# Batch Jobs

The real power of SLURM for handling lots of work and complex analysis pipelines in exploited through the use of batch jobs scripts.

# Job Scripts

The SLURM sbatch command is used to submit a job script to the scheduler for later execution subject to the load and the jobs priority. SLURM allows a job script to be written in any scripting language that accepts the use of # as a comment delimiter, but typically job scripts are written in a shell scripting language such as sh or bash. A simple job script using the /bin/bash shell is:

#!/bin/bash  
  
#SBATCH --job-name=SimpleJobScript  
#SBATCH --time=1:00  
#SBATCH --nodes=1  
#SBATCH --ntasks=1  
  
# Display all variables set by slurm  
env | grep "^SLURM" | sort  
  
# Print hostname job executed on.  
echo  
echo "My hostname is: $(hostname -s)"  
echo

The job will produce an output file called slurm\_JOBID.out which will contain all output from the script that was sent to [stdout](http://en.wikipedia.org/wiki/Standard_streams#Standard_output_.28stdout.29) and [stderr](http://en.wikipedia.org/wiki/Standard_streams#Standard_error_.28stderr.29). In the case of this example, that will be a listing of all the available variables set by SLURM within the jobs execution environment which are available for use within the job script as shown later in more advanced examples.

Most parameters to sbatch (many of which are shared with srun and salloc) can be included in the job script as special comments of the form

#SBATCH --parameter=value

Some of the more commonly used options are:

| Option | Description |
| --- | --- |
| #SBATCH --time=24:00:00 | Time limit, formatted as [DD-][HH]:[MM]:[SS] |
| #SBATCH --nodes=N | Number of nodes |
| #SBATCH --partition=PARTITION\_NAME | Partition(s) to run in |
| #SBATCH --ntasks=N | Number of tasks to start in the job |
| #SBATCH --cpus-per-task | Number of cores to request per task |
| #SBATCH --job-name=NAME\_OF\_JOB | Gives the job a friendly name |
| #SBATCH --gpus=N | Number of GPUs for the job |
| #SBATCH --constraints=CONSTRAINTS | A list of constraints to place on the job |

A full listing of available parameters can be found in the man pages:

| Command | Man Page Command/Link |
| --- | --- |
| sbatch | [man sbatch](http://slurm.schedmd.com/sbatch.html) |
| srun | [man srun](http://slurm.schedmd.com/srun.html) |
| salloc | [man salloc](http://slurm.schedmd.com/salloc.html) |

`

# Metaprogramming Applied to Job Scripts

A very powerful way to create jobs is to have a script or command produce the job scripts for you. The simplest example relies on --wrap which wraps your command. Let’s say there is a directory with 1000 sequence files, each file containing one sequence per line and you’d like to search all the files for a specific k-mer. Since each file is 10’s or 100s of gigabytes you’d like to do these operations in parallel. One way to do this is with a simple for loop and the sbatch feature --wrap which takes the argument to wrap as input, creates and submits a job script for you:

# Bash one-liner to run a command on all files in a directory. Caveat: this is  
# generally not best practice and for large numbers of files (10's or 100's of   
# thousands) can cause the scheduler or storage to slow or die.   
[user@login1:~]$ for file in \*; do sbatch --wrap "grep -i accatgtggtac $file"; done  
Submitted batch job 87516  
Submitted batch job 87517  
Submitted batch job 87518  
Submitted batch job 87519  
Submitted batch job 87520  
Submitted batch job 87521  
...

This would produce a slurm-JOBID.out file with your search results for each input file. However, this approach ignores the resulting I/O bottlenecks from having these execute one per core on a single node. A more efficient approach would request some number of cores per job to limit how many processes per node would be streaming data and spread the work across many nodes to speed up overall throughput. For more info and help please [wikiContactUs contact us] as we would be happy to explore ways to improve your workflow.

From this example it follows that a script can produce and submit other scripts, which can produce and submit other scripts,…ad infinitum. When writing scripts that submit jobs, careful debugging is important. It’s easy for a mistake to submit thousands of jobs very quickly. While there are limits in place to prevent going to infinity, these limits are still quite high to allow for more flexibility in using the scheduler so problems with runaway submissions can still occur.