. For more information refers to the "accounting" section.

## Other PBS directives

```
#PBS -A <account_no> --> name of the project to be accounted to ("saldo -b" for a list of prjs)
#PBS -N name --> job name
#PBS -q destination --> queue destination. For a list and description of available
                        queues, please refer to the specific cluster description of the guide
#PBS -o path --> redirects output file
#PBS -e path --> redirects error file
#PBS -j eo --> merge std-err and std-out
#PBS -m mail_events --> specify email notification (a=aborted,b=begin,e=end,n=no_mail)
#PBS -M user_list --> set email destination (email addr)
```

Please note that Galileo and Marconi are served by a routing queue (directing all jobs to the "shared" queue), which is also the default queue. Hence there is no need to specify the #PBS -q directive. The explicit request of the shared queue (on Galileo) and of the debug, prod, bigprod queues (on Marconi) as the destination will be rejected by PBS. You need to use that directive only if you need to submit jobs to the archive queue.

## qsub attributes

The attributes can also be set using the **qsub** command options:

```
> qsub [-N name] [-q destination] [-o path] [-e path] [-j eo]
      [-m mail_events] [-M user_list] <name of script>
```

The resources can also be requested using the **qsub** command options.

```
> qsub [-l walltime=<value>] [-select=<value>] [-A <account_no>] <name of script>
```

**The qsub command options override script directives.**

-

## Examples

### Serial job script

For a typical serial job you can take the following script as a template, modifying it depending on your needs.

The script asks for 10 minutes wallclock time and runs a serial application (R). The input data are in file "data", the output file is "out.txt"; job.out will contain the std-out and std-err of the script. The working directory is \$CINECA\_SCRATCH/test/.

The account number (#PBS -A) is required to specify the project to be accounted. To find out the list of your account number/s, please use the "**saldo -b**" command.

```
#!/bin/bash
#PBS -o job.out
#PBS -j eo
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1
#PBS -A <my_account>
#
cd $CINECA_SCRATCH/test/
module load R
R < data > out.txt
```

### Serial job script with specific queue request

This script is similar to the previous one but asks for a specific queue. Since "archive" runs on login nodes, a job running here can access given filesystems mounted only on these nodes.

```
#!/bin/bash
#PBS -o job.out
#PBS -j eo
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1
#PBS -A <my_account>
#PBS -q archive
#
cd $CINECA_SCRATCH/test/
cp /gss/gss_work/DRES_my/* .
```

### OpenMP job script

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=8:mpiprocs=1
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>

cd $PBS_O_WORKDIR

module load intel
./myprogram
```

### MPI Job Scripts

For a typical MPI job you can take the following script as a template, modifying it depending on your needs.

The script asks for 20 tasks and 1 hour of wallclock time, and runs a MPI application (myprogram) compiled with the intel compiler and the openmpi library. The input data are in file "myinput", the output file is "myoutput", the working directory is where the job was submitted from.

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=2:ncpus=10:mpiprocs=10      # 2 nodes, 10 procs/node = 20 MPI tasks
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>

cd $PBS_O_WORKDIR # this is the dir where the job was submitted from

module load intel intelmpi
mpirun ./myprogram < myinput > myoutput
```

### MPI+OpenMP job script

The script asks for 2 MPI tasks and 8 OpenMP threads, 1 hours of wallclock time. The application (myprogram) was compiled with the intel compiler and the openmpi library. The input data are in file "myinput", the output file is "myoutput", the working directory is where the job was submitted from.

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=2:ncpus=8:mpiprocs=1
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>

cd $PBS_O_WORKDIR

module load intel intelmpi
mpirun ./myprogram
```

### Serial job script using 1 GPU

For a typical serial job using 1 GPU you can take the following script as a template, modifying it depending on your needs.

The script asks for 30 minutes wallclock time and runs a serial application on the GPU resource.

```
#!/bin/bash
#PBS -l walltime=30:00
#PBS -l select=1:ncpus=1:ngpus=1
#PBS -o job.out
#PBS -e job.err
#PBS -A <my_account>
cd $PBS_O_WORKDIR

./myCUDAprogram
```

## Chaining multiple jobs

-

In some cases, you may want to chain multiple jobs together, in a way that the output of a run can be used as input of the next run. This is typical when you perform Molecular Dynamics Simulations and you want to obtain a long trajectory from multiple simulation runs.

-

In order to exploit this feature you need to submit the job with the following syntax:

-

```

> qsub -W depend=afterok:JOBID.io01 job.sh      (Galileo)
> qsub -W depend=afterok:JOBID.node001 job.sh   (Pico)

> qsub -W depend=afterok:JOBID.r000u17101 job.sh (Marconi A1)
> qsub -W depend=afterok:JOBID.r064u06s01 job.sh (Marconi A2)
> qsub -W depend=afterok:JOBID.r000u26s04 job.sh (Marconi A3)
> qsub -W depend=afterok:JOBID job.sh           (all clusters)

```

where JOBID is the job id (e.g. 204375) of the job you want to concatenate. There are multiple choices for the dependency (afterok, afternotok, afterany), please refer to the PBS manual (PBS Professional [User Guide 13.0](#)) .

## High throughput Computing with PBS

*Array jobs* is an efficient way to perform multiple similar runs, either serial or parallel, by submitting a unique job. The maximum allowed number of runs in an array job depends on the cluster.

In the following examples, 20 serial simulations, that differ only by the input, are submitted through the "job.cmd" script.

```

#!/bin/bash
#PBS -N job_array
#PBS -l select=1:ncpus=1:mpiprocs=1
#PBS -l walltime=24:00:00
#PBS -A <account_name>
#PBS -J 1-20
#PBS -r y

cd $PBS_O_WORKDIR
cd $PBS_ARRAY_INDEX

../exe input$PBS_ARRAY_INDEX.txt

```

20 sub-jobs, each with 1 cpu and 8 GB of memory, are generated in according to the total number requested through the **#PBS -J** option. The **\$PBS\_ARRAY\_INDEX** environment variable takes on each value in the range specified by the -J option. This variable can be used to identify the directories, created in advance for each input, and the input files needed by the sub-job. For an array-type job, the job itself must be declared re-runnable (**#PBS -r y**).

```

#!/bin/bash
#PBS -l select=1:ncpus=1:mpiprocs=1
#PBS -l walltime=24:00
#PBS -N job_array
#PBS -A <account_name>
#PBS -J 2-21
#PBS -r y
# Comment: in this example the first value of the range is 2 and not 1
# because the first line of the list resulting from "ls -l" command
# is a configuration file valid for all simulations. The elements of the
# range can take on any value, but they can not be more than 20

cd $PBS_O_WORKDIR

filename=`ls -l inputs/ | tail -n +${PBS_ARRAY_INDEX} | head -1`

mkdir outputs/$PBS_ARRAY_INDEX
cd outputs/$PBS_ARRAY_INDEX
cp ../../inputs/configuration_file .
cp ../../inputs/$filename .

../../exe $filename

```

In this example, we show how you can use, for instance, arbitrary input file names: you can put them into an "input" directory and select the file to process in each simulation by using the "tail and head" commands and the value of the environment variable \$PBS\_ARRAY\_INDEX. Each run creates an output directory (identified by the value of the array variable) where both the output and a copy of the corresponding input are written.

Please notice that you always need to add the PBS directive which declares the job to be re-runnable, which is mandatory in order to submit an array job:

```
#PBS -r y
```

Same useful commands:

**qsub job\_array.sh** --> to submit a job array

**qstat <job\_array\_id>** --> to check the status of your job array

**qstat -t <job\_array id>** --> to check the status of all sub-jobs of a job array

## Further documentation


-

More specific information about queues, limits and available accelerators are described on the "system specific" pages of this Guide, for [GALILEO](#), [PICO](#), and [MARCONI](#) as well as "man" pages about PBS commands:

-

```
> man pbs
> man qsub
> man qstat
> man qdel
> man ...
```

Outgoing links:

 Unknown macro: 'outgoing-links'