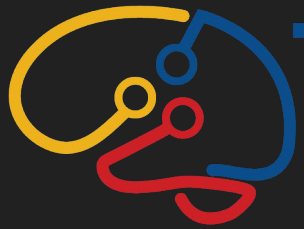


TReNDS
Translational Research in
Neuroimaging & Data Science

Introduction to the TReNDs Cluster

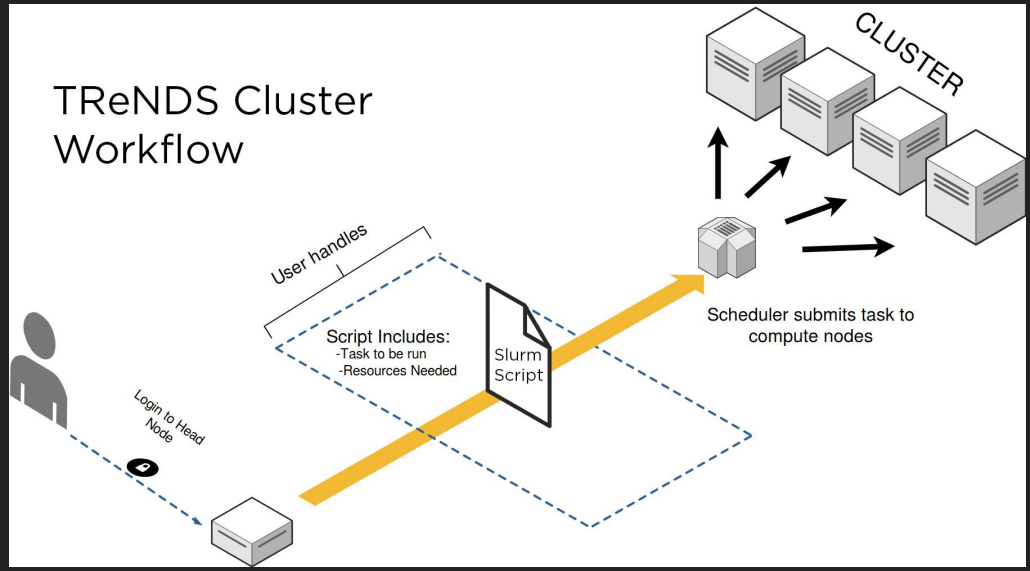
January 2024

Slides and Resources:

https://github.com/trendscenter/ClusterWorkshop

What is the TReNDs Cluster? Why should I care?

- A set of computational resources with a shared file system
- Submitted processes managed by SLURM, which allocates resources
- Work with interactive sessions or BATCH scripting



Overview of Available Resources

We actually have more than this now!

Number	Manufacturer	Cores	Memory	GPUs	Name
20	Intel	32	768 GB		arctrdcn{001-020}
3	Intel	96	1.5 TB		arctrdhm{001-003}
20	AMD	64	512 GB	1xNvidia 2080	arctrdagn{001-020}
4	Nvidia DGX-1	40	512 GB	8xNvidia V100	arctrddgx{001-004}
2	Dell	40	192 GB	4xNvidia V100	arctrdgn{001-002}

How do I get Help?

Attend this Workshop!

Github Link: https://github.com/trendscenter/ClusterWorkshop

Then, check the wiki:

TReNDs Cluster Wiki: https://trendscenter.github.io/wiki/

Next, ask here:

HPC-TIPS Slack Channel

or email me: bbaker43@gsu.edu

Preferably message me on slack!

Finally, if we need to get IT involved!

HYDRA Tickets: hydra.gsu.edu (how to create a Hydra ticket)

The Very Basics

Account Requests and SSH Installation

Step 1:

Navigate to: elpis.rs.gsu.edu

Login with GSU Credentials (if you don't have them proceed to step 2)

If you already have an allocation for TReNDs, you're done! Move to step 3.

Step 2:

Contact your PI to request an account or allocation.

They will contact the operations team to get your account set up

Step 3:

OSX + Linux come with SSH built in

On Windows, you will need to install OpenSSH
(IT will need to do this on managed machines)

Configuring your VPN (Remote Only)

Check out the GSU Guide: https://technology.gsu.edu/technology-services/it-services/security/virtual-private-network/

You will need to download the client

(IT may need to install on a managed machine)

You will also need to set up DUO for two-factor authentication

Finally, use your GSU credentials and the address **secureaccess.gsu.edu**.

Generating and Signing SSH Keypairs

Hands-on Walkthrough

$ mkdir ~/.ssh

$ cd ~/.ssh

$ ssh-keygen -f id_<campusid>

$ cat ~/.ssh/id_<campusid>.pub

Log into https://elpis.rs.gsu.edu/ and we will sign the certificate

Download the files, including id_<campusid>-cert.pub and move into your .ssh folder

$ mv ~/Downloads/id_<campusid>-cert.pub ~/.ssh

$ ssh-add ~/.ssh/id_<campusid>

Configuring SSH

Hands on Walkthrough

Create the file ~/.ssh/config

Add the following to it

```
Host arclogin
    HostName arctrdlogin001.rs.gsu.edu
    User <campusid>
    ForwardAgent yes
    CertificateFile ~/.ssh/id_<campusid>-cert.pub
    IdentityFile ~/.ssh/id_<campusid>
```

(You can also add specific nodes and other things to ease access)

Connecting to the Login Node

Hands on demonstration

$ssh arclogin

And that’s it! You’re officially on the cluster

Best Practices: Login Node

- The Login Node is **everyone's gateway to the Cluster**.
  - If it goes down, people will be unable to work.
  - If it is busy running other processes, people will be unable to work
- **DO NOT RUN ANY ANALYSIS ON THE LOGIN NODE**
- **DO NOT RUN VSCODE OR OTHER REMOTE GUIs ON THE LOGIN NODE**
- **DO NOT LIST HUGE DIRECTORIES, OR DO OTHER FILESYSTEM OPERATIONS ON THE LOGIN NODE**
  - Do not use it to copy files, move directories, etc. All of this uses the processing power needed on that node
- In general, only use the login node to **SUBMIT and MONITOR SLURM JOBS.**

~Break~ Linux Tips Part 1

Directory Navigation

Navigate to a directory: `$ cd ~`

List its contents: `$ ls ~`

Local Machine Resources

Processors: `$ lscpu`

RAM: `$ free -h`

Check current usage: `$ top`

`$ htop`

Check free space on the server: `$ df -h`

GPUs: `$ nvidia-smi`

`$ nvtop`

Command Line Text Manipulation

CAT: `$ cat <filename>`

HEAD: `$ head -n 1 <filename>`

TAIL: `$ tail -n 1 <filename>`

NANO Editor: `$ nano <filename>`

VIM Editor: `$ vi <filename>`

If you want to try it, add these lines to the file ~/.bashrc:

```
source /usr/share/lmod/lmod/init/bash
module use /application/ubuntumodules/localmodules
```

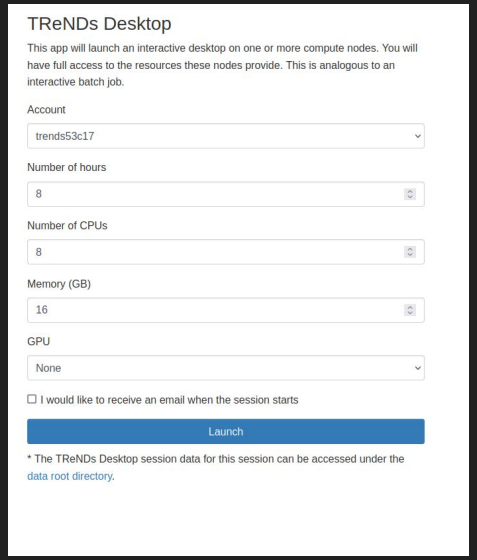
Visual Editing on the Cluster

Hemera: **hemera.rs.gsu.edu**

- Log in with your credentials
- Complete GUI-based access to the cluster via a desktop
- Or specific applications: jupyter, MATLAB, etc

VSCode: **code.visualstudio.com**

- Install VSCode from online
- Install the Remote Development Extension
- Make sure SSH is properly configured
- Connect!



The Dev Nodes

What are they?

Three Machines which can be directly accessed without SLURM. These are useful for developing and testing code, navigating the cluster, transferring files, etc.

arctrdcn017.rs.gsu.edu (CPU-only)

arctrdagn019.rs.gsu.edu (GPU)

arctrdgndev101.rs.gsu.edu (GPU)

How to use them?

Add to your SSH config!

```
Host arctrdcn017
    Hostname arctrdcn017.rs.gsu.edu
    User <campusid>

Host arctrdagn019
    Hostname arctrdagn019.rs.gsu.edu
    User <campusid>

Host arctrdgndev101
    Hostname arctrdgndev101.rs.gsu.edu
    User <campusid>
```

Common Issues

Service not started (Windows):

```
$ Set-Service -Name ssh-agent -StartupType Automatic -Status Running
```

Permission Denied (MAC):

Make sure these are in your ssh config:

```
Host *
    AddKeysToAgent yes
    IgnoreUnknown UseKeychain
    UseKeychain yes
    ForwardAgent yes
```

Data Storage

Data Storage - No place like $HOME

What is your **$HOME** directory?

- Where you land when you login by default
- Limited storage space per user - don’t put too much in there
- Instead use a directory in **/data/users#**

Data Storage - Your Users Directory

Everyone gets a personal directory under the **/data/users#** hierarchy. We recently redivided these directories

```
$ cd /data/users#
```

If you don’t have one - you can create one!

Check the other directories and see how many users are there.

Then add your directory somewhere to keep the balance!

```
$ mkdir /data/users#/<campusid>
```

TIP: Avoid hardcoding this location by using environment variables

```
$ export MYDATA=/data/users#/<campusid>
```

Data Storage - Transferring Data To/From the Cluster

- IN GENERAL, **DON'T PULL DOWN FULL DATA SETS.** You will violate DUAs and storage agreements. If you want to pull single scans, check with your PI if it is ok to do so.
- SIMILARLY, **DO NOT PUT DATA WITH PHI FROM YOUR MACHINE ONTO THE CLUSTER**. Talk to your PI about proper anonymization procedures
- IN PRINCIPAL, FOLLOW HIPPA COMPLIANCE AND OTHER TRAINING. USE **COMMON SENSE SECURITY PRACTICES**
- You can follow this guide to upload data with GLOBUS. Use the online form to place your data in /data/trends_public, then PROMPTLY MOVE IT
- For smaller amounts of files and folders, you can use SCP/SFTP

```
$ sftp arctrdgndev101
> cd /data/users#/<campusid>
> put local_filename.txt
> get remote_filename.txt
```

Data Storage - Where are data and applications?

Applications:

`/trdapps`

Neuromark Datasets:

`/data/qneuromark`

`/data/neuromark2`

Data sets from Collaborators:

`/data/collaboration`

These directories and other related ones are highly managed, i.e. you will not have permission to put data there.

If there is new data that will be put onto the cluster for your project, talk to your PI and they will help facilitate that process

Data Storage Best Practices

- Avoid duplicating data sets that already exist
  - Use references to static copies
  - Write to your users directory, and reuse results when possible
- Remove unneeded or duplicate data/results
- Compress folders you want to save for the future
  - Use tar: https://www.geeksforgeeks.org/tar-command-linux-examples/
  - Or zip: https://ioflood.com/blog/zip-linux-command/
- Use GitHub to store code (do not place data there)
- Use common sense security
  - Do not upload data with sensitive information (yours or others')
  - Get permission before

Some Notes about Github

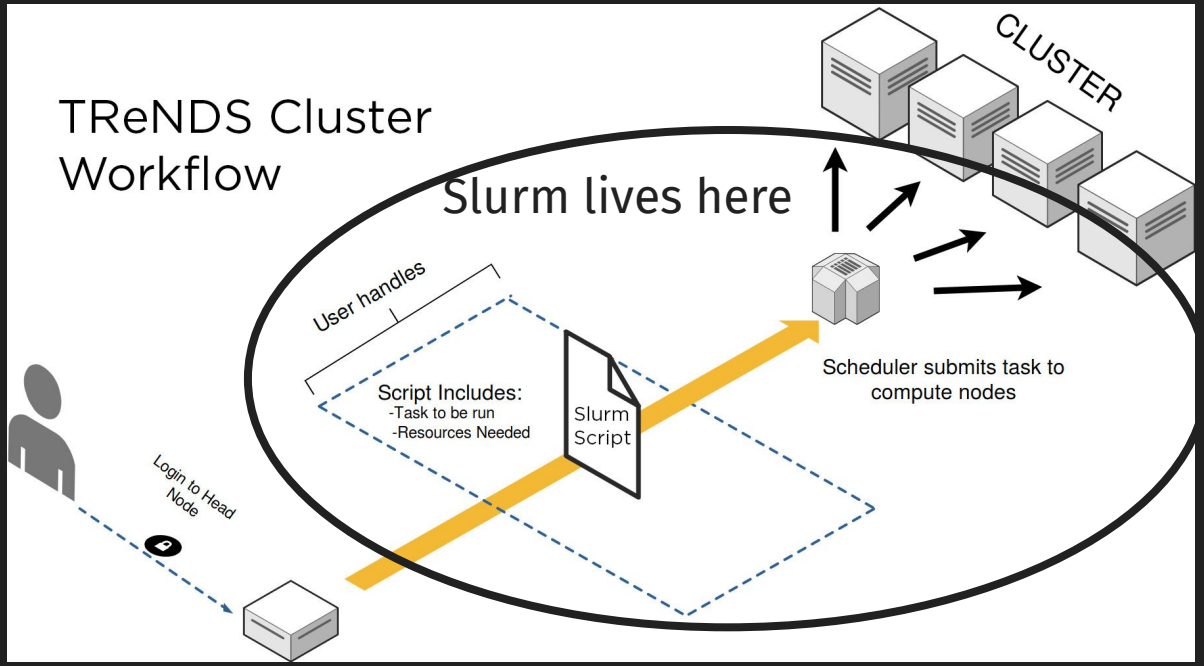
- Git is an extremely useful tool for tracking changes in code
- Learn the basics of using Git on the command line
  - $ git clone
  - $ git push
  - $ git pull
  - $ git add
  - $ git commit -m “my commit message”
- Github allows storing and tracking of projects which you can share
- We have a TReNDsCenter Github
- Again use common sense
  - Avoid uploading large images and full data sets to github
  - Do not upload sensitive information (passwords, filesystem locations etc)
  - Make sure to manage access to repositories to protect your IP (and TReNDs’)
- Take some time to carefully manage permissions
- Delete unused repositories to avoid clutter
- Add extensive documentation etc
- **Walkthrough:** creating a github repository

Using SLURM - the Basics

What is SLURM?

Basically - users submit “jobs” and SLURM decides which nodes to use and manages multiple users

We use commands to monitor nodes and schedule jobs



The Slurm Queues

qTRD - general purpose computing : CPUs only

qTRDGPU - Hybrid Use + GPU computing: few GPUs per machine

qTRDHM - Many CPUs (32+) and High Memory: CPUs only

qTRDGPUH - HIgh priority GPU nodes (Max 8 GPUs per user)

qTRDGPUM - Medium priority GPU nodes (Max 16 GPUs per user)

qTRDGPUL - Low priority GPU nodes (No limit)

Monitoring Jobs and Resources

There are three big commands for monitoring SLURM resources:

$ squeue - monitors jobs in the queues
$ sinfo - what nodes are available
$ sacct - monitor all jobs
(even completed)

Note - Queuing, Resources, and Preemption

SLURM will tend to **prioritize new jobs** submitted from **multiple users**, over many old jobs from one user

The exception to this is if one user gets a ton of jobs allocated, and the jobs take a long time.

Preemption and resource limits solve this issue on the GPU nodes. Users can only use so many resources with maximum priority. Even if they use more resources at lower priority, someone else can access those resources.

Preemption in theory suspends the job and it should restart when the resource is available again.

You can explicitly specify priority for other jobs too, but be wary of preemption.

Interactive Jobs - the SRUN command

$ srun - open up an interactive terminal on a cluster node

Hands on examples:

```
$ srun -p qTRD -A trends53c17 -c 4 --nodes=1 --ntasks-per-node=1 --mem=4G --time=1:00:00 --pty -J myInteractiveJob /bin/bash
```

```
$ srun -p qTRDGPU -A trends53c17 -c 4 --gres=gpu:1 --nodes=1 --ntasks-per-node=1 --mem=4G --time=1:00:00 --pty -J myInteractiveJob /bin/bash
```

Interactive Jobs - Requesting Resources

Let's break apart those last two commands:

srun	**Interactive Job**	srun
-p qTRD	**The queue**	-p qTRDGPU
-A trends53c17	**The cluster account**	-A trends53c17
	# of GPU resources	--gres=gpu:1
-c 4	**# of CPUs**	-c 4
--nodes=1	**# of Nodes**	--nodes=1
--ntasks-per-node=1	**# of Tasks**	--ntasks-per-node=1
--mem=32G	**Amount of RAM**	--mem=32G
--time=1:00:00	**Timeout for Job**	--time=1:00:00
--pty	**Run a terminal**	--pty
-J myInteractiveJob	**Job Name**	-J myInteractiveJob
/bin/bash	**Command to run**	/bin/bash

Requesting Resources - What can I request?

SLURM has a TON of options for requesting resources. You can research more in the documentation. These are the most common to use for SRUN:

-p, --partition	The queue to use e.g. qTRD
-A, --account	The account you have on elpis
--gres	The GPU resources to request.
-c, --cpus-per-task	The CPU resources to request
-n, --nodes	The number of nodes to request
-N, --ntasks-per-node	The number of tasks on each node
--mem	The amount of RAM to request
-t, --time	The time to run the job
--pty	Treat the job as a terminal
-J, --job-name	Give the job a name

Note: you can request specific GPU types with the following:
--gres=gpu:RTX:1, --gres=gpu:V100:1, --gres=gpu:A100:1, --gres=gpu:A40:1

What resources should I request?

- Configurations that don't exist will result in errors: e.g. requesting GPU resources on qTRD.
- In general, request the **minimum resources** needed for a job.
- Try to guess your needed runtime, and don't leave sessions idle without using them.
- Some rules of thumb:
  - Line up the resources you request so that resources will not be idle.
  - For example: qTRD Machines have 768 GB of RAM, and 32 CPUs. Therefore, try not to go above **768/32=24 GB** of RAM per CPU. Otherwise, there will be idle CPUs with no RAM remaining, or idle RAM with no CPUs available.
  - **Especially important for GPU nodes!!!!**
  - For example: qTRDGPU machines have 512 GB of RAM and 64 GPUs, but only **ONE GPU PER MACHINE**. Try not to go above 6-7 GB of ram per CPU, and think about how many CPUs you really need for GPU jobs *(usually no more than 4-8)*.
  - Leave some resources available for use with GPUs!

Fair Resource Request Suggestion Table

Here's an idea of how to USE resources fairly

queue	GPU Type	RAM per machine	# CPUs per machine	# GPUs per machine	Recommended MAX RAM per CPU	Recommend Max CPU-Only Usage on Hybrid Nodes	Recommend MAX resources per GPU
qTRD	N/A	768 GB	32	N/A	24 GB	N/A	N/A
qTRDHM	N/A	1500 GB	96	N/A	~15 GB	N/A	N/A
qTRDGPU	RTX	512 GB	64	1	7 GB	Leave 8 CPUs and 128GB free	4 CPUs, ~128 GB RAM
qTRDGPU	A40	512 GB	128	2	2 GB	Leave 16 CPUs and 256 GB RAM free	4-8 CPUs, ~128 GB RAM
qTRDGPUL/M/H	V100	512 GB	40	4	**DO NOT RUN CPU ONLY JOBS**	N/A	10 CPUs, ~128 GB per GPU
qTRDGPUL/M/H	A100	1000 GB	192(-256)	8	**DO NOT RUN CPU ONLY JOBS**	N/A	24 CPUs, ~128 GB per GPU

Interactive Jobs - Using Modules and TReNDs Apps

For a full list of software use:

```
$ module avail
```

To load a particular module:

```
$ module load matlab
```

To unload a module:

```
$ module unload matlab
```

Some software is available in /trdapps. You can access it by adding it to your PATH.

E.g. for linux binaries:

```
export PATH=$PATH:/trdapps/linux-x86_64/bin/
```

Or for MATLAB toolboxes:

```
>> addpath(genpath('/trdapps/linux-x86_64/matlab/toolboxes/GroupICATv4.0c/'));
```

Interactive Jobs - Hands on Examples

MATLAB - Accessing GIFT

Step 0: Add the following to your .bashrc

```
source /usr/share/lmod/lmod/init/bash
module use /application/ubuntumodules/localmodules
```

Step 1: Start the interactive job

```
$ srun -p qTRD -A trends53c17 -c 1 --nodes=1 --ntasks-per-node=1 --mem=4G --time=1:00:00 --pty -J matlab /bin/bash
```

Step 2: Load MATLAB, and open the MATLAB Command Line

```
$ module load matlab
$ matlab
```

Step 3: Add GIFT to path

```
>>addpath(genpath('/trdapps/linux-x86_64/matlab/toolboxes/GroupICATv4.0c'));
```

Step 4+: Use GIFT Scripts in the Command Line

```
>>help icatb_loadData
```

PYTHON Command Line + NVTOP GPU

Step 0: do step 0 from the MATLAB example

Step 1: Start the interactive job

```
$ srun -p qTRDGPU -A trends53c17 -c 1 --nodes=1 --ntasks-per-node=1 --mem=1G --gres=gpu:1 --time=1:00:00 --pty -J python /bin/bas
```

Step 2: Load the python module

```
$ module load python
```

Step 3: Open the python command line

```
$ python3
```

Step 4: exit and run nvtop and nvidia-smi to check the GPU usage (none)

```
$ /trdapps/linux-x86_64/bin/nvtop
$ nvidia-smi
```

Interactive Jobs - Best Practices

- Do not leave interactive jobs running! Set reasonable time-limits and exit your jobs when finished.

- In general interactive jobs are useful for debugging code and running small examples, but they are often inefficient for large analyses

- Interactive jobs are also useful for moving around data in your folders (DO NOT USE THE LOGIN NODE TO MOVE OR LIST LARGE DIRECTORIES)

- Experiment with different available modules!
  We have R, GCC, AFNI, ANTS, various other tools

~Break: Linux Tips Part 2~

What are .bashrc and .bash-profile?

Basically, they contain commands which run on startup

Environment Variables and the PATH

- Environment variables are available to all processes within a terminal.
- Your **PATH** defines the locations where your terminal looks for executables
- Use the **.bashrc** to export environment variables which you use frequently, such as the location of your directory, and to permanently modify your PATH if needed!
- For example, add this to your .bashrc:

  export MYDIR=/data/users#/campusid (for example using nano or vim)

  Then run **$ source ~/.bashrc** (this just reloads the .bashrc file)

  And try the command **$ cd $MYDIR**

~Break: Linux Tips Part 2~

You can also add useful shortcut commands to your .bashrc called **aliases.**

For example, try adding this to your .bashrc:

```
alias go2data=”cd /data/users#/campusid”
```

The *language* of the .bashrc and of the following section on SBATCH is **BASH**, a shell language. There are a ton of resources on learning the basics of BASH. Here are some quick tips that will be useful later:

```
# Comments use the the Hashtag symbol
# Save a string into the variable myvar
myvar=”some string”
# Access a variable by preceding it with $
echo $myvar
# Save the output of a command in a variable by using asterisks
myvar2=`echo $myvar`
# Access input arguments by using $1, $2, $3, etc.
echo $1
```

~ Break: Python Tips ~

For use of Python on the cluster, there are a few ways you can install packages. The easiest way to do it is to create your own miniconda3 installation:

```
$ mkdir -p /data/users#/<campusid>/bin/miniconda3
$ cd /data/users#/<campusid>/bin
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O miniconda_install.sh
$ bash miniconda_install.sh -b -u -p /data/users#/<campusid>/bin/miniconda3
```

<follow the instructions on the command line…>

Make sure you point the installation to the `/data/users#/bin/miniconda3`

Then, restart your terminal and you can use a full conda environment with full installation capabilities

SLURM - BATCH Processing

SLURM BATCH - Getting Started

SBATCH jobs allow you to submit a process to run, and then walk away

To run a SBATCH job, you will create a script to run, and then use the sbatch command.

Unlike with SRUN, you can specify all of the options in a header within that script

Let's create a script called JobSubmit.sh with the information on the right, and run it using the following command:

$ sbatch JobSubmit.sh

Example Script

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=10G
#SBATCH -t 1:00:00
#SBATCH -e error%A.err
#SBATCH -o out%A.out
#SBATCH -A trends53c17
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your email>
#SBATCH --oversubscribe
# a small delay at the start often helps
sleep 10s

# print some message to the log
echo "hello sbatch world!"

# it can be helpful for debugging to get the node name
echo $HOSTNAME >&2
# a delay at the end is also good practice
sleep 10s
```

SLURM BATCH - Understanding the Header

The header in a SBATCH script defines the resources you want to request for that job! Just like we did before with SRUN. Plus, we have some additional options

These options define the location of error and output logs for your job!

These options define where SLURM will send E-Mail notifications when jobs complete!

Just leave the --oversubscribe option. Check the SLURM docs for more details.

The %A value places the job ID in the log name.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=10G
#SBATCH -t 1:00:00
#SBATCH -e error%A.err
#SBATCH -o out%A.out
#SBATCH -A trends53c17
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your email>
#SBATCH --oversubscribe
# a small delay at the start often helps
sleep 10s

# print some message to the log
echo “hello sbatch world!”

# it can be helpful for debugging to get the node name
echo $HOSTNAME >&2
# a delay at the end is also good practice
sleep 10s
```

SLURM BATCH - Monitoring and Control

You can use the **squeue** command to monitor your jobs only:

```
squeue -u <campusid>
```

Or the jobs in a particular queue:

```
squeue -p qTRDGPU
```

You can also cancel your submitted jobs using **scancel**:

```
scancel <jobid>
```

Or all of your jobs in the queue:

```
scancel -u <campusid>
```

Or all of your jobs on a particular queue:

```
scancel -u <campusid> -p qTRDGPU
```

Check the SLURM documentation for more flags for these commands!

SLURM BATCH - Logging Tips

All of the options we had available for SRUN are also available for SBATCH, and they can also be specified when you call SBATCH on the command-line, rather than in the header.

This can be useful, for example, if you want to create logs dynamically as you submit your script multiple times:

```
$ sbatch -e err_myjob1_%A.err -o out_myjob1_%A.out JobSubmit.sh
```

```
$ sbatch -e err_myjob2_%A.err -o out_myjob2_%A.out JobSubmit.sh
```

You can imagine how this can be useful for submitting multiple jobs with different configurations!

You can also specify for jobs to exist in a folder, but if that folder doesn't exist the job will error out quietly! This is a common bug!

SLURM BATCH - Array Jobs

SLURM Batch is perhaps most useful for submitting many jobs at once. Although you could use FOR-LOOPS to submit individual SBATCH calls, ARRAYS are the way to go!

Basically, it runs an SBATCH script N many times where each script gets a unique index i.e. the $SLURM_ARRAY_TASK_ID

This is useful when you have many jobs that can run in parallel, where each job uses a single index to grab a particular data set for example

The %a references the Array Index

```
#!/bin/bash
#SBATCH -p qTRD
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=1g
#SBATCH -p qTRD
#SBATCH -t 1:00:00
#SBATCH -J basic_array
#SBATCH -e error%A-%a.err
#SBATCH -o out%A-%a.out
#SBATCH -A trends53c17
#SBATCH --oversubscribe
sleep 10s

echo $HOSTNAME >&2

echo Array Index: $SLURM_ARRAY_TASK_ID

sleep 10s
```

Try putting the above script into JobArray.sh, and run the following:
$ sbatch --array=0-4 JobArray.sh

SLURM BATCH - Array Job Tips

Accessing a particular line in a file using the $SLURM_ARRAY_TASK_ID

Suppose I have a file with a list of paths for individual subjects. I can grab the subject I need in a particular file by using

```
sed -n "$(( $SLURM_ARRAY_TASK_ID + 1 )) p" lines.txt
```

Try the example on the right in JobArray_sed.sh:

```
sbatch --array=0-4 JobArray_sed.sh
```

```
#!/bin/bash
#SBATCH -p qTRD
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=1g
#SBATCH -t 1:00:00
#SBATCH -J sed_example
#SBATCH -e error%A-%a.err
#SBATCH -o out%A-%a.out
#SBATCH -A trends53c17
#SBATCH --oversubscribe
sleep 10s
echo $HOSTNAME >&2
echo Array Index: $SLURM_ARRAY_TASK_ID
cd $MYDIR/ClusterWorkshop/Examples/Basics

# this uses a bash trick to save the output from the sed command in
variable
lineFromFile=`sed -n "$(( $SLURM_ARRAY_TASK_ID + 1 )) p" lines.txt`
echo $lineFromFile
sleep 10s
```

SLURM BATCH - Array Job Tips 2

Rather than submitting a full array, you can use the modulus to indicate that only every Nth job will run simultaneously:

```
sbatch --array=1-8%2 JobArray.sh
```

The cluster has an **array size limit of 5000**, so if you want to use higher indices, you'll need to break up the submissions and reindex your arrays. E.g.

```
offset_index=$(($SLURM_ARRAY_TASK_ID + 10000))
```

SLURM BATCH - BEST PRACTICES

- Just like with SRUN - be wary of letting jobs run for too long. Set reasonable time limits, and stop your jobs using **scancel** if you expect they are hanging or not using resources.
- Keep a close eye on your jobs - if they are running forever, stop them!
- Only request the resource you need! Use the same practices as we discussed with SRUN.
- When submitting many jobs, try to find a way to use ARRAYS rather than submitting a huge number of SBATCH calls
- Limit array sizes to whatever you need in a given time, and use the modulus to limit the amount of jobs that run simultaneously!

Putting it All Together

Hands on Example 1 - Single Subject Neuromark ICA

- Step 0:
  - $cp /data/users2/bbaker/fbirn_subject_list.txt $MYDIR/ClusterWorkshop/Examples/
- Step 1:
  - $ cd ClusterWorkshop/Examples/SingleSubjectICA
- Step 2:
  - Modify gigica_step1.m if needed
- Step 3:
  - $ sbatch JobSubmit.sh

```
#!/bin/bash
#SBATCH -p qTRD
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=4G
#SBATCH -t 1:00:00
#SBATCH -e error%A.err
#SBATCH -o out%A.out
#SBATCH -A trends53c17
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your email>
#SBATCH --oversubscribe
#SBATCH -J cworkshop_multi_ica

# a small delay at the start often helps
sleep 10s

# print some message to the log
module load matlab

# CD into your directory
cd $MYDIR/Examples/SingleSubjectICA
# run the matlab batch script
matlab -batch 'gigica_step2'

# a delay at the end is also good practice
sleep 10s
```

Hands on Example 2 - Multi Subject Neuromark ICA

- Step 0:
  - $cp /data/users2/bbaker/fbirn_subject_list.txt $MYDIR/ClusterWorkshop/Examples/
- Step 1:
  - $ cd ClusterWorkshop/Examples/MultiSubjectICA
- Step 2:
  - Modify gigica_step1.m if needed
- Step 3:
  - $ sbatch --array=0-4 JobArray.sh

```
#!/bin/bash
#SBATCH -p qTRD
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=4G
#SBATCH -t 1:00:00
#SBATCH -e error%A_%a.err
#SBATCH -o out%A_%a.out
#SBATCH -A trends53c17
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<your email>
#SBATCH --oversubscribe
#SBATCH -J cworkshop_multi_ica

# a small delay at the start often helps
sleep 10s

# load the matlab module
module load matlab

# CD into your directory
cd $MYDIR/Examples/MultiSubjectICA

# run matlab batch
matlab -batch 'gigica_step2'

# a delay at the end is also good practice
sleep 10s
```

Hands on Example 3 - PyTorch with GPU

- Step 0:
  - We are going to use a conda environment with pytorch installed. Do this in an interactive session:
    - $ conda create -y --name cw_torch
    - $ conda activate cw_torch
    - $ conda install -y pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia
    - conda install -y -c conda-forge scikit-learn
- Step 1:
  - $ cd ClusterWorkshop/Examples/PytorchClassification
- Step 2:
  - $ sbatch JobSubmit.sh

```
#!/bin/bash
#SBATCH -p qTRDGPU
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SGATCH --gres=gpu:1
#SBATCH --mem=4G
#SBATCH -t 1:00:00
#SBATCH -e error%A.err
#SBATCH -o out%A.out
#SBATCH -A trends53c17
#SBATCH --oversubscribe
#SBATCH -J cworkshop_pytorch

# a small delay at the start often helps
sleep 10s

# activate conda
eval "$(conda shell.bash hook)"
conda activate cw_torch

# CD into your directory
cd $MYDIR/Examples/PytorchClassification
# run the matlab batch script
python mnist_classification.py

# a delay at the end is also good practice
sleep 10s
```

Hands on Example 4 - PyTorch with GPU : Cross Validation

- Step 0:
  - We are going to use a conda environment with pytorch installed
  - $ conda create -y --name cw_torch
  - $ conda activate cw_torch
  - $ conda install -y pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia
  - $ pip install scikit-learn
- Step 1:
  - $ cd ClusterWorkshop/Examples/PytorchClassificationCV
- Step 2:
  - $ sbatch --array=0-4 JobSubmit.sh

```
#!/bin/bash
#SBATCH -p qTRDGPU
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SGATCH --gres=gpu:1
#SBATCH --mem=4G
#SBATCH -t 1:00:00
#SBATCH -e error%A_%a.err
#SBATCH -o out%A_%a.out
#SBATCH -A trends53c17
#SBATCH --oversubscribe
#SBATCH -J cworkshop_pytorch_cv

# a small delay at the start often helps
sleep 10s

# activate conda
eval "$(conda shell.bash hook)"
conda activate cw_torch

# CD into your directory
cd $MYDIR/Examples/PytorchClassificationCV
# run the matlab batch script
python mnist_classification.py -k $SLURM_ARRAY_TASK_ID

# a delay at the end is also good practice
sleep 10s
```

EXTRA CREDIT

Flexible BATCH Scripting

- You can use BASH arguments to supply variables to SBATCH scripts.
- You can't really dynamically allocate the SBATCH header, so use the flags as needed
- Example:
  - $ sbatch -e myerror.e -o myout.o JobSubmit.sh arg1 arg2 arg3

```
#!/bin/bash
#SBATCH -p qTRD
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=10G
#SBATCH -t 1:00:00
#SBATCH -A trends53c17
#SBATCH --oversubscribe
#SBATCH -J cw_ex_args
# a small delay at the start often helps
sleep 10s

# print some message to the log
echo "hello sbatch world!"

echo We got some arguments $1 $2 $3

# it can be helpful for debugging to get the node name
echo $HOSTNAME >&2
# a delay at the end is also good practice
sleep 10s
```

Building and Using Singularity Images

- You can build singularity images using the docker client (restricted access)
  - Or you can build directly from remote docker repositories!
  - Example from here
- $ module load singularity
- $ singularity build fmriprep-<version>.simg docker://poldracklab/fmriprep:<version>
- $ singularity run --cleanenv fmriprep.simg \
  path/to/data/dir path/to/output/dir \
  participant \
  --participant-label label

Jupyter Notebooks without Hemera

- Step 1: Create a slurm script to host the jupyter server
- Step 2: Submit the job and recover the node address
- Step 3: Run SSH tunnel to the node and use the port
- Navigate to http://localhost:<port>

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 1
#SBATCH --mem=10g
#SBATCH -p qTRD
#SBATCH -t 1440
#SBATCH -J jupyter
#SBATCH --output=jupyter-%j.out
#SBATCH -A <slurm_account_code>
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<email address>
#SBATCH --oversubscribe

eval "$(<path_to_conda> shell.bash hook)"
conda activate <your_enviroment>
cat /etc/hosts
jupyter-lab --ip=0.0.0.0 --port=${1:-<port>}
```

Granular Control with Multiple Tasks

- You can allocate multiple nodes with SBATCH, and then use SRUN within that to tell each node to do multiple things

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -n 2
#SBATCH -c 10
#SBATCH --mem=100g
#SBATCH -p qTRD
#SBATCH -t 1440
#SBATCH -J granulartest
#SBATCH -e error%A.err
#SBATCH -o out%A.out
#SBATCH -A trends53c17
#SBATCH --oversubscribe

sleep 10s

echo $HOSTNAME >&2

module load matlab/R2022a
srun -N1 -n1 numactl --localalloc echo "hello node1" &
srun -N1 -n1 numactl --localalloc echo "hello node2" &

wait

sleep 10s
```

Multi-GPU Jobs: Pytorch Example

- Very simple for single machines with multiple GPUs:
  - JUST REQUEST MORE GPUS!
- More difficult is performing distributed computing with multiple NODES
  - You can use the granular control example to do that
  - Or some clever tricks with BASH
  - That’s beyond extra credit :)

```
#!/bin/bash
#SBATCH -p qTRDGPU
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -c 8
#SBATCH --gres=gpu:2
#SBATCH --mem=20G
#SBATCH -t 1:00:00
#SBATCH -e error%A.err
#SBATCH -o out%A.out
#SBATCH -A trends53c17
#SBATCH --oversubscribe
#SBATCH -J cworkshop_pytorch

# a small delay at the start often helps
sleep 10s

# print some message to the log
eval "$(conda shell.bash hook)"
conda activate cw_torch

# CD into your directory
cd $MYDIR/ClusterWorkshop/Examples/ExtraCredit/MultiGPUPytorch
# run the matlab batch script
python -u dataparallel.py

# a delay at the end is also good practice
sleep 10s
```

Thank you!