

# Queuing-System (SLURM) and Jobfiles

## *Important info:*

Big parts of this script was taken from the documentation of the [Hamburg HPC Competence Center \(HHCC\)](#). Please visit their website for more details on the [Use of the Command Line Interface](#) or about [Using Shell Scripts](#).

## *Description:*

- You will learn to how to use Environment Modules, a widely used system for handling different software environments (basic level)
- You will learn to use the workload manager SLURM to allocate HPC resources (e.g. CPUs) and to submit a batch job (basic level)
- You will learn how a simple jobscript looks like and how to submit it (basic to intermediate level)

## General Information

*Environment Modules* are a tool for managing environment variables of the shell. Modules can be loaded and unloaded dynamically and atomically, in a clean fashion. Details can be found on the [official website](#).

The workload manager used on the Phoenix-Cluster is SLURM (Simple Linux Utility for Resource Management). SLURM a widely used open source workload managers for large and small Linux clusters which is controlled via a CLI (Command Line Interface). Details can be found in the [official documentation](#).

## Environment Modules

### *Introduction:*

The `module load` command extends variables containing search paths (e.g. `PATH` or `MANPATH`). The `module unload` command is the corresponding inverse operation, it removes entries from search paths. By extending search paths software is made callable. Effectively software can be provided through Modules. An advantage over defining environment variables directly in the shell is that Modules allow to undo changes of environment variables. The idea of introducing Modules is to be

able to define software environments in a modular way. In the context of HPC, Modules make it easy to switch compilers or libraries, or to choose between different versions of an application software package.

### *Naming:*

Names of Modules have the format `program/version`, just `program` or even a slightly more nested path description. Modules can be loaded (and always be unloaded) without specifying a version. If the `version` is not specified the default `version` will be loaded. The default `version` is either explicitly defined (and will be marked in the output of `module avail`) or module will load the `version` that appears to be the latest one. Because defaults can change **versions should always be given if reproducibility is required**.

### *Dependences and conflicts:*

Modules can have dependences, i.e. a Module can enforce that other Modules that it depends on must be loaded before the Module itself can be loaded. Module can be conflicting, i.e. these modules must not be loaded at the same time (e.g. two version of a compiler). A conflicting Module must be unloaded before the Module it conflicts with can be loaded.

### *Caveats:*

The name Modules suggest that Modules can be picked and combined in a modular fashion. For Modules providing application packages this is true (up to possible dependences and conflicts described above), i.e. it is possible to chose any combination of application software.

However, today, environments for building software are not modular anymore. In particular, it is no longer guaranteed that a library that was built with one compiler can be used with code generated by a different compiler. Hence, the corresponding Modules cannot be modular either. A popular way to handle this situation is to append compiler information to the version information of library Modules. Firstly, this leads to long names and secondly, to very many Modules that are hard to overlook. A more modern way is to build up toolchains with Modules. For example, in such a toolchain only compiler Modules are available at the beginning. Once a compiler Module is loaded, MPI libraries (the next level of tools) become available and after that all other Modules (that were built with that chain).

### *Important commands:*

Important Module commands are:

list Modules currently loaded	<code>module list</code>
list available Modules	<code>module avail</code>
load a Module	<code>module load program[/version]</code>
unload a Module	<code>module unload program</code>
switch a Module (e.g. compiler version)	<code>module switch program program/version</code>

add or remove a directory/path to the Module search path (e.g. by an own Module directory)	<code>module [un]use [--append] path</code>
---	---

## *Self-documentation:*

Modules are self-documented:

show the actions of a Module	<code>module display program/version</code>
short description of [one or] all Modules	<code>module whatis [program/version]</code>
longer help text on a Module	<code>module help program/version</code>
help on module itself	<code>module help</code>

# Basics of SLURM

## *Introduction:*

There are three key functions of SLURM described on the SLURM website:

*“... First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work. ...”*

SLURM’s default scheduling is based on a FIFO-queue, which is typically enhanced with the Multifactor Priority Plugin to achieve a very versatile facility for ordering the queue of jobs waiting to be scheduled. In contrast to other workload managers SLURM does not use several job queues. Cluster nodes in a SLURM configuration can be assigned to multiple partitions by the cluster administrators instead. This enables the same functionality.

A compute center will seek to configure SLURM in a way that resource utilization and throughput are maximized, waiting times and turnaround times are minimized, and all users are treated fairly.

The basic functionality of SLURM can be divided into three areas:

- Job submission and cancellation
- Monitoring job and system information
- Retrieving accounting information

## *Job submission and cancellation:*

There are three commands for handling job submissions:

- `sbatch`

- submits a batch job script to SLURM's job queue for (later) execution. The batch script may be given to sbatch by a file name on the command line or can be read from stdin. Resources needed by the job may be specified via command line options and/or directly in the job script. A job script may contain several job steps to perform several parallel tasks within the same script. Job steps themselves may be run sequentially or in parallel. SLURM regards the script as the first job step.
- `salloc`
  - allocates a set of nodes, typically for interactive use. Resources needed may be specified via command line options.
- `srun`
  - usually runs a command on nodes previously allocated via sbatch or salloc. Each invocation of srun within a job script corresponds to a job step and launches parallel tasks across the allocated resources. A task is represented e.g. by a program, command, or script. If srun is not invoked within an allocation it will via command line options first create a resource allocation in which to run the parallel job.

SLURM assigns a unique *jobid* (integer number) to each job when it is submitted. This *jobid* is returned at submission time or can be obtained from the `squeue` command.

The `scancel` command is used to abort a job or job step that is running or waiting for execution.

The `scontrol` command is mainly used by cluster administrators to view or modify the configuration of the SLURM system but it also offers the users the possibility to control their jobs (e.g. to hold and release a pending job).

The Table below lists basic user activities for job submission and cancellation and the corresponding SLURM commands.

User activities for job submission and cancellation (user supplied information is given in *italics*)

User activity	SLURM command
Submit a job script for (later) execution	<code>sbatch</code> <i>job-script</i>
Allocate a set of nodes for interactive use	<code>salloc</code> <i>-nodes=N</i>
Launch a parallel task (e.g. program, command, or script) within allocated resources by <code>sbatch</code> (i.e. within a job script) or <code>salloc</code>	<code>srun</code> <i>task</i>
Allocate a set of nodes and launch a parallel task directly	<code>srun</code> <i>-nodes=N task</i>
Abort a job that is running or waiting for execution	<code>scancel</code> <i>jobid</i>
Abort all jobs of a user	<code>scancel</code> <i>-user=username</i> or generally <code>scancel</code> <i>-user=\$USER</i>

User activity	SLURM command
Put a job on hold (i.e. pause waiting) and Release a job from hold (These related commands are rarely used in standard operation.)	<code>scontrol hold <i>jobid</i></code> <code>scontrol release <i>jobid</i></code>

The major command line options that are used for `sbatch` and `salloc` are listed in the Table below. These options can also be specified for `srun`, if `srun` is not used in the context of nodes previously allocated via `sbatch` or `salloc`.

Major `sbatch` and `salloc` options

Specification	Option	Comments
Number of nodes requested	<code>-nodes=<i>N</i></code>	
Number of tasks to invoke on each node	<code>-tasks-per-node=<i>n</i></code>	Can be used to specify the number of cores to use per node, e.g. to avoid <a href="#">hyper-threading</a> . (If option is omitted, all cores and hyperthreads are used; Hint: using hyperthreads is not always advantageous.)
Partition	<code>-partition= <i>partitionname</i></code>	
Job time limit	<code>-time=<i>time-limit</i></code>	<code>time-limit</code> may be given as minutes or in <code>hh:mm:ss</code> or <code>d-hh:mm:ss</code> format ( <code>d</code> means number of days)
Output file	<code>-output=<i>out</i></code>	Location of stdout redirection

For the `sbatch` command these options may also be specified directly in the job script using a pseudo comment directive starting with `#SBATCH` as a prefix. The directives must precede any executable command in the batch script:

```
#!/bin/bash
#SBATCH --partition=std
#SBATCH --nodes=2
#SBATCH --tasks-per-node=16
#SBATCH --time=00:10:00
...
srun ./helloParallelWorld
```

A complete list of parameters can be retrieved from the `man` pages for `sbatch`, `salloc`, or `srun`, e.g. via

```
man sbatch
```

*Monitoring job and system information:*

There are four commands for monitoring job and system information:

- `sinfo`
  - shows current information about nodes and partitions for a system managed by SLURM. Command line options can be used to filter, sort, and format the output in a variety of ways. By default it essentially shows for each partition if it is available and how many nodes and which nodes in the partition are allocated or idle (or are possibly in another state like down or drain, i.e. not available for some time). This is useful for the user e.g. to decide in which partition to run a job. The number of allocated and idle nodes indicates the actual utilization of the cluster.
- `squeue`
  - shows current information about jobs in the SLURM scheduling queue. Command line options can be used to filter, sort, and format the output in a variety of ways. By default it lists all pending jobs, sorted descending by their priority, followed by all running jobs, sorted descending by their priority. The major job states are:
    - R for Running
    - PD for Pending
    - CD for Completed
    - F for Failed
    - CA for Cancelled
  - The `TIME` column shows for running jobs their execution time so far (or 0:00 for pending jobs).
  - The `NODELIST (REASON)` column shows either on which nodes a job is running or why the job is pending. A job is pending for two main reasons:
    - it is still waiting for resources to become scheduled, shown as `(Resources)`,
    - its priority is still not sufficient for it to become executed, shown as `(Priority)`, i.e. there are other jobs with a higher priority pending in the queue.
  - The position of a pending job in the queue indicates how many jobs are executed before and after it. The `squeue` command is the main way to monitor a job and can e.g. also be used to get the information about the expected starting time of a job (see Table below).
- `sstat`
  - is mainly used to display various status information of a running job taken as a snapshot. The information relates to CPU, task, node, Resident Set Size (RSS), and virtual memory (VM), etc.
- `scontrol`
  - is mainly used by cluster administrators to view or modify the configuration of the SLURM system, but it also offers users the possibility to get some information about the cluster configuration (e.g. about partitions, nodes, and jobs).

The Table below lists basic user activities for job and system monitoring and the corresponding SLURM commands.

User activity	SLURM command
---------------	---------------

View information about currently available nodes and partitions. The state of a partition may be <code>UP</code> , <code>DOWN</code> , or <code>INACTIVE</code> . If the state is <code>INACTIVE</code> , no new submissions are allowed to the partition.	<code>sinfo</code> <code>[-partition=<i>partitionname</i>]</code>
View summary about currently available nodes and partitions. The <code>NODES</code> ( <code>A/I/O/T</code> ) column contains corresponding number of nodes being allocated, idle, in some other state and the total of the three numbers.	<code>sinfo</code> <code>-s</code>
Check the state of all jobs.	<code>squeue</code>
Check the state of all own jobs.	<code>squeue</code> <code>-user=\$USER</code>
Check the state of a single job.	<code>squeue</code> <code>-j <i>jobid</i></code>
Check the expected starting time of a pending job.	<code>squeue</code> <code>-start -j <i>jobid</i></code>
Display status information of a running job (e.g. average CPU time, average Virtual Memory (VM) usage – see <code>sstat</code> <code>-helpformat</code> and <code>man sstat</code> for information on more options).	<code>sstat</code> <code>-format=AveCPU, AveVMSize -j <i>jobid</i></code>
View SLURM configuration information for a partition cluster node (e.g. associated nodes).	<code>scontrol</code> <code>show partition <i>partitionname</i></code>
View SLURM configuration information for a cluster node.	<code>scontrol</code> <code>show node <i>nodename</i></code>
View detailed job information.	<code>scontrol</code> <code>show job <i>jobid</i></code>

## Retrieving accounting information:

There are two commands for retrieving accounting information:

- `sacct`
  - shows accounting information for jobs and job steps in the SLURM job accounting log or SLURM database. For active jobs the accounting information is accessed via the job accounting log file. For completed jobs it is accessed via the log data saved in the SLURM database. Command line options can be used to filter, sort, and format the output in a variety of ways. Columns for jobid, jobname, partition, account, allocated CPUs, state, and exit code are shown by default for each of the user's jobs eligible after midnight of the current day.
- `sacctmgr`
  - is mainly used by cluster administrators to view or modify the SLURM account information, but it also offers users the possibility to get some information about their account. The account information is maintained within the SLURM database. Command line options can be used to filter, sort, and format the output in a variety of ways.
  - The Table below lists basic user activities for retrieving accounting information and the corresponding SLURM commands.

User Activity	SLURM Command
View job account information for a specific job.	<code>sacct -j <i>jobid</i></code>
View all job information from a specific start date (given as yyyy-mm-dd).	<code>sacct -S <i>startdate</i> -u \$USER</code>
View execution time for (completed) job (formatted as days-hh:mm:ss, cumulated over job steps, and without any header).	<code>sacct -n -X -P -o Elapsed -j <i>jobid</i></code>

## Submitting a batch job:

Below an example script for a SLURM batch job – in the sense of a hello world program – is given. The job is suited to be run in the Phoenix HPC cluster at the Gauß-IT-Zentrum. For other cluster systems some appropriate adjustments will probably be necessary.

```
#!/bin/bash
# Do not forget to select a proper partition if the default
# one is no fit for the job! You can do that either in the sbatch
# command line or here with the other settings.
#SBATCH --partition=standard
# Number of nodes used:
#SBATCH --nodes=2
# Wall clock limit:
#SBATCH --time=12:00:00
# Name of the job:
#SBATCH --job-name=nearest
# Number of tasks (cores) per node:
#SBATCH --ntasks-per-node=20

# If needed, set your working environment here.
working_dir=~
cd $working_dir

# Load environment modules for your application here.
module load comp/gcc/6.3.0
module load mpi/openmpi/2.1.0/gcc

# Execute the application.
mpiexec -np 40 ./test/mpinearest
```

The job script file above can be stored e.g. in `$HOME/hello_world.sh` (`$HOME` is mapped to the user's home directory).

The job is submitted to SLURM's batch queue using the default value for partition (scontrol show partitions (also see above) can be used to show that information):

```
[exampleusername@node001 14:48:33]~$ sbatch $HOME/hello_world.sh
Submitted batch job 123456
```

The start time can be selected via `-begin`, for example::

```
--begin=16:00
--begin=now+1hour
--begin=now+60 (seconds by default)
--begin=2010-01-20T12:34:00
```

More information can be found via `man sbatch`. All parameters shown there can be included in the jobscript via `#SBATCH`.

The output of `sbatch` will contain the jobid, like 123456 in this example. During execution the output of the job is written to a file, named `slurm-123456.out`. If there had been errors (i.e. any output to the `stderrstream`) a corresponding file named `slurm-123456.err` would have been created.

### *Cancelling a batch job:*

```
scancel <jobid>
```

The required ID can be viewed via the general command `squeue` or the user specific command `squeue -u $USER`.

If you want to delete all jobs of a user:

```
scancel -u <username>
```

### *How to change a node status (root only):*

```
scontrol update nodename=node[005-008] state=drain reason="RMA"
```

This command will exclude the node from the list of available nodes. This ensures that no more jobs can be submitted to this node, allowing it to be used for testing etc.

```
scontrol update nodename=node[005-008] state=idle
```

This reverses the previous command and returns the node back to the list of available nodes. Executing this command might also be necessary if a node crash caused a removal of a node from the batch system.

## Interactive jobs (intermediate difficulty):

### *Method one:*

Assume you have submitted a job as follows:

```
sbatch beispiel.job
Submitted batch job 1256
```

Let the corresponding jobfile be the following:

```
beispiel.job

#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --time=7-00:00:00
#SBATCH --job-name=towhee
#SBATCH --ntasks-per-node=1

cd ~/data_dir
sleep 168h
```

In this case, the command `squeue -l` will show you which node the job is currently running on. For example:

```
1256 standard towhee raskrato RUNNING      0:04 7-00:00:00      1 node282
```

You can then log onto that node via `ssh node282` and start a new shell via [screen](#) (please follow the link for further information). The program can then be started in this new shell.

Once you are done, you can exit the shell via:

```
strg a d
```

- You can start as many shells as you like. The command `screen -r` will show a list of all shells (if it is only one, you will instead return to said shell).
- You can access a shell running in the background via `screen -r <shellnummer>`.
- You can quit a shell by pressing the key-combination `CTRL+C` and typing in `exit`.

Another way to use the allocated nodes is via the `salloc` command (see method two below).

### Method two:

Interactive sessions under control of the batch system can be created via `salloc`. `salloc` differs from `sbatch` by the fact that resources are initially only reserved (i.e. allocated) without executing a job script. Also, the session is running on the node on which `salloc` was invoked (but not on a compute node in contrast to submission with `sbatch`). This is often useful during the interactive development of a parallel program.

A single node is reserved for interactive usage as follows:

```
[exampleusername@node001 14:48:33]~$ salloc
```

When the resources are granted by SLURM, `salloc` will start a new shell on the (login or head) node where `salloc` was executed. This interactive session is terminated by exiting the shell or by

reaching the time limit.

An OpenMP program using  $N$  threads, for example, can be started on the allocated node as follows:

```
[exampleusername@node001 14:48:33]~$ export OMP_NUM_THREADS=N  
[exampleusername@node001 14:48:33]~$ srun my-openmp-binary
```

To start an interactive parallel MPI program  $N$  nodes can be allocated as follows:

```
[exampleusername@node001 14:48:33]~$ salloc --nodes=N
```

The MPI Program using  $n=32$  processes, for example, can be started on the allocated nodes as follows:

```
[exampleusername@node001 14:48:33]~$ mpirun -np 32 my-mpi-binary
```

Another way to use the allocated nodes is to use ssh to establish connections to them (see method one above).

---

Revision #3

Created 12 March 2024 14:09:37 by Michael Giemsa

Updated 12 March 2024 15:02:12 by Michael Giemsa