PURDUE UNIVERSITY | Rosen Center for Advanced Computing

SLURM quick reference sheet

SLURM

SLURM (Simple Load-Leveling Utility Resource Manager) is a job scheduler that manages a cluster's resources. You package your commands into a "job script," and SLURM runs it on powerful "compute nodes" when resources are available. This ensures fairness, efficiency, and scalability.

RCAC clusters use SLURM for:

- submitting and managing jobs on shared HPC systems
- allocating CPUs, memory, time limits, and GPUs to your analyses
- queueing and scheduling work so resources are shared efficiently
- supporting large batch workflows through job arrays

Basic SLURM job script template

```
#!/bin/sh -l

#SBATCH --account=accountname
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --partition=cpu
#SBATCH --qos=normal
#SBATCH --time=1:30:00
#SBATCH --job-name=myjobname

# load modules/environments
module load yourmodule

# run your program
your_command_here
```

Basic SLURM job script for GPU

```
#!/bin/sh -l

#SBATCH --account=accountname
#SBATCH --nodes=1
#SBATCH --gpus-per-node=1
#SBATCH --ntasks=14
#SBATCH --partition=ai
#SBATCH --qos=normal
#SBATCH --time=1:30:00
#SBATCH --job-name=myjobname

# load modules/environments
module load yourmodule
# run your program
your_command_here
```

Finding information for your SLURM script

`slist`: Lists the accounts you belong to

`showpartitions`: Available partitions with, number of nodes, memory per node, and walltime limits

`sfeatures`: Lists features of each node type

`myquota`: Reports storage allocations and usage

`sinfo`: Current availability of nodes in each partition

Directive	Purpose	Example
`--job-name`	Name of your job (appears in squeue), optional	`--job-name=trinity`
`--partition`	Which queue/partition to run in, required	`--partition=scholar-long`
`--account`	PI or project account, required	`--account=mygroup`
`--time`	Time your job can run, required	`--time=2-00:00:00`
`--ntasks`	Tasks to run; almost always 1 (default), optional	`--ntasks=1`
`--cpus-per-task`	CPU cores for your program; default 1, optional	`--cpus-per-task=16`
`--mem`	Memory required, optional (by default proportional to CPUs)	`--mem=128G`
`--nodes`	Number of nodes; almost always 1 (default), optional	`--nodes=1`
`--gres`	Generic resources, e.g., GPU, required if requesting GPUs	`--gres=gpu:1`
`--output`	File for standard output, optional	`--output=job_%j.out`
`--error`	File for standard error, optional	`--error=job_%j.err`
`--array`	Submit job arrays, optional	`--array=1-50`
`--mail-user`	Email for notifications, optional	`--mail-user=you@purdue.edu`
`--mail-type`	When to send mail (BEGIN, END, FAIL), optional	`--mail-type=END,FAIL`

Use these commands to submit, monitor, and manage your SLURM jobs on RCAC systems.

Directive	Purpose	Example
`sbatch job.sh`	submit a job script	`sbatch rnaseq-align.sh`
`sbatch -d afterok:JOBID job.sh`	run a job only after another finishes successfully	`sbatch -d afterok:1234567 next.sh`
`squeue -u $USER`	view your running and pending jobs	`squeue -u $USER`
`scontrol show job JOBID`	detailed job information	`scontrol show job 1234567`
`scontrol hold JOBID`	place a submitted job on hold	`scontrol hold 1234567`
`scontrol release JOBID`	release a previously held job	`scontrol release 1234567`
`sinteractive`	request an interactive shell on a compute node	`sinteractive -A mylab -p cpu -t 01:00:00`
`scancel JOBID`	cancel a job	`scancel 1234567`
`sacct -j JOBID`	view completed job history and resource usage	`sacct -j 1234567`

Array jobs in SLURM

Array jobs let you run the same analysis on many inputs at once. SLURM creates multiple tasks, each with its own index ($SLURM_ARRAY_TASK_ID), so you can easily process dozens or hundreds of samples in parallel using a single job script.

When to use Array jobs:

- Process many samples in parallel using a single script
- Run the same analysis with different inputs or parameters
- Split large workloads into independent tasks for faster completion
- Avoid submitting dozens or hundreds of individual jobs manually

Basic syntax

```
sbatch --array=1-100 script.sh # Tasks 1 to 100
sbatch --array=1-100:10 script.sh # Tasks 1,11,21,...,91
sbatch --array=1-100%20 script.sh # Max 20 tasks running simultaneously
sbatch --array=1,5,10,15 script.sh # Specific task IDs
scancel <JOBID>_<TASKID> # Cancel specific task
scancel <JOBID>_[10-20] # Cancel range
scancel <JOBID> # Cancel entire array
```

samples.txt

```
sample1
sample2
sample3
sample4
sample5
sample6
...
...
```

Example 1: mapping each fastq sample

```
#!/bin/bash
#SBATCH --nodes=1 --cpus-per-task=8
#SBATCH --account=accountname --partition=cpu
#SBATCH --array=1-3
#SBATCH --time=4:00:00
#SBATCH --output=logs/sample_%A_%a.out

# Get sample name from line N of file
SAMPLE=$(sed -n "${SLURM_ARRAY_TASK_ID}p" samples.txt)

# Run analysis
bwa mem -t 8 ref.fa ${SAMPLE}_R1.fq ${SAMPLE}_R2.fq | \
  samtools sort -@ 8 -o ${SAMPLE}.bam
```

Example 2: process each chromosome

```
#!/bin/bash
#SBATCH --nodes=1 --cpus-per-task=4
#SBATCH --account=accountname --partition=cpu
#SBATCH --array=1-22
#SBATCH --time=2:00:00
#SBATCH --output=logs/sample_%A_%a.out

# chromosomes 1 thru 22
CHR=chr"${SLURM_ARRAY_TASK_ID}"

# Run analysis
bcftools mpileup -r ${CHR} -f ref.fa input.bam | \
bcftools call -mv -Oz -o variants_${CHR}.vcf.gz
```

Example 3: process each fastq sample (generic)

```
#!/bin/bash
#SBATCH --nodes=1 --cpus-per-task=8 --time=4:00:00
#SBATCH --account=accountname --partition=cpu
#SBATCH --array=1-$(wc -l < samples.txt)3
#SBATCH --output=logs/sample_%A_%a.out

# Get sample name from line N of file
SAMPLE=$(sed -n "${SLURM_ARRAY_TASK_ID}p" samples.txt)

# Run analysis
command -R1 ${SAMPLE}_R1.fq -R2 ${SAMPLE}_R2.fq
```

Example 4: parameter sweep (kmers)

```
#!/bin/bash
#SBATCH --nodes=1 --cpus-per-task=4
#SBATCH --account=accountname --partition=cpu
#SBATCH --array=1-10 --time=2:00:00
#SBATCH --output=logs/sample_%A_%a.out

# Test different k-mer sizes for assembly
KMER=$((21 + SLURM_ARRAY_TASK_ID * 10))

# Run assembly
spades.py -k ${KMER} -1 R1.fq -2 R2.fq \
          -o asm_k${KMER}
```

Variable	Meaning	Typical use
$SLURM_JOBID	Unique ID of the job	Naming output files, debugging, `sacct -j`
$SLURM_JOB_NAME	Name of the job	Logging, building filenames
$SLURM_SUBMIT_DIR	Directory where the job was submitted	Ensuring outputs go back to the right place
$SLURM_CPUS_PER_TASK	Number of CPUs allocated to the task	Setting tool threads (`-t, --threads`)
$SLURM_ARRAY_JOB_ID	Job ID shared by the array	Grouping output for all array tasks
$SLURM_ARRAY_TASK_ID	Index of the current array sub-task	Selecting the correct sample or file
$SLURM_ARRAY_TASK_MIN	First index of the array	Useful for bounds checks or logging
$SLURM_ARRAY_TASK_MAX	Last index of the array	Automation, loops, verification
$SLURM_ARRAY_TASK_STEP	Step size of the array	Parameter sweeps, non-continuous arrays