# Submitting Arrays

Array jobs are much like single job submissions with a few exceptions:

1. Array jobs have the additional #SBATCH --array=N-M[:S][%T] argument specifying:
   * N = start
   * M = end
   * S = step size for the array ID range
   * T = Task throttle limit
2. The submitted script will be executed once for each array ID, setting these variables to appropriate values each time:
   * SLURM\_ARRAY\_JOB\_ID = The JOBID of this task.
   * SLURM\_ARRAY\_TASK\_ID = The task ID of this task.

A simple array example script is:

#!/bin/bash  
  
#SBATCH --job-name=ArrayJobScript  
#SBATCH --time=10:00  
#SBATCH --nodes=1  
#SBATCH --ntasks=1  
#SBATCH --array=0-4  
  
  
# Print some job information  
echo  
echo "My hostname is: $(hostname -s)"  
echo "I am job $SLURM\_JOB\_ID"  
echo "I am a member of the job array $SLURM\_ARRAY\_JOB\_ID"  
echo "My task ID is $SLURM\_ARRAY\_TASK\_ID"  
echo  
  
# Sleep for 1 minute  
sleep 60s

# SLURM\_ARRAY\_TASK\_ID as a parameter

Within an array job script, the ${SLURM\_ARRAY\_TASK\_ID} value can be used to change the scripts behavior for each element job in the array. For example, suppose you have an application with takes a parameter as an argument that is an integer value and you wish to test different values over a range from 1 to 10. A script for this might look like:

#!/bin/bash  
  
#SBATCH --job-name=ParameterTesting  
#SBATCH --time=10:00  
#SBATCH --nodes=1  
#SBATCH --ntasks=1  
#SBATCH --array=1-10  
  
# This is only using sleep, the sleep function could be replaced with a  
# computational model taking one parameter.  
sleep $SLURM\_ARRAY\_TASK\_ID

# Reading Multiple Arbitrary Parameters

An arbitrary number of key-value pairs can be associated with each SLURM\_ARRAY\_TASK\_ID by having the job script read a file which is indexed on SLURM\_ARRAY\_TASK\_ID and has the values in some known arrangement that can be extracted. An example of this approach is these two files, first the indexed file of values with values delimited by colons:

0:10:28:1.657:80.0023  
1:12:14:0.324:32.9  
2:1:7:9.41:16.8

Next, the array script which processes them:

#!/bin/bash  
  
#SBATCH --job-name=ArrayIndexFileExample  
#SBATCH --time=10:00  
#SBATCH --nodes=1  
#SBATCH --ntasks=1  
#SBATCH --array=0-2  
  
# Print some job information  
echo  
echo "My execution hostname is: $(hostname -s)."  
echo "I am job $SLURM\_JOB\_ID, a member of the job array $SLURM\_ARRAY\_JOB\_ID"  
echo "and my task ID is $SLURM\_ARRAY\_TASK\_ID"  
echo  
  
# We expect this file to be in the working directory, otherwise use full path.  
# Values are grabbed by selecting the line that begins with the  
# SLURM\_ARRAY\_TASK\_ID  
values=$(grep "^${SLURM\_ARRAY\_TASK\_ID}:" array\_index.txt)  
  
# Cut each individual value from it's corresponding position in the line we  
# pulled out above.  
param1=$(echo $values | cut -f 2 -d:)  
param2=$(echo $values | cut -f 3 -d:)  
param3=$(echo $values | cut -f 4 -d:)  
  
# Echo this out for illustration, but in practice we would just run the  
# command (remove the "echo" statement and the double quotes).  
echo "command -arg1=$param1 -arg2=$param2 -arg3=$param3"

When the script runs SLURM will assign each task a SLURM\_ARRAY\_TASK\_ID and the script will use that to pull out the values for each job array element. As this script demonstrates, the task id can be used to access any number of arbitrary command, arguments, scripts, files, etc, making job arrays a very powerful feature for parallelizing tasks requiring no intercommunication between each task.

# ${SLURM\_ARRAY\_TASK\_ID} to Process a List

Given a list of items, one per line in a file, it’s easy to use an array to process each item. Note that it’s also possible to adjust which lines get processed with the array specification. For instance, If you only wanted to process lines 100 - 120, simply use {{{–array=100-120}}}. The list could be as simple as a list of files prepared by ls -1 > mylistfofiles.txt or as complicated as a list of full command lines to execute in individual jobs.

An example list:

This is line 1.  
And now we have line 2.  
After 1 and 2 comes line 3.  
2 + 2 is 4.  
5 is a prime number.  
A hexagon has 6 sides.  
"Prime 7 is." (Yoda during a short stint as a math teacher.)  
There are only 10 kinds of people, those who understand octal and ... nevermind.

Next, the array script which processes the list items:

#!/bin/bash  
  
#SBATCH --job-name=ArrayLinesFromFileExample  
#SBATCH --time=10:00  
#SBATCH --nodes=1  
#SBATCH --ntasks=1  
#SBATCH --array=1-8  
  
# Print some job information  
echo  
echo "My execution hostname is: $(hostname -s)."  
echo "I am job $SLURM\_JOB\_ID, a member of the job array $SLURM\_ARRAY\_JOB\_ID"  
echo "and my task ID is $SLURM\_ARRAY\_TASK\_ID"  
echo  
  
# We expect this file to be in the working directory. Use full path if it's not.  
value=$(sed -n "${SLURM\_ARRAY\_TASK\_ID}p" array\_lines.txt)  
  
echo "command ${value}"

## Array indexing examples

--array=0-100:4

Changes step to 4 from default of 1, results in array task IDs of 0,4,8,12,16,…100

--array=2,34,5,89,1,23

Runs specific tasks from the list of array indices

## Job Array Throttling

It is often very useful to be able to limit the number of subtasks of a given job array that can run concurrently. This is known as **job array throttling** and may be accomplished with the % modifier, as shown in the following example:

--array=0-50%5

Results in 51 tasks numbered 0-50 but limits them to no more than 5 running tasks at any one time.

This may be done on the fly even after the job array has been submitted and some task elements are running. Reducing the number of concurrent tasks will NOT kill jobs, it will allow any running subtasks to complete, and will not allow further jobs to start until the throttled limit has been reached. To apply/change a throttle on a job array that is has been submitted, one may use scontrol as follows, modify MYJOBID with the job array’s base jobid (the numbers before the underscore in the squeue output):

scontrol update jobid=MYJOBID arraytaskthrottle=MYLIMIT