

Useful Slurm commands

Slurm provides a variety of tools that allow a user to manage and understand their jobs. This tutorial will introduce these tools, as well as provide details on how to use them.

Finding information in the work queue with squeue

The `squeue` command is a tool we use to get information about the jobs in the queue. By default, the `squeue` command will print the job ID, the partition, the job name, the job user, the job status, the running time, the number of nodes, and the list of allocated nodes:

```
squeue
JOBID PARTITION  NAME  USER ST  TIME  NODES NODELIST(REASON)
111111  batch  my_job  myuser R  1:21:59  1 node0101-1
```

We can generate unabbreviated information with the `--long` flag. This flag will print the default unabbreviated information with the addition of a time limit field:

```
squeue -l
squeue --long
```

The `squeue` command also gives users a mean to calculate the estimated start time of a job by adding the `--start` flag to our command. This will add Slurm's estimated start time for each job to our output data.

```
squeue --user=username --start
```

Note

The start time provided by this command may be inaccurate. This is because the calculated time is based on the jobs queued or running on the system. If a job with a higher priority is queued after executing the command, your job may be delayed.

When checking the status of a job, you may want to repeatedly call the `squeue` command to check for updates. We can accomplish this by adding the `--iterate` flag to our `squeue` command. This will execute `squeue` every `n` seconds, allowing for frequent and continuous updating of queue information without the need to repeatedly call `squeue`:

```
squeue --start --iterate=n_seconds
```

Jobs state

Once a job has been submitted to a job queue, the execution will follow these states:

- **PENDING** o **PD** : The job has entered the queue but the requested resources are not yet available for it to start working, i.e. there are no free nodes.
- **RUNNING** o **R** : The job is running in the queue with the resources that have been requested.
- **COMPLETED** o **CD** : The job has been executed correctly, or at least what has been specified in the launch script.
- **COMPLETING** o **CG** : The job is in the process of being completed in a good state.
- **SUSPEND** o **S** : the execution of the job has been suspended and the resources used have been released for other work.
- **CANCELLED** o **CA** : the job has been cancelled either by the user or by the system administrators.
- **FAILED** o **F** : The job has failed.
- **NODE_FAIL** o **NF** : An error occurred with the node and the job could not be launched. By default, Slurm relaunches the job again.

Reasons for a job to be PENDING

When a job is in the **PENDING** status, the reason why it is pending execution is added and this can be:

- **(Resources)**: the job is on hold until the requested resources are available.
- **(Dependency)**: the job is dependent on another and therefore will not start until the condition of the dependency is satisfied.
- **(DependencyNeverSatisfied)**: The job is waiting for a dependency that has not been satisfied. The job will remain in this state forever, therefore, the job must be cancelled.
- **(AssocGrpCpuLimit)**: The job cannot be executed because the allocated CPU quota has been consumed.
- **(AssocGrpJobsLimit)**: The job cannot be executed because the limit of concurrent jobs that the user or account is allowed to run has been reached.

- **(ReqNodeNotAvail)**: The specified node is not available. It may be in use, it may be reserved, or it may be marked as "out of service".

Info

For more information, visit the Slurm manual on [squeue](#)

Stopping jobs with scancel

Sometimes you may need to stop a job entirely while it's running. The best way to accomplish this is with the `scancel` command. The `scancel` command allows you to cancel jobs you are running on Research Computing resources using the job's ID. The command looks like this:

```
scancel your_job-id
```

To cancel multiple jobs, you can use a comma-separated list of job IDs:

```
scancel your_job-id1, your_job-id2, your_jobid3
```

Info

For more information, visit the Slurm manual on [scancel](#)

Learning status information with sstat

The `sstat` command allows users to easily pull up status information about their currently running jobs. This includes information about CPU usage, task information, node information, resident set size (RSS), and virtual memory (VM). We can invoke the `sstat` command as such:

```
sstat --jobs=your_job-id
```

Formatting sstat output

By default, `sstat` will pull up significantly more information than what would be needed in the commands default output. To remedy this, we can use the `--format` flag to choose what we want in our output. The `format` flag takes a list of comma separated variables which specify output data:

```
sstat --jobs=your_job-id --format=var_1,var_2, ... , var_N
```

Some of these variables are listed in the table below:

| Variable | Description |
|-----------|--|
| avecpu | Average CPU time of all tasks in job. |
| averss | Average resident set size of all tasks. |
| avevmsize | Average virtual memory of all tasks in a job. |
| jobid | The id of the Job. |
| maxrss | Maximum number of bytes read by all tasks in the job. |
| maxvsize | Maximum number of bytes written by all tasks in the job. |
| ntasks | Number of tasks in a job. |

For an example, let's print out a job's average job id, cpu time, max rss, and number of tasks. We can do this by typing out the command:

```
sstat --jobs=your_job-id --format=jobid,cputime,maxrss,ntasks
```

Info

A full list of variables that specify data handled by sstat can be found with the `--helpformat` flag or by visiting the slurm page on [sstat](#).

Analyzing past jobs with sacct

The `sacct` command allows users to pull up status information about past jobs. This command is very similar to `sstat`, but is used on jobs that have been previously run on the system instead of currently running jobs. We can use a job's id...

- For all jobs executed:

```
sacct
```

- For a single job, given its ID:

```
sacct --jobs=your_job_id
```

By default, `sacct` will only pull up jobs that were run on the **current day**. We can use the `--starttime` flag to tell the command to look beyond its short-term cache of jobs.

```
sacct --jobs=your_job-id --starttime=YYYY-MM-DD
```

To see a non-abbreviated version of `sacct` output, use the `--long` flag:

```
sacct --jobs=your_job-id --starttime=YYYY-MM-DD --long
```

Formatting `sacct` output

Like `sstat`, the standard output of `sacct` may not provide the information we want. To remedy this, we can use the `--format` flag to choose what we want in our output. Similarly, the format flag is handled by a list of comma separated variables which specify output data:

```
sacct --user=your_rc-username --format=var_1,var_2, ... ,var_N
```

Some of these variables are provided below:

| Variable | Description |
|----------|---|
| account | Account the job ran under |
| avecpu | Average CPU time of all tasks in job. |
| averss | Average resident set size of all tasks in the job. |
| cputime | Formatted (Elapsed time * CPU) count used by a job or step. |
| elapsed | Jobs elapsed time formatted as DD-HH:MM:SS. |
| exitcode | The exit code returned by the job script or <code>salloc</code> . |

| Variable | Description |
|-----------------|--|
| jobid | The id of the Job. |
| jobname | The name of the Job. |
| maxdiskread | Maximum number of bytes read by all tasks in the job. |
| maxdiskwrite | Maximum number of bytes written by all tasks in the job. |
| maxrss | Maximum resident set size of all tasks in the job. |
| ncpus | Amount of allocated CPUs. |
| nnodes | The number of nodes used in a job. |
| ntasks | Number of tasks in a job. |
| priority | Slurm priority. |
| qos | Quality of service. |
| reqcpu | Required number of CPUs |
| reqmem | Required amount of memory for a job. |
| user | Username of the person who ran the job. |

As an example, suppose you want to find information about jobs that were run on March 12, 2018. You want to show information regarding the job name, the number of nodes used in the job, the number of cpus, the maxrss, and the elapsed time. Your command would look like this:

```
sacct --jobs=your_job-id --starttime=2018-03-12 --
format=jobname,nnodes,ncpus,maxrss,elapsed
```

As another example, suppose you would like to pull up information on jobs that were run on February 21, 2018. You would like information on job ID, job name, QoS, Number of Nodes used, Number of CPUs used, Maximum RSS, CPU time, Average CPU time, and elapsed time. Your command would look like this:

```
sacct --jobs=your_job-id --starttime=2018-02-21 --  
format=jobid,jobname,qos,nnodes,ncpu,maxrss,cputime,avecpu,elapsed
```

Info

A full list of variables that specify data handled by `sacct` can be found with the `--helpformat` flag or by visiting the [slurm page on sacct](#).

Controlling queued and running jobs using `scontrol`

The `scontrol` command provides users extended control of their jobs run through Slurm. This includes actions like suspending a job, holding a job from running, or pulling extensive status information on jobs.

To suspend a job that is currently running on the system, we can use `scontrol` with the `suspend` command. This will stop a running job on its current step that can be resumed at a later time. We can suspend a job by typing the command:

```
scontrol suspend job_id
```

To resume a paused job, we use `scontrol` with the `resume` command:

```
scontrol resume job_id
```

Slurm also provides a utility to hold jobs that are queued in the system. Holding a job will place the job in the lowest priority, effectively “holding” the job from being run. A job can only be held if it’s waiting on the system to be run. We use the `hold` command to place a job into a held state:

```
scontrol hold job_id
```

We can then release a held job using the `release` command:

```
scontrol release job_id
```

`scontrol` can also provide information on jobs using the `show job` command. The information provided from this command is quite extensive and detailed, so be sure to either clear your terminal window, `grep` certain information from the command, or pipe the output to a separate text file:

```
scontrol show job job_id
```


Streaming output to a textfile

```
scontrol show job job_id > outputfile.txt
```

Piping output to Grep and find lines containing the word "Time"

```
scontrol show job job_id | grep Time
```