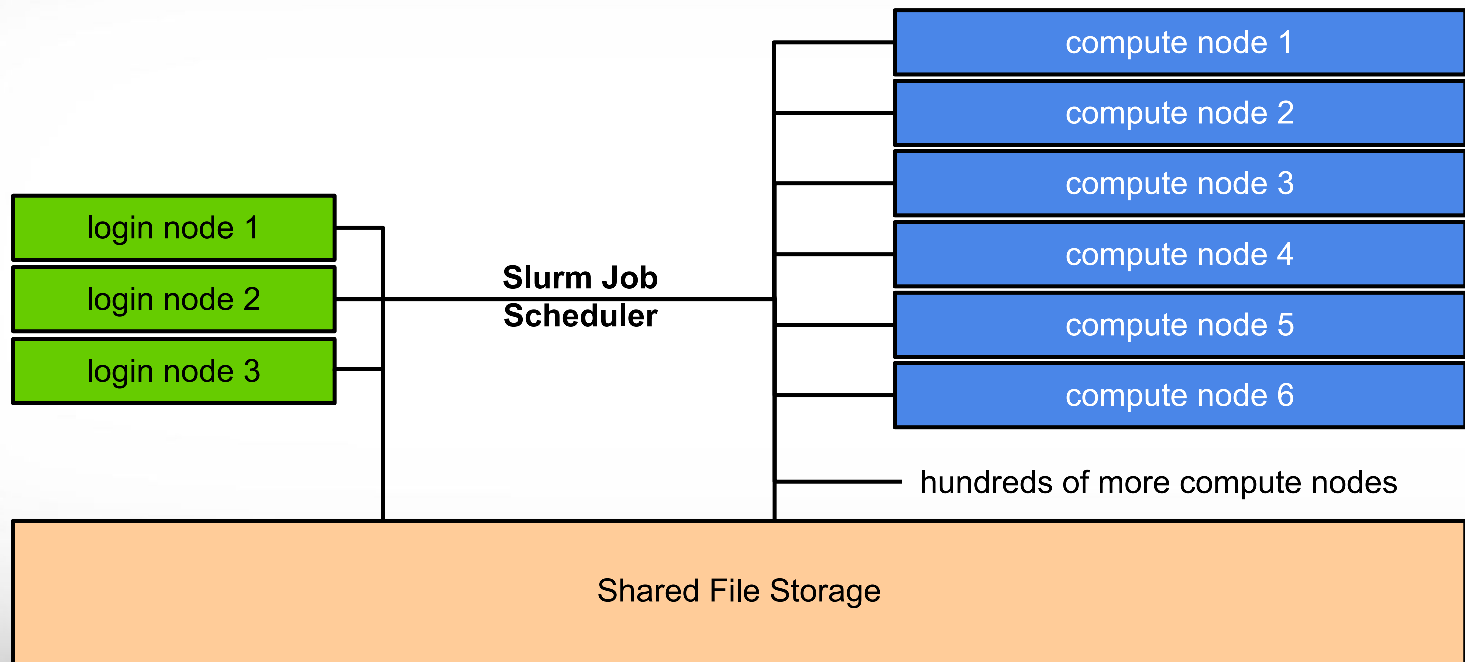
Slurm Job Scheduling

Job Scheduling

- SBATCH Parameters
- Single node jobs
  - single-core
  - multi-core
- Multi-node jobs
  - MPI jobs
  - TAMULauncher
  - array jobs
- Monitoring job resource usage
  - at runtime
  - after job completion
  - job debugging

HPC Diagram



Terra Service Unit Calculations

- For the Terra 64GB memory nodes (56GB available), you are charged Service Units (SUs) based on one of the following values whichever is greater.
  - 1 SU per CPU per hour or 1 SU per 2GB of requested memory per hour

Number of Cores	Total Memory per node (GB)	Hours	SUs charged
1	2	1	1
1	3	1	2
1	56	1	28
28	56	1	28

Single vs Multi-Core Jobs

- When to use single-core jobs
  - The software being used only supports commands utilizing a single-core
- When to use multi-core jobs
  - If the software supports multiple-cores (--threads, --cpus, …) then configure the job script and command options to utilize all CPUs on a compute node to get the job done faster unless the software specifically recommends a limited number of cores
    - 28 CPUs per compute node on Terra 64GB memory nodes
    - 56GB of available memory per compute node on Terra 64GB memory nodes
  - Can group multiple single-core commands into a multi-core job using TAMULauncher on one or multiple nodes
- A compute node is one computer unit of an HPC cluster. A cluster can contain a few to thousands of compute nodes. These are often referred to as just nodes.
  - number of nodes for a job can be specified with the --nodes parameter

Submitting Slurm Jobs

- Jobs can be submitted using a job script or directly on the command line
- A job script is a text file of Unix commands with #SBATCH parameters
- #SBATCH parameters provide resource configuration request values
  - time, memory, nodes, cpus, output files, ...
- Submit the job using sbatch command with the job script name
  - your job script provides a record of commands used for an analysis
  - **`sbatch job_script.sh`**
- Submit command on the command line by specifying all necessary parameters
  - must rely on your bash history to see #SBATCH parameters used which is not reliable
  - **`sbatch -t 01:00:00 -n 1 -J myjob --mem 2G -o stdout.%j commands.sh`**

slurm.schedmd.com/sbatch.html

Slurm Job Script Parameters

```
#!/bin/bash
#SBATCH --export=NONE              # do not export current env to the job
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00          # request 1 day; Format: days-hours:minutes:seconds
#SBATCH --nodes=1                  # request 1 node (optional since default=1)
#SBATCH --ntasks-per-node=1        # request 1 task (command) per node
#SBATCH --cpus-per-task=1          # request 1 cpu (core, thread) per task
#SBATCH --mem=2G                   # request 2GB total memory per node
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

# load software modules
# run commands
```

- Always include the first three lines exactly as they are
- Slurm job parameters begin with **`#SBATCH`** and can add comments as above
- Name the job script whatever you like
  - run_program_project.sh
  - my_job_script.job
  - my_job_script.sbatch
  - my_job_script.txt

Slurm Parameters: nodes, tasks, cpus

- --nodes
  - number of nodes to use where a node is one computer unit of many in an HPC cluster
    - `--nodes=1                # request 1 node (optional since default=1)`
  - used for multi-node jobs
    - `--nodes=10`
  - if number of cpus per node is not specified then defaults to 1 cpu
  - defaults to 1 node if --nodes not used & can use together with --ntasks-per-node and --cpus-per-task
  - do not use --nodes with --array
- --ntasks
  - a task can be considered a command such as blastn, bwa, script.py, etc.
  - `--ntasks=28              # total tasks across all nodes per job`
  - when using --ntasks without --nodes, the values for --ntasks-per-node and --cpus-per-task will default to 1 node, 1 task per node and 1 cpu per task
- --ntasks-per-node
  - use together with --cpus-per-task
  - `--ntasks-per-node=1`
- --cpus-per-task
  - number of CPUs (cores) for each task (command)   Example: `magicblast -num_threads 28`
  - `--cpus-per-task=28`

Additional Slurm Parameters

- --job-name
  - set the job name, keep it short and concise without spaces
  - `--job-name=myjob`
- --time
  - max runtime for job; format: days-hours:minutes:seconds (days- is optional)
  - `--time=24:00:00          # max runtime 24 hours`
  - `--time=7-00:00:00        # max runtime 7 days`
- --mem
  - total memory for each node (not for each CPU core as with Ada)
  - `--mem=56G                # request 56GB total memory (max available on 64gb nodes)`
- --partition
  - specify a partition (queue) to use
  - partition is automatically assigned to short, medium, long and gpu (when using --gres=gpu)
  - only need to specify --partition parameter to use vnc, xlong, special, knl
  - `--partition=xlong        # select xlong partion`
- --output
  - save all stdout to a specified file
  - `--output=stdout.%j       # saves stdout to a file named stdout.JOBID`
- --error
  - save all stderr to a specified file
  - `--error=stderr.%j        # saves stderr to a file named stderr.JOBID`
  - use just --output to save stdout and stderr to the same output file: `--output=output.%j.log`

Optional Slurm Parameters

- --gres
  - used to request 1 or 2 GPUs; automatically assigns --partition=gpu
  - `--gres=gpu:1            # request 1 GPU; use :2 for two GPUs`
- --account
  - specify which HPRC account to use; see your accounts with the `myproject` command
  - `--account=ACCOUNTNUMBER`
  - default account from `myproject` output is used if not specified
- --mail-user
  - send email to user
  - `--mail-user=myemail@tamu.edu`
- --mail-type
  - send email per job event: BEGIN, END, FAIL, ALL
  - `--mail-type=ALL`
- --dependency
  - schedule a job to start after a previous job successfully completes
  - `--dependency=afterok:JOBID`

Single-Node Jobs

Single-Node Single-Core Job Scripts

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1          # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=2G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module load SPAdes/3.13.0-foss-2018b
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 1
```

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1          # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=4G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module load SPAdes/3.13.0-foss-2018
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 1
```

specify number of threads to match SBATCH parameters

Example 1		Example 2
1	Total CPUs requested	1
1	CPUs per node	1
2	total requested GB memory	4
1	SUs to start job	2

Slurm Parameter: --ntasks

```
#!/bin/bash
#SBATCH --export=NONE              # do not export current env to the job
#SBATCH --job-name=myjob           # job name
#SBATCH --time=1:00:00             # set the wall clock limit to 1 hour
#SBATCH --ntasks=1                 # request 1 task (command) per node
#SBATCH --mem=2G                   # request 2GB of memory per node
#SBATCH --output=stdout.%j         # create a file for stdout
#SBATCH --error=stderr.%j          # create a file for stderr
```

When only --ntasks is used, the --ntasks-per-node value will be automatically set to match --ntasks and defaults to --cpus-per-task=1 and --nodes=1

- --ntasks=1
  - NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1
- --ntasks=28
  - NumNodes=1 NumCPUs=28 NumTasks=28 CPUs/Task=1

Requesting all CPUs and Available Memory on Terra Compute Nodes

28 cores, 64 GB nodes (256 nodes available)

```
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=56G
```

28 cores, 128 GB memory GPU nodes (48 nodes avail)

```
#SBATCH --gres=gpu:2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=112G
```

68 cores, 96 GB KNL nodes (8 nodes available)

```
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=68
#SBATCH --mem=86G
```

72 cores, 96 GB memory nodes (8 nodes available)

```
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=72
#SBATCH --mem=86G
```

Single-Node Multi-Core Job Scripts

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1     # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=14
#SBATCH --mem=56G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module load SPAdes/3.13.0-foss-2018b
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 14
```

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=1     # optional since default=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=56G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module load SPAdes/3.13.0-foss-2018b
spades.py -1 s1_R1 -2 s1_R2 -o outdir --threads 28
```

specify number of threads to match SBATCH parameters

14	Total CPUs requested	28
14	CPUs per node	28
56	total requested GB memory	56
672	SUs to start job	672

It is best to request all cores and all memory if using the entire node

GPU Jobs

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=my_gpu_job
#SBATCH --time=1-00:00:00
#SBATCH --ntasks=1
#SBATCH --gres=gpu:1                    # request 1 GPU; use :2 for two GPUs
#SBATCH --mem=2G                        # can request max of 112GB on GPU nodes
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

# load required modules
module load intel/2017A
module load CUDA/9.2.148.1

# run your gpu command
my_gpu_command
```

- gpu partition (--partition=gpu) will be automatically applied when using --gres=gpu:n
- 28 SUs will be charged for using GPU nodes whether you specify --ntasks=1 or --ntasks=28

Multi-Node Jobs

Slurm Parameters: --nodes --ntasks-per-node

```
#!/bin/bash
#SBATCH --export=NONE              # do not export current env to the job
#SBATCH --job-name=myjob           # job name
#SBATCH --time=1:00:00             # set the wall clock limit to 1 hour
#SBATCH --nodes=2                  # request 2 nodes
#SBATCH --ntasks-per-node=1        # request 1 task (command) per node
#SBATCH --cpus-per-task=28         # request 28 cores per task
#SBATCH --mem=56G                  # request 56GB of memory per node
#SBATCH --output=stdout.%j         # create a file for stdout
#SBATCH --error=stderr.%j          # create a file for stderr
```

It is easier to scale jobs by using --nodes with --ntasks-per-node instead of with --ntasks.
If you use --nodes with --ntasks, you need to calculate total CPUs for all nodes as the --ntasks value

- --nodes=2 --ntasks-per-node=28
  - NumNodes=2-2 NumCPUs=56 NumTasks=56 CPUs/Task=1 mem=112G
- --nodes=1 --ntasks=28
  - NumNodes=1 NumCPUs=28 NumTasks=28 CPUs/Task=1 mem=56G
- --nodes=2 --ntasks=56
  - NumNodes=2 NumCPUs=56 NumTasks=56 CPUs/Task=1 mem=112G
- **when --nodes is > 1, avoid using --ntasks:** --nodes=2 --ntasks=28
  - will allocate 1 core on one node and 27 cores on a second node

MPI Multi-Node Multi-Core Job Script: Example 1

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=moose
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=56G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module load MOOSE/moose-dev-gcc-ompi
mpirun -np 280 -npernode 28 /path/to/moose-opt -i moose.i
```

280	Total CPUs requested
28	CPUs per node
56	total requested GB memory per node
6720	SUs to start job (nodes * cpus * hours)

MPI Multi-Node Multi-Core Job Script: Example 2

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=abyss-pe
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=56G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module load ABySS/2.0.2-foss-2018a
abyss-pe np=280 j=28 lib='lib1' lib1='sample1_R1 sample1_R2'
```

280	Total CPUs requested
28	CPUs per node
56	total requested GB memory per node
6720	SUs to start job (nodes * cpus * hours)

TAMULauncher

- hprc.tamu.edu/wiki/SW:tamulauncher
- Use when you have hundreds or thousands of commands to run each utilizing a single-core or a few cores
  - tamulauncher keeps track of which commands completed successfully
    - to see the list of completed commands
      - `tamulauncher --status commands_file.txt`
    - if time runs out, then tamulauncher can be restarted and it will know which was the last successfully completed command
  - submit tamulauncher as a batch job within your job script
  - can run tamulauncher interactively on login node; limited to 8 cores
  - you can check the --status on the command line from the working directory
- run a single command of your thousands to make sure the command is correct and to get an estimate of resource usage (CPUs, memory, time)
- request all cores and memory on the compute node(s) and configure your commands to use all available cores

TAMULauncher Multi-Node Single-Core Commands

commands.txt (300 lines for example)

```
spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 1
spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 1
spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 1
spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 1
spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 1
spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 1
spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 1
spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 1
spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 1
spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 1
spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 1
spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 1
spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 1
spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 1
spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 1
spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 1
spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 1
spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 1
spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 1
spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 1
spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 1
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 1
```

run_spades_tamulauncher.sh

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=28
#SBATCH --cpus-per-task=1
#SBATCH --mem=56G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j

module load SPAdes/3.13.0-foss-2018b

tamulauncher commands.txt
```

run 28 spades.py commands per node with each command using 1 core. Requesting all 28 cores reserves entire node for your job

- run 28 single-core commands per node; useful when each command requires < 2GB memory
- create a commands file (named whatever you want) to go with the the job script
- load the software module in the job script not the commands file

TAMULauncher Multi-Node Multi-Core Commands

commands.txt (300 lines for example)

```
spades.py -1 s1_R1.fastq.gz -2 s1_R2.fastq.gz -o s1_out --threads 4
spades.py -1 s2_R1.fastq.gz -2 s2_R2.fastq.gz -o s2_out --threads 4
spades.py -1 s3_R1.fastq.gz -2 s3_R2.fastq.gz -o s3_out --threads 4
spades.py -1 s4_R1.fastq.gz -2 s4_R2.fastq.gz -o s4_out --threads 4
spades.py -1 s5_R1.fastq.gz -2 s5_R2.fastq.gz -o s5_out --threads 4
spades.py -1 s6_R1.fastq.gz -2 s6_R2.fastq.gz -o s6_out --threads 4
spades.py -1 s7_R1.fastq.gz -2 s7_R2.fastq.gz -o s7_out --threads 4
spades.py -1 s8_R1.fastq.gz -2 s8_R2.fastq.gz -o s8_out --threads 4
spades.py -1 s9_R1.fastq.gz -2 s9_R2.fastq.gz -o s9_out --threads 4
spades.py -1 s10_R1.fastq.gz -2 s10_R2.fastq.gz -o s10_out --threads 4
spades.py -1 s11_R1.fastq.gz -2 s11_R2.fastq.gz -o s11_out --threads 4
spades.py -1 s12_R1.fastq.gz -2 s12_R2.fastq.gz -o s12_out --threads 4
spades.py -1 s13_R1.fastq.gz -2 s13_R2.fastq.gz -o s13_out --threads 4
spades.py -1 s14_R1.fastq.gz -2 s14_R2.fastq.gz -o s14_out --threads 4
spades.py -1 s15_R1.fastq.gz -2 s15_R2.fastq.gz -o s15_out --threads 4
spades.py -1 s16_R1.fastq.gz -2 s16_R2.fastq.gz -o s16_out --threads 4
spades.py -1 s17_R1.fastq.gz -2 s17_R2.fastq.gz -o s17_out --threads 4
spades.py -1 s18_R1.fastq.gz -2 s18_R2.fastq.gz -o s18_out --threads 4
spades.py -1 s19_R1.fastq.gz -2 s19_R2.fastq.gz -o s19_out --threads 4
spades.py -1 s20_R1.fastq.gz -2 s20_R2.fastq.gz -o s20_out --threads 4
spades.py -1 s21_R1.fastq.gz -2 s21_R2.fastq.gz -o s21_out --threads 4
spades.py -1 s22_R1.fastq.gz -2 s22_R2.fastq.gz -o s22_out --threads 4
```

run_spades_tamulauncher.sh

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=spades
#SBATCH --time=1-00:00:00
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=7
#SBATCH --cpus-per-task=4
#SBATCH --mem=56G
#SBATCH --output=stdout.%j
#SBATCH --error=stderr.%j


module load SPAdes/3.13.0-foss-2018b

tamulauncher commands.txt
```

run 7 spades.py commands per node with each command using 4 cores. Requesting all 28 cores reserves entire node for your job

- useful when each command requires more than 2GB but less than all available memory
- use OMP_NUM_THREADS if needed when running fewer commands than requested cores
  - add on the line before the tamulauncher command
  - `export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK`

Making a TAMULauncher Commands File Part 1

Input files are two files per sample and named: s1_R1.fastq.gz s1_R2.fastq.gz

Run this command to create the example files:

```
mkdir seqs && touch seqs/s{1..40}_R{1,2}.fastq.gz
```

Run the following commands to get familiar with useful shell commands for creating and manipulating variables

```
file=seqs/s1_R1.fastq.gz            # create variable named file
echo $file                          # show contents of variable
basename $file                      # strip off path of variable
sample=$(basename $file)            # create variable named sample
echo $sample                        # show contents of variable
echo ${sample/_R1.fastq.gz}         # strip off _R1.fastq.gz
echo ${sample/R1/R2}                # substitute text R1 with R2
```

Making a TAMULauncher Commands File Part 2

Input files are two files per sample and named: s1_R1.fastq.gz   s1_R2.fastq.gz

Run the following commands to loop through all R1 files in the reads directory and create the commands.txt
Use just the R1 files because we only need to capture the sample names once.

```
for file in seqs/*_R1.*gz
do
read1=$file
read2=${read1/_R1./_R2.}
sample=$(basename ${read1/_R1.fastq.gz})
echo spades.py -1 $read1 -2 $read2 -o ${sample}_out --threads 1
done > commands.txt
```

Match as much as possible to avoid matching sample names

Other Useful Unix Commands

```
${variable#*SubStr}     # will drop beginning of variable value up to first occurrence of 'SubStr'
${variable##*SubStr}    # will drop beginning of variable value up to last occurrence of 'SubStr'
${variable%SubStr*}     # will drop part of variable value from last occurrence of 'SubStr' to the end
${variable%%SubStr*}    # will drop part of variable value from first occurrence of 'SubStr' to the end
```

These are useful if the part of the filename for each sample that needs to be removed is not the same.

s1_S1_R1.fastq.gz **s2**_S2_R1.fastq.gz **s3**_S3_R1.fastq.gz

want to remove this part from each file name

Make a new directory and create a new set of files for this exercise.

```
mkdir seqs2
for i in {1..10}; do touch seqs2/s${i}_S${i}_R{1,2}.fastq.gz; done
```

Unix Command	Output
`file=seqs2/s1_S1_R1.fastq.gz`	
`echo $file`	`seqs2/s1_S1_R1.fastq.gz`
`echo ${file%_S*}`	`seqs2/s1`
`basename ${file%_S*}`	`s1`
`sample=$(basename ${file%_S*})`	
`echo $sample`	`s1`

Slurm Job Array Parameters and Runtime Environment Variables

- Array jobs are good to use when you have multiple samples each of which can utilize an entire compute node running software that supports multiple threads but does not support MPI
  - `--array=0-23       # job array indexes 0-23`
  - `--array=1-24       # job array indexes 1-24`
  - `--array=1,3,5,7    # job array indexes 1,3,5,7`
  - `--array=1-7:2      # job array indexes 1 to 7 with a step size of 2`
  - `do not use --nodes with --array`
- Use the index value to select which commands to run either from a text file of commands or as part of the input file name or parameter value
  - `$SLURM_ARRAY_TASK_ID` is the array index value
- stdout and stderr files can be saved per index value
  - `--output=stdout-%A_%a`
  - `--error=stderr-%A_%a`
- Limit the number of simultaneously running tasks
  - can help prevent reaching file and disk quotas due to many intermediate and temporary files
  - as one job completes another array index is run on the available node
  - `--array=1-40%5     # job array with indexes 1-40; max of 5 running nodes`

Slurm Job Array Example 1

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=bwa_array
#SBATCH --time=1-00:00:00
#SBATCH --array=1-40%5     # job array with indexes 1-40; max of 5 running nodes
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=56G
#SBATCH --output=stdout.%A_%a
#SBATCH --error=stderr.%A_%a

module load BWA/0.7.17-intel-2018b
# get a line from commands.txt file into a variable named command
command=$(sed -n ${SLURM_ARRAY_TASK_ID}p commands.txt)
$command
```

- The sed command will print a specified line number from commands.txt based on the **`$SLURM_ARRAY_TASK_ID`**
- The number of lines in your commands.txt file should be the same as the number of array indexes
- Can use %5 to limit the array to a maximum of 5 nodes used simultaneously but you need enough SUs to cover entire number of array indexes in order to submit the job. May be useful to prevent creating too many temporary files.
- There are other ways to use $SLURM_ARRAY_TASK_ID but this example is useful because it has a file of all commands used in each array index

Slurm Job Array Example 2

```
#!/bin/bash
#SBATCH --export=NONE
#SBATCH --job-name=bwa_array
#SBATCH --time=1-00:00:00
#SBATCH --array=1-40          # run all 40 array indexes simultaneously using 40 nodes
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=56G
#SBATCH --output=stdout.%A_%a
#SBATCH --error=stderr.%A_%a

module load BWA/0.7.17-intel-2018b

bwa mem -M -t 28 -R '@RG\tID:\tLB:pe\tSM:DR34\tPL:ILLUMINA' genome.fasta \
sample_${SLURM_ARRAY_TASK_ID}_R1.fastq.gz  sample_${SLURM_ARRAY_TASK_ID}_R2.fastq.gz \
| samtools view -h -Sb - | samtools sort -o sample_${SLURM_ARRAY_TASK_ID}.out.bam \
-m 2G -@ 1 -T $TMPDIR/tmp4sort${SLURM_ARRAY_TASK_ID} -
```

- Can use `$SLURM_ARRAY_TASK_ID` if your commands only differ by a number in the file names
- The `$SLURM_ARRAY_TASK_ID` variable will be assigned the array index from 1 to 40 in this example

Useful Slurm Runtime Environment Variables

- $TMPDIR
  - this is a temporary local disk space (~848GB) created at runtime and is deleted when the job completes
  - the directory is mounted on the compute node and files created in $TMPDIR do not count against your file and disk quotas
  - **`samtools sort -T $TMPDIR/sort`**
- $SLURM_CPUS_PER_TASK
  - returns how many CPU cores were allocated on this node
  - can be used in your command to match requested #SBATCH cpus
    - **`#SBATCH --cpus-per-task=28`**
    - **`samtools sort --threads $SLURM_CPUS_PER_TASK`**
- $SLURM_ARRAY_TASK_ID
  - can be used to select or run one of many commands when using a job array
- $SLURM_JOB_NAME
  - populated by the --job-name parameter
  - **`#SBATCH --job-name=bwa_array`**
- $SLURM_JOB_NODELIST
  - can be used to get the list of nodes assigned at runtime
- $SLURM_JOBID
  - can be used to capture JOBID at runtime

Useful Unix Environment Variables

- Type `env` to see all Unix environment variables for your login session
- $USER
  - This will be automatically populated with your NetID
    - `echo $USER`
- $SCRATCH
  - You can use this to change to your /scratch/user/netid directory
    - `cd $SCRATCH`
- $OMP_NUM_THREADS
  - useful when software uses OpenMP for multithreading; default is 1
    - `export OMP_NUM_THREADS=28`
- $PWD
  - contains the full path of the current working directory

Monitoring Job Resource Usage

Submit a Slurm Job

- Submit a job
  - `sbatch my_job_file.sh`
- See status and JOBID of all your submitted jobs
  - `squeue -u $USER`

```
Thu Oct 17 11:32:44 2019
JOBID      PARTITION  NAME      USER       STATE      TIME       TIME_LIMIT  NODES NODELIST(REASON)
3279434    short,med  bwa       netid      PENDING         0:00   1-00:00:00    1 (Priority)
```

- Cancel (kill) a queued or running job using JOBID
  - `scancel JOBID`
- Get an estimate of when your pending job will start.
  - It is just an estimate based on all currently scheduled jobs running to the maximum specified runtime.
  - Can be useful for queues: gpu, vnc, xlong, special, knl
  - It will usually start before the estimated time.
    - `squeue --start --job JOBID`

Monitor a Running Job

- See status and JOBID of all your submitted jobs
  - `squeue -u $USER`
- See summary of a running job (list node utilization)
  - `lnu JOBID`

```
JOBID       NAME      USER      PARTITION      NODES  CPUS  STATE       TIME   TIME_LEFT   START_TIME
3276433     bwa       netid     short          1      1     RUNNING     5:33   54:27       2019-10-15T10:52:06

HOSTNAMES   CPU_LOAD  FREE_MEM  MEMORY    CPUS(A/I/O/T)
tnxt-0424   1.64      54407     57344     1/27/0/28
```

For the CPUS values:

- `A` = Active (in use by running jobs)
- `I` = Idle (available for jobs)
- `O` = Offline (unavailable for jobs)
- `T` = Total

- See CPU and memory usage of all your running jobs
  - `pestat -u $USER`
  - stats for pestat are updated every 3 minutes
  - can use with watch command to run pestat every 2 seconds
    - `watch pestat -u $USER`

```
Hostname       Partition   Node  Num_CPU  CPUload   Memsize  Freemem  Joblist
                           State Use/Tot            (MB)     (MB)     JobId User ...
tnxt-0703      xlong       alloc  28  28   16.23*   57344    55506    565849 net
tnxt-0704      xlong       alloc  28  28   19.60*   57344    53408    565849 net
tnxt-0705      xlong       alloc  28  28   19.56*   57344    53408    565849 net
tnxt-0701      xlong       alloc  28  28   27.50    57344    13206    565851 net
tnxt-0709      xlong       alloc  28  28   26.47*   57344    53408    565855 net
```

Low CPU load utilization highlighted in **Red**
Good CPU load utilization highlighted in **Purple**
Ideal CPU load utilization displayed in White
(Freemem should also be noted)

Monitor a Running Job Array

- List node utilization of a running job
  - **`lnu JOBID`**
  - CPU_LOAD of 28.00 means 100% node utilization
    - all 28 of 28 total cores are at 100% utilization
  - **`lnu 3278982`**

```
JOBID    NAME          USER       PARTITION           NODES  CPUS  STATE    TIME     TIME_LEFT      START_TIME

3279004  bwa_array     netid   short                  5      140   RUNNING  0:18       59:42        2019-10-17T12:22:38

HOSTNAMES     CPU_LOAD    FREE_MEM    MEMORY      CPUS(A/I/O/T)
tnxt-0327     27.89         51372          57344          28/0/0/28
tnxt-0467     27.97         48965          57344          28/0/0/28
tnxt-0609     8.83          53920          57344          28/0/0/28
tnxt-0635     0.26          56812          57344          28/0/0/28
tnxt-0711     0.58          58701          57344          28/0/0/28
```

For the CPUS values:
- `A` = Active (in use by running jobs)
- `I` = Idle (available for jobs)
- `O` = Offline (unavailable for jobs)
- `T` = Total

  - this appears to be two jobs because it is a job array
    - **`#SBATCH --array=1-5`**

Monitor a Running Job

- See lots of info about your running or recently completed (~10 minutes) job
  - `scontrol show job JOBID`
- gpu jobs are charged 28 SUs per job regardless of whether 1 or all CPUs are selected
- can add this command at the end of your job script to capture job info into the stdout file
  - `scontrol show job $SLURM_JOBID`

```
JobId=3279341 JobName=bwa
   UserId=netid(16555) GroupId=netid(16555) MCS_label=N/A
   Priority=93521 Nice=0 Account=122787215270 QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:10 TimeLimit=01:00:00 TimeMin=N/A
   SubmitTime=2019-10-17T15:33:26 EligibleTime=2019-10-17T15:33:26
   AccrueTime=2019-10-17T15:33:26
   StartTime=2019-10-17T15:33:27 EndTime=2019-10-17T16:33:27 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   LastSchedEval=2019-10-17T15:33:27
   Partition=short AllocNode:Sid=tlogin-0502:47498
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=tnxt-0469
   BatchHost=tnxt-0469
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=56G,node=1,billing=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=56G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/scratch/user/netid/myjobdir/run_my_job_terra.sh
   WorkDir=/scratch/user/netid/myjobdir/bwa_tamulauncher
   Comment=(from job_submit) your job is charged as below
             Project Account: 12278721234
             Account Balance: 199872.875303
             Requested SUs:   28
   StdErr=/scratch/user/netid/myjobdir/stderr.3279341
   StdIn=/dev/null
   StdOut=/scratch/user/netid/myjobdir/stdout.3279341
```

See Completed Job Stats

- `sacct -j JOBID`
  - will only give you the following headers

`JobID JobName User NCPUS NNodes State Elapsed CPUTime Start End ReqMem NodeList`

- Use the following command to see MaxRSS (max memory used)

```
sacct --format user,jobid,jobname,partition,nodelist,maxrss,maxdiskwrite,state,exitcode,alloctres%36 -j JOBID
```

```
 User         JobID    JobName     Partition  NodeList  MaxRSS       MaxDiskWrite    State      ExitCode                     AllocTRES
------ ------------ ---------- ---------- ---------- ---------- ------------       ---------- -------- ------------------------------------
netid  3243578        spades_pe  medium     tnxt-0459                              COMPLETED       0:0    billing=28,cpu=28,mem=56G,node=1
       3243578.bat+   batch                 tnxt-0459  36547764K   26729.78M       COMPLETED       0:0              cpu=28,mem=56G,node=1
       3243578.ext+   extern                tnxt-0459       908K           0       COMPLETED       0:0    billing=28,cpu=28,mem=56G,node=1
```

36.5 GB max memory used

- There are three lines for this job: 1 job and 2 steps
  - spades_pe
  - batch (JobID.batch)
  - extern (JobID.extern)

you can set an alias in your .bashrc for the sacct --format command

See All Your Terra Job for Current Fiscal Year

- `myproject -j all`
  - ProjectAccount
  - JobID
  - JobArrayIndex
  - SubmitTime
  - StartTime
  - EndTime
  - Walltime
  - TotalSlots
  - UsedSUs
  - Total Jobs
  - Total Usage (SUs)

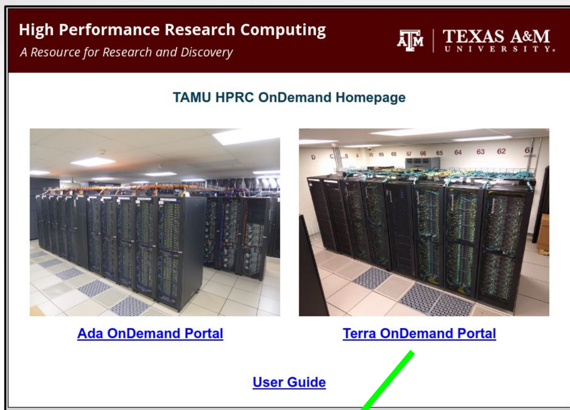
Debugging Slurm Jobs

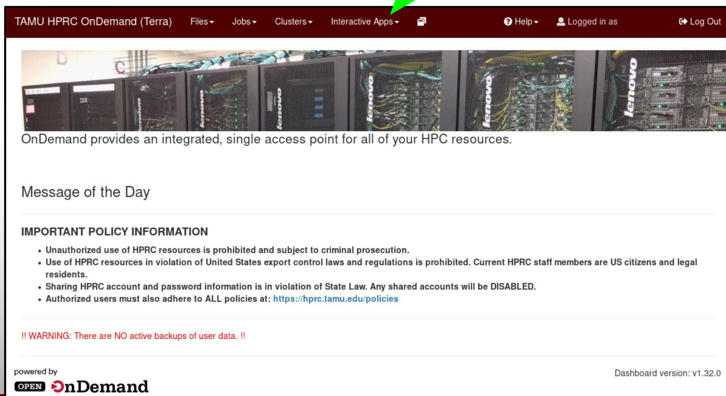
- If job was not scheduled, check your HPRC account to see if you have enough SUs
  - **`myproject`**
- Look for an out of memory error message; could occur in only one index of a job array

```
slurmstepd: error: Exceeded job memory limit at some point.
```

  - Make the necessary adjustments to SBATCH memory parameters in your job script and resubmit the job
- If you see an 'Out of disk space' error
  - check your file and disk quotas using the showquota command
    - **`showquota`**
  - reduce the number of files you have generated
    - delete any unnecessary or temporary files
    - use **`$TMPDIR`** in your command if software supports a temporary directory
    - create a .tar.gz package of completed projects to free up disk space
  - request an increase in file and/or disk quota for your project

portal.hprc.tamu.edu





The HPRC portal allows users to do the following

- Browse files on the Ada (Curie), Terra filesystem
- Access the Ada, Terra, Curie Unix command line
  - no SUs charged for using command line
  - runs on login node; please limit to 8 cores
- Launch jobs
  - SUs charged when launching jobs
- Compose job scripts
- Launch interactive GUI apps (SUs charged)
- Ada
  - ANSYS Workbench
  - Abaqus/CAE
  - IGV
  - LS-PREPOST
  - MATLAB
  - ParaView
  - VNC
  - Galaxy
  - JupyterLab
  - Jupyter Notebook
  - RStudio
- Terra
  - ANSYS Workbench
  - Abaqus/CAE
  - JBrowse
  - LS-PREPOST
  - MATLAB
  - ParaView
  - VNC
  - ImageJ and other imaging sw
  - Jupyter Notebook, JupyterLab
  - BEAUti, Tracer, FigTree
  - RStudio
- Monitor and stop running jobs and interactive sessions

For More HPRC Help...

Website: hprc.tamu.edu
Email: help@hprc.tamu.edu
Telephone: (979) 845-0219
Visit us in person: Henderson Hall, Room 114A
(best to email or call in advance and make an appointment)

Help us, help you -- we need more info

- Which Cluster
- UserID/NetID
- Job id(s) if any
- Location of your jobfile, input/output files
- Application used if any
- Module(s) loaded if any
- Error messages
- Steps you have taken, so we can reproduce the problem