# Running jobs

From CC Doc

This page is intended for the user who is already familiar with the concepts of job scheduling and job scripts, and who wants guidance on submitting jobs to Compute Canada clusters. If you have not worked on a large shared computer cluster before, you should probably read **What is a scheduler?** first.

**All jobs must be submitted via the scheduler!**
Exceptions are made for compilation and other tasks not expected to consume more than about 10 CPU-minutes and about 4 gigabytes of RAM. Such tasks may be run on a login node. In no case should you run processes on compute nodes except via the scheduler.

On Compute Canada clusters, the job scheduler is the **Slurm Workload Manager (https://en.wikipedia.org/wiki/Slurm_Workload_Manager)**. Comprehensive **documentation for Slurm (https://slurm.schedmd.com/documentation.html)** is maintained by SchedMD. If you are coming to Slurm from PBS/Torque, SGE, LSF, or LoadLeveler, you might find this table of **corresponding commands (https://slurm.schedmd.com/rosetta.pdf)** useful.

## Contents

- 1 Use sbatch to submit jobs
- 2 Use squeue to list jobs
- 3 Where does the output go?
- 4 Accounts and projects
- 5 Examples of job scripts
  - 5.1 Serial job
  - 5.2 Array job
  - 5.3 Threaded or OpenMP job
  - 5.4 MPI job
  - 5.5 GPU job
- 6 Interactive jobs
- 7 Monitoring jobs
  - 7.1 Current jobs
    - 7.1.1 Email notification
  - 7.2 Completed jobs
  - 7.3 Attaching to a running job
- 8 Cancelling jobs
- 9 Resubmitting jobs for long running computations
  - 9.1 Restarting using job arrays
  - 9.2 Resubmission from the job script
- 10 Cluster particularities
- 11 Troubleshooting
  - 11.1 Avoid hidden characters in job scripts
  - 11.2 Cancellation of jobs with dependency conditions which cannot be met
  - 11.3 Job cannot load a module
  - 11.4 Jobs inherit environment variables
- 12 Job status and priority
- 13 Further reading

## Use `sbatch` to submit jobs

The command to submit a job is **`sbatch`** **(https://slurm.schedmd.com/sbatch.html)**:

```
[someuser@host ~]$ sbatch simple_job.sh
Submitted batch job 123456
```

A minimal Slurm job script looks like this:

**File :** simple_job.sh

```
#!/bin/bash
#SBATCH --time=00:01:00
#SBATCH --account=def-someuser
echo 'Hello, world!'
sleep 30
```

Directives (or "options") in the job script are prefixed with `#SBATCH` and must precede all executable commands. All available directives are described on the **sbatch page (https://slurm.schedmd.com/sbatch.html)**. Compute Canada policies require that you supply at least a time limit (`--time`) for each job. You may also need to supply an account name (`--account`). See **Accounts and projects** below.

Memory may be requested with `--mem-per-cpu` (memory per core) or `--mem` (memory per node). We recommend that you specify memory in megabytes (e.g. 8000M) rather than gigabytes (e.g. 8G). In many circumstances specifying memory requests in thousands of megabytes will result in shorter queue wait times than specifying gigabytes. For example, requesting `--mem=128G` (equivalent to 131072M) is more memory than is available for jobs on nodes with a nominal 128G . A job requesting `--mem=128G` qualifies for fewer nodes than one requesting `--mem=128000M`, and is therefore likely to wait longer. Similar reasoning also applies to `--mem-per-cpu` requests.

A default memory amount of 256 MB per core will be allocated unless you make some other memory request.

You can also specify directives as command-line arguments to `sbatch`. So for example,

```
[someuser@host ~]$ sbatch --time=00:30:00 simple_job.sh
```

will submit the above job script with a time limit of 30 minutes. The acceptable time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds".

Please be cautious if you use a script to submit multiple Slurm jobs in a short time. Submitting thousands of jobs at a time can cause Slurm to become **unresponsive** to other users. Consider using an **array job** instead, or use `sleep` to space out calls to `sbatch` by one second or more.

## Use squeue to list jobs

The **`squeue` (https://slurm.schedmd.com/squeue.html)** command lists pending and running jobs. Supply your username as an argument with `-u` to list only your own jobs:

```
[someuser@host ~]$ squeue -u $USER
      JOBID PARTITION      NAME     USER ST   TIME  NODES NODELIST(REASON)
     123456 cpubase_b  simple_j someuser  R   0:03      1 cdr234
     123457 cpubase_b  simple_j someuser PD             1 (Priority)
```

The ST column of the output shows the status of each job. The two most common states are "PD" for "pending" or "R" for "running". See the **squeue page (https://slurm.schedmd.com/squeue.html)** for more on selecting, formatting, and interpreting the `squeue` output.

**Do not** run `squeue` from a script or program at high frequency, *e.g.*, every few seconds. Responding to `squeue` adds load to Slurm, and may interfere with its performance or correct operation. See **Email notification** below for another way to learn when your job starts or ends.

## Where does the output go?

By default the output is placed in a file named "slurm-", suffixed with the job ID number and ".out", *e.g.* `slurm-123456.out`, in the directory from which the job was submitted. You can use `--output` to specify a different name or location. Certain replacement symbols can be used in the filename, *e.g.* `%j` will be replaced by the job ID number. See **sbatch (https://slurm.schedmd.com/sbatch.html)** for a complete list.

The following sample script sets a *job name* (which appears in `squeue` output) and sends the output to a file with a name constructed from the job name (%x) and the job ID number (%j).

**File :** name_output.sh

```
#!/bin/bash
#SBATCH --account=def-someuser
#SBATCH --time=00:01:00
#SBATCH --job-name=test
#SBATCH --output=%x-%j.out
echo 'Hello, world!'
```

Error output will normally appear in the same file as standard output, just as it would if you were typing commands interactively. If you want to send the standard error channel (stderr) to a separate file, use `--error`.

## Accounts and projects

Every job must have an associated *account name* corresponding to a Compute Canada **Resource Allocation Project (https://ccdb.computecanada.ca/me/faq#what_is_rap)** (RAP).

If you try to submit a job with `sbatch` and receive one of these messages:

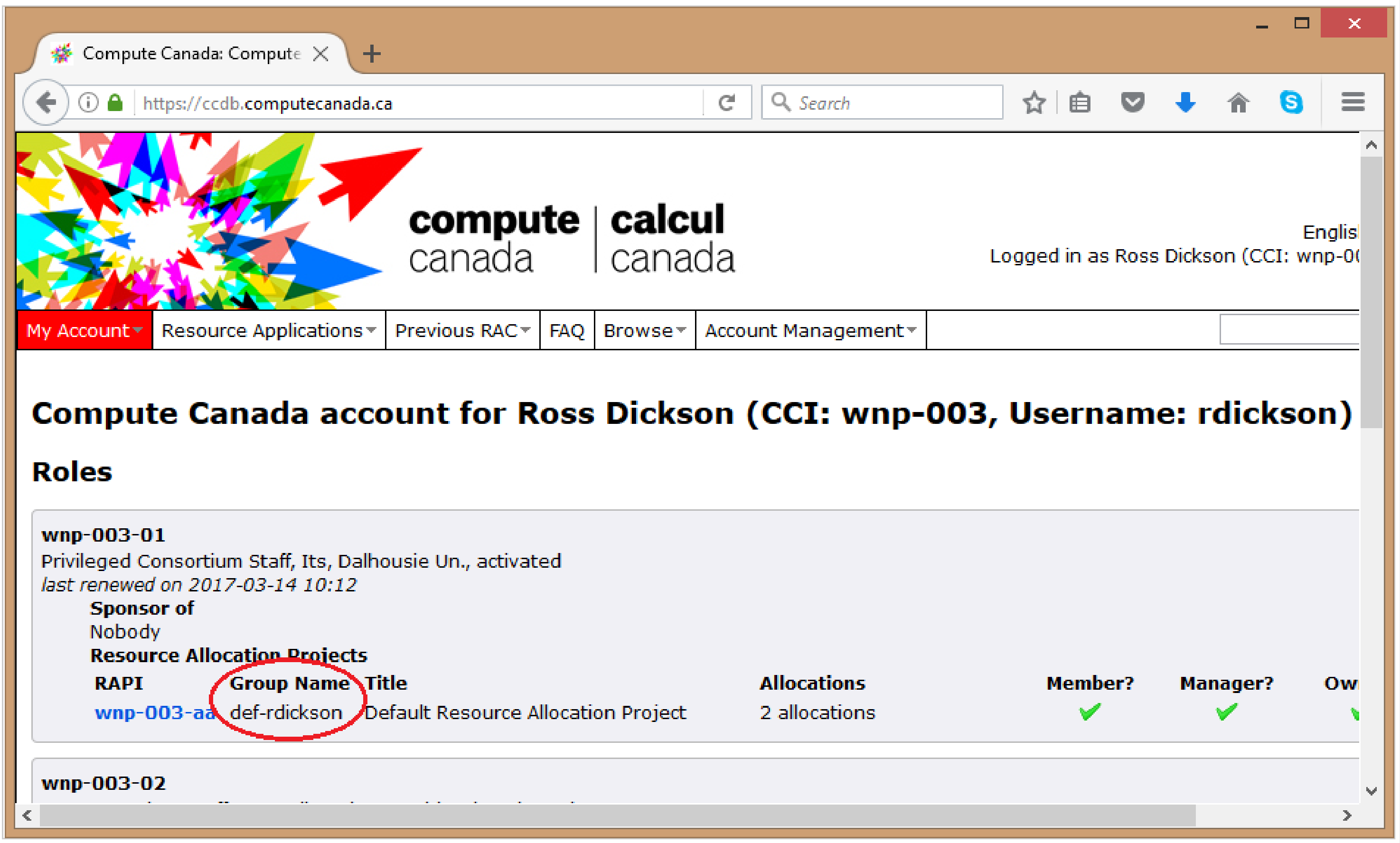
```
You are associated with multiple _cpu allocations...
Please specify one of the following accounts to submit this job:

You are associated with multiple _gpu allocations...
Please specify one of the following accounts to submit this job:
```

then you have more than one valid account, and you will have to specify one using the `--account` directive:

```
#SBATCH --account=def-user-ab
```

To find out which account name corresponds to a given Resource Allocation Project, log in to **CCDB (https://ccdb.computecanada.ca)** and click on "My Account -> Account Details". You will see a list of all the projects you are a member of. The string you should use with the `--account` for a given project is under the column **Group Name**. Note that a Resource Allocation Project may only apply to a specific cluster (or set of clusters) and therefore may not be transferable from one cluster to another.

In the illustration below, jobs submitted with `--account=def-rdickson` will be accounted against RAP wnp-003-aa.



Logged in as Ross Dickson (CCI: wnp-0

Compute Canada account for Ross Dickson (CCI: wnp-003, Username: rdickson)

Roles

wnp-003-01
Privileged Consortium Staff, Its, Dalhousie Un., activated
*last renewed on 2017-03-14 10:12*
Sponsor of
Nobody
Resource Allocation Projects

| RAPI | Group Name | Title | Allocations | Member? | Manager? | Ow |
|---|---|---|---|---|---|---|
| wnp-003-aa | def-rdickson | Default Resource Allocation Project | 2 allocations | ✓ | ✓ | ✓ |

wnp-003-02

Finding the group name for a Resource Allocation Project (RAP)

If you plan to use one account consistently for all jobs, once you have determined the right account name you may find it convenient to set the following three environment variables in your `~/.bashrc` file:

```
export SLURM_ACCOUNT=def-someuser
export SBATCH_ACCOUNT=$SLURM_ACCOUNT
export SALLOC_ACCOUNT=$SLURM_ACCOUNT
```

Slurm will use the value of `SBATCH_ACCOUNT` in place of the `--account` directive in the job script. Note that even if you supply an account name inside the job script, *the environment variable takes priority.* In order to override the environment variable you must supply an account name as a command-line argument to `sbatch`.

`SLURM_ACCOUNT` plays the same role as `SBATCH_ACCOUNT`, but for the `srun` command instead of `sbatch`. The same idea holds for `SALLOC_ACCOUNT`.

## Examples of job scripts

### Serial job

A serial job is a job which only requests a single core. It is the simplest type of job. The "simple_job.sh" which appears above in **"Use sbatch to submit jobs"** is an example.

### Array job

Also known as a *task array*, an array job is a way to submit a whole set of jobs with one command. The individual jobs in the array are distinguished by an environment variable, `$SLURM_ARRAY_TASK_ID`, which is set to a different value for each instance of the job. The following example will create 10 tasks, with values of `$SLURM_ARRAY_TASK_ID` ranging from 1 to 10:

**File :** array_job.sh

```
#!/bin/bash
#SBATCH --account=def-someuser
#SBATCH --time=0-0:5
#SBATCH --array=1-10
./myapplication $SLURM_ARRAY_TASK_ID
```

For more examples, see **Job arrays**. See **Job Array Support (https://slurm.schedmd.com/job_array.html)** at SchedMD.com for detailed documentation.

### Threaded or OpenMP job

This example script launches a single process with eight CPU cores. Bear in mind that for an application to use OpenMP it must be compiled with the appropriate flag, e.g. `gcc -fopenmp ...` or `icc -openmp ...`

**File :** openmp_job.sh

```
#!/bin/bash
#SBATCH --account=def-someuser
#SBATCH --time=0-0:5
#SBATCH --cpus-per-task=8
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./ompHello
```

### MPI job

This example script launches four MPI processes, each with 1024 MB of memory. The run time is limited to 5 minutes.

**File :** mpi_job.sh

```
#!/bin/bash
#SBATCH --account=def-someuser
#SBATCH --ntasks=4               # number of MPI processes
#SBATCH --mem-per-cpu=1024M      # memory; default unit is megabytes
#SBATCH --time=0-00:05           # time (DD-HH:MM)
srun ./mpi_program               # mpirun or mpiexec also work
```

Large MPI jobs, specifically those which can efficiently use whole nodes, should use `--nodes` and `--ntasks-per-node` instead of `--ntasks`. Hybrid MPI/threaded jobs are also possible. For more on these and other options relating to distributed parallel jobs, see **Advanced MPI scheduling**.

For more on writing and running parallel programs with OpenMP, see **OpenMP**.

### GPU job

There are many options involved in requesting GPUs because

- the GPU-equipped nodes at **Cedar** and **Graham** have different configurations,
- there are two different configurations at Cedar, and
- there are different policies for the different Cedar GPU nodes.

Please see **Using GPUs with Slurm** for a discussion and examples of how to schedule various job types on the available GPU resources.

## Interactive jobs

Though batch submission is the most common and most efficient way to take advantage of our clusters, interactive jobs are also supported. These can be useful for things like:

- Data exploration at the command line
- Interactive "console tools" like R and iPython
- Significant software development, debugging, or compiling

You can start an interactive session on a compute node with **salloc (https://slurm.schedmd.com/salloc.html)**. In the following example we request two tasks, which corresponds to two CPU cores, for an hour:

```
[name@login ~]$ salloc --time=1:0:0 --ntasks=2 --account=def-someuser
salloc: Granted job allocation 1234567
[name@node01 ~]$ ...             # do some work
[name@node01 ~]$ exit            # terminate the allocation
salloc: Relinquishing job allocation 1234567
```

It is also possible to run graphical programs interactively on a compute node by adding the **--x11** flag to your *salloc* command. In order for this to work, you must first connect to the cluster with X11 forwarding enabled (see the **SSH** page for instructions on how to do that).

## Monitoring jobs

### Current jobs

By default **squeue (https://slurm.schedmd.com/squeue.html)** will show all the jobs the scheduler is managing at the moment. It may run much faster if you ask only about your own jobs with

```
squeue -u $USER
```

You can show only running jobs, or only pending jobs:

```
squeue -u <username> -t RUNNING
squeue -u <username> -t PENDING
```

You can show detailed information for a specific job with **scontrol (https://slurm.schedmd.com/scontrol.html)**:

```
scontrol show job -dd <jobid>
```

**Do not** run `squeue` from a script or program at high frequency, e.g., every few seconds. Responding to `squeue` adds load to Slurm, and may interfere with its performance or correct operation.

**Email notification**

You can ask to be notified by email of certain job conditions by supplying options to **sbatch (https://slurm.schedmd.com/sbatch.html)**:

```
#SBATCH --mail-user=<email_address>
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END
#SBATCH --mail-type=FAIL
#SBATCH --mail-type=REQUEUE
#SBATCH --mail-type=ALL
```

## Completed jobs

Get a short summary of the CPU- and memory-efficiency of a job with `seff`:

```
$ seff 12345678
Job ID: 12345678
Cluster: cedar
User/Group: jsmith/jsmith
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 02:48:58
CPU Efficiency: 99.72% of 02:49:26 core-walltime
Job Wall-clock time: 02:49:26
Memory Utilized: 213.85 MB
Memory Efficiency: 0.17% of 125.00 GB
```

Find more detailed information about a completed job with **sacct (https://slurm.schedmd.com/sacct.html)**, and optionally, control what it prints using `--format`:

```
sacct -j <jobid>
sacct -j <jobid> --format=JobID,JobName,MaxRSS,Elapsed
```

The output from `sacct` typically includes records labelled `.bat+` and `.ext+`, and possibly `.0, .1, .2, ...`. The batch step (`.bat+`) is your submission script - for many jobs that's where the main part of the work is done and where the resources are consumed. If you use `srun` in your submission script, that would create a `.0` step that would consume most of the resources. The extern (`.ext+`) step is basically prologue and epilogue and normally doesn't consume any significant resources.

If a node fails while running a job, the job may be restarted. `sacct` will normally show you only the record for the last (presumably successful) run. If you wish to see all records related to a given job, add the `--duplicates` option.

Use the MaxRSS accounting field to determine how much memory a job needed. The value returned will be the largest **resident set size (https://en.wikipedia.org/wiki/Resident_set_size)** for any of the tasks. If you want to know which task and node this occurred on, print the MaxRSSTask and MaxRSSNode fields also.

The **sstat (https://slurm.schedmd.com/sstat.html)** command works on a running job much the same way that **sacct (https://slurm.schedmd.com/sacct.html)** works on a completed job.

## Attaching to a running job

It is possible to connect to the node running a job and execute new processes there. You might want to do this for troubleshooting or to monitor the progress of a job.

Suppose you want to run the utility **`nvidia-smi` (https://developer.nvidia.com/nvidia-system-management-interface)** to monitor GPU usage on a node where you have a job running. The following command runs `watch` on the node assigned to the given job, which in turn runs `nvidia-smi` every 30 seconds, displaying the output on your terminal.

```
[name@server ~]$ srun --jobid 123456 --pty watch -n 30 nvidia-smi
```

It is possible to launch multiple monitoring commands using **`tmux` (https://en.wikipedia.org/wiki/Tmux)**. The following command launches `htop` and `nvidia-smi` in separate panes to monitor the activity on a node assigned to the given job.

```
[name@server ~]$ srun --jobid 123456 --pty tmux new-session -d 'htop -u $USER' \; split-window -h 'watch nvidia-smi' \; attach
```

Processes launched with `srun` share the resources with the job specified. You should therefore be careful not to launch processes that would use a significant portion of the resources allocated for the job. Using too much memory, for example, might result in the job being killed; using too many

CPU cycles will slow down the job.

**Note:** The `srun` commands shown above work only to monitor a job submitted with `sbatch`. To monitor an interactive job, create multiple panes with `tmux` and start each process in its own pane.

## Cancelling jobs

Use **scancel (https://slurm.schedmd.com/scancel.html)** with the job ID to cancel a job:

```
scancel <jobid>
```

You can also use it to cancel all your jobs, or all your pending jobs:

```
scancel -u $USER
scancel -t PENDING -u $USER
```

## Resubmitting jobs for long running computations

When a computation is going to require a long time to complete, so long that it cannot be done within the time limits on the system, the application you are running must support **checkpointing**. The application should be able to save its state to a file, called a *checkpoint file*, and then it should be able to restart and continue the computation from that saved state.

For many users restarting a calculation will be rare and may be done manually, but some workflows require frequent restarts. In this case some kind of automation technique may be employed.

Here are two recommended methods of automatic restarting:

- Using SLURM **job arrays**.
- Resubmitting from the end of the job script.

### Restarting using job arrays

Using the `--array=1-100%10` syntax mentioned earlier one can submit a collection of identical jobs with the condition that only one job of them will run at any given time. The script should be written to ensure that the last checkpoint is always used for the next job. The number of restarts is fixed by the `--array` argument.

Consider, for example, a molecular dynamics simulations that has to be run for 1 000 000 steps, and such simulation does not fit into the time limit on the cluster. We can split the simulation into 10 smaller jobs of 100 000 steps, one after another.

An example of using a job array to restart a simulation:

**File :** job_array_restart.sh

```
#!/bin/bash
# ---------------------------------------------------------------------
# SLURM script for a multi-step job on a Compute Canada cluster.
# ---------------------------------------------------------------------
#SBATCH --account=def-someuser
#SBATCH --cpus-per-task=1
#SBATCH --time=0-10:00
#SBATCH --mem=100M
#SBATCH --array=1-10%1   # Run a 10-job array, one job at a time.
# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------
echo ""
echo "Job Array ID / Job ID: $SLURM_ARRAY_JOB_ID / $SLURM_JOB_ID"
echo "This is job $SLURM_ARRAY_TASK_ID out of $SLURM_ARRAY_TASK_COUNT jobs."
echo ""
# ---------------------------------------------------------------------
# Run your simulation step here...

if test -e state.cpt; then
     # There is a checkpoint file, restart;
     mdrun --restart state.cpt
else
     # There is no checkpoint file, start a new simulation.
     mdrun
fi

# ---------------------------------------------------------------------
```

```
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

### Resubmission from the job script

In this case one submits a job that runs the first chunk of the calculation and saves a checkpoint. Once the chunk is done but before the allocated run-time of the job has elapsed, the script checks if the end of the calculation has been reached. If the calculation is not yet finished, the script submits a copy of itself to continue working.

An example of a job script with resubmission:

**File :** job_resubmission.sh

```bash
#!/bin/bash
# ---------------------------------------------------------------------
# SLURM script for job resubmission on a Compute Canada cluster.
# ---------------------------------------------------------------------
#SBATCH --job-name=job_chain
#SBATCH --account=def-someuser
#SBATCH --cpus-per-task=1
#SBATCH --time=0-10:00
#SBATCH --mem=100M
# ---------------------------------------------------------------------
echo "Current working directory: `pwd`"
echo "Starting run at: `date`"
# ---------------------------------------------------------------------
# Run your simulation step here...

if test -e state.cpt; then
     # There is a checkpoint file, restart;
     mdrun --restart state.cpt
else
     # There is no checkpoint file, start a new simulation.
     mdrun
fi

# Resubmit if not all work has been done yet.
# You must define the function work_should_continue().
if work_should_continue; then
     sbatch ${BASH_SOURCE[0]}
fi

# ---------------------------------------------------------------------
echo "Job finished with exit code $? at: `date`"
# ---------------------------------------------------------------------
```

**Please note:** The test to determine whether to submit a followup job, abbreviated as `work_should_continue` in the above example, should be a *positive test*. There may be a temptation to test for a stopping condition (e.g. is some convergence criterion met?) and submit a new job if the condition is *not* detected. But if some error arises that you didn't foresee, the stopping condition might never be met and your chain of jobs may continue indefinitely, doing nothing useful.

## Cluster particularities

There are certain differences in the job scheduling policies from one Compute Canada cluster to another and these are summarized by tab in the following section:

Cedar | Niagara

Cedar has two distinct CPU architectures available: **Broadwell (https://en.wikipedia.org/wiki/Broadwell_(microarchitecture))** and **Skylake (https://en.wikipedia.org/wiki/Skylake_(microarchitecture))**. Users requiring a specific architecture can request it when submitting a job using the `--constraint` flag. Note that the names should be written all in lower-case, `skylake` or `broadwell`.

An example job requesting the `skylake` feature on Cedar:

```
#!/bin/bash
#SBATCH --account=def-someuser
#SBATCH --constraint=skylake
#SBATCH --time=0:5:0
# Display CPU-specific information with 'lscpu'.
# Skylake CPUs will have 'avx512f' in the 'Flags' section of the output.
lscpu
```

Keep in mind that a job which would have obtained an entire node for itself by specifying for example `#SBATCH --cpus-per-task=32` will now share the remaining 16 CPU cores with another job if it happens to use a Skylake node; if you wish to reserve the entire node you will need to request all 48 cores or add the `#SBATCH --constraint=broadwell` option to your job script.

*If you are unsure if your job requires a specific architecture, do not use this option.* Jobs that do not specify a CPU architecture can be scheduled on either Broadwell or Skylake nodes, and will therefore generally start earlier.

## Troubleshooting

### Avoid hidden characters in job scripts

Preparing a job script with a *word processor* instead of a *text editor* is a common cause of trouble. Best practice is to prepare your job script on the cluster using an **editor** such as nano, vim, or emacs. If you prefer to prepare or alter the script off-line, then:

- **Windows users:**
  - Use a text editor such as Notepad or **Notepad++ (https://notepad-plus-plus.org/)**.
  - After uploading the script, use `dos2unix` to change Windows end-of-line characters to Linux end-of-line characters.
- **Mac users:**
  - Open a terminal window and use an **editor** such as nano, vim, or emacs.

### Cancellation of jobs with dependency conditions which cannot be met

A job submitted with `dependency=afterok:<jobid>` is a "dependent job". A dependent job will wait for the parent job to be completed. If the parent job fails (that is, ends with a non-zero exit code) the dependent job can never be scheduled and so will be automatically cancelled. See **sbatch (https://slurm.schedmd.com/sbatch.html)** for more on dependency.

### Job cannot load a module

It is possible to see an error such as:

```
Lmod has detected the following error: These module(s) exist but cannot be
loaded as requested: "<module-name>/<version>"
   Try: "module spider <module-name>/<version>" to see how to load the module(s).
```

This can occur if the particular module has an unsatisfied prerequisite. For example

```
[name@server]$ module load gcc
[name@server]$ module load quantumespresso/6.1
Lmod has detected the following error:  These module(s) exist but cannot be loaded as requested: "quantumespresso/6.1"
   Try: "module spider quantumespresso/6.1" to see how to load the module(s).
[name@server]$ module spider quantumespresso/6.1

-----------------------------------------
  quantumespresso: quantumespresso/6.1
------------------------------------------
    Description:
      Quantum ESPRESSO is an integrated suite of computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is
      norm-conserving and ultrasoft).

    Properties:
      Chemistry libraries/apps / Logiciels de chimie

    You will need to load all module(s) on any one of the lines below before the "quantumespresso/6.1" module is available to load.

      nixpkgs/16.09  intel/2016.4  openmpi/2.1.1

    Help:
```

```
    Description
    ===========
    Quantum ESPRESSO  is an integrated suite of computer codes
     for electronic-structure calculations and materials modeling at the nanoscale.
     It is based on density-functional theory, plane waves, and pseudopotentials
      (both norm-conserving and ultrasoft).


    More information
    ================
     - Homepage: http://www.pwscf.org/
```

In this case adding the line `module load nixpkgs/16.09 intel/2016.4 openmpi/2.1.1` to your job script before loading the "quantumespresso/6.1" will solve this problem.

#### Jobs inherit environment variables

By default a job will inherit the environment variables of the shell where the job was submitted. The **module** command, which is used to make various software packages available, changes and sets environment variables. Changes will propagate to any job submitted from the shell and thus could affect the job's ability to load modules if there are missing prerequisites. It is best to include the line `module purge` in your job script before loading all the required modules to ensure a consistent state for each job submission and avoid changes made in your shell affecting your jobs.

Inheriting environment settings from the submitting shell can sometimes lead to hard-to-diagnose problems. If you wish to suppress this inheritance, use the `--export=none` directive when submitting jobs.

## Job status and priority

- For a discussion of how job priority is determined and how things like time limits may affect the scheduling of your jobs at Cedar and Graham, see **Job scheduling policies**.

## Further reading

- Comprehensive **documentation (https://slurm.schedmd.com/documentation.html)** is maintained by SchedMD, as well as some **tutorials (https://slurm.schedmd.com/tutorials.html)**.
  - **sbatch (https://slurm.schedmd.com/sbatch.html)** command options
- There is also a **"Rosetta stone" (https://slurm.schedmd.com/rosetta.pdf)** mapping commands and directives from PBS/Torque, SGE, LSF, and LoadLeveler, to SLURM. NERSC also offers some **tables comparing Torque and SLURM (http://www.nersc.gov/users/computational-systems/cori/running-jobs/for-edison-users/torque-moab-vs-slurm-comparisons/)**.
- Here is a text tutorial from **CÉCI (http://www.ceci-hpc.be/slurm_tutorial.html)**, Belgium
- Here is a rather minimal text tutorial from **Bright Computing (http://www.brightcomputing.com/blog/bid/174099/slurm-101-basic-slurm-usage-for-linux-clusters)**

Retrieved from "https://docs.computecanada.ca/mediawiki/index.php?title=Running_jobs&oldid=69369"

Category: SLURM

---

- This page was last modified on 8 March 2019, at 19:55.