ECE 455
Euler Cluster and Slurm

Dr. Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Wisconsin at Madison, Madison, WI

Objective

- **Go through how to run a program on a Euler node using Slurm**
  - You use Slumr to submit your assignments to Euler cluster for grading
  - You can work on your local machines and submit final code to Euler

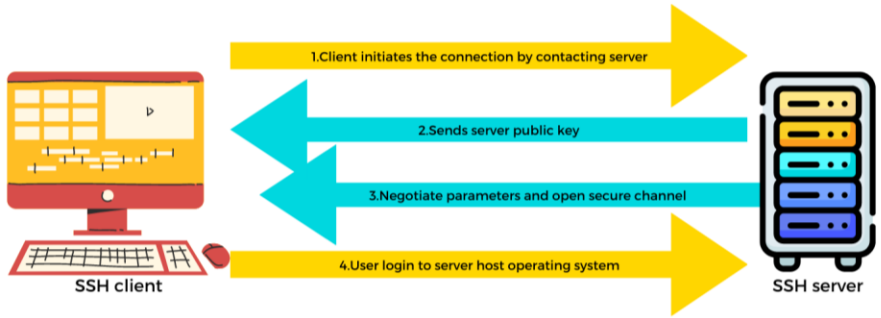
- **Do not run code on the log-in node of Euler**
  - CAE will suspend your account.
  - Use Slurm to launch your program.

Background of Euler

- **Euler is a heterogeneous cluster managed by CAE within CoE**
  - Heterogeneous: nodes with CPUs/GPUs from AMD, Intel, NVIDA, IBM, etc.
- **Euler runs Linux – basic Linux skills are needed**
  - Need to know a few commands to work from command line interface (CLI)
  - Ex: “ls” to list the current directory structure, “date” to show the current time
- **Euler is a shared resource**
  - Please use in a way that is courteous toward other users
  - Do not submit hundreds of job at one time which will block others’ queues
    - Carefully name your submitted jobs – other people can see your job names

Connecting to Euler

- **Connecting to Euler is done using ssh**
  - Windows, Mac, or Linux users will be able to use ssh from their terminals
- **Connect to Euler like this**



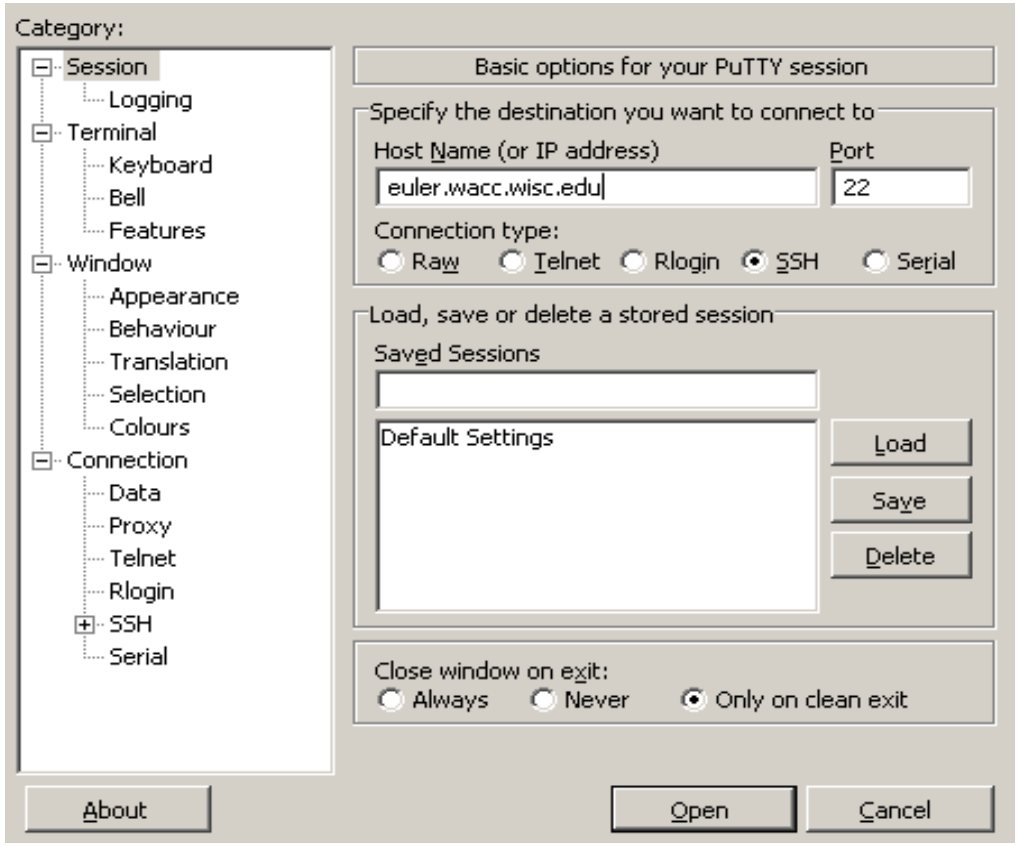

```
harry@oak:~$ ssh haoruiz@euler.engr.wisc.edu
The authenticity of host 'euler.engr.wisc.edu (2607:f388:1082:4a00:ec4:7aff:fe6f:28bc)' can't be established.
ED25519 key fingerprint is SHA256:KZ6Qjew+ZNGiu9ob0pnd70ncfE2am35q8Ww3eELoAXE.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:4: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'euler.engr.wisc.edu' (ED25519) to the list of known hosts.
================================================================================

UPCOMING OUTAGE: 8/28 - 8/30

Announcements, outage notices, and status updates are available via the
Euler Mailing List. Sign up with a wisc.edu email address at:

https://go.wisc.edu/77vc50

================================================================================
Last login: Fri Jul 14 17:07:09 2023 from 10.140.138.80
[haoruiz@euler-login-2 ~]$ ls
chrono-internal  data  Packages  repo759  test_multi.sh  test.sh
[haoruiz@euler-login-2 ~]$
```

ssh username@euler.engr.wisc.edu

- username is your “CAE username”, password is your CAE account password
- **Tutorial:** https://kb.wisc.edu/cae/page.php?id=138004

Various Third-party SSH Clients Exist

- **Like putty: https://www.putty.org/**
  - Pretty clunky…
- **Best way is via your or CAE machines**



OK, You're in Euler. Now What?

- **Slurm Workload Manager**
  - Originally **S**imple **L**inux **U**tility for **R**esource **M**anagement
  - A suite of tools that schedule jobs on clusters and supercomputers
  - Widely used in industry for cluster-based resource management
    - Nvidia, Cadence, Synopsys, Citadel, etc.
- **Euler uses Slurm to share hardware and software resources**
  - Slurm: Uses a batch script as input; does what the shell commands in the script ask it to do…
  - **slurm** is the name of the scheduler; **sbatch** is one of its utilities

Shell and Shell Scripts

Content of file `hello.sh`

```
#!/usr/bin/env bash

name_str=“TW”
echo "What is your name?"
read name_str

echo "Hello, $name_str!"
```

```
$ chmod +x hello.sh
$ hello.sh
What is your name?
Joe
Hello, Joe!
$
```

- A shell script is just a series of commands written as you would type them into the interactive shell
- The commands are read and executed as if you had typed them at the shell prompt

Comments on the `hello.sh` script & its execution

`#!/usr/bin/env bash`

- The hash-bang operator `#!` specifies the absolute path to the interpreter to use for a script
- Classically, this would be something like `#!/bin/sh` or `#!/bin/bash`
- Instead, here `/usr/bin/env` is used to locate `bash` in our environment

`name_str=`

- Just like in the shell, we can declare variables

`echo “…”`

- Just like in the shell, we can run commands and their arguments will be substituted and expanded as normal.

`read name_str`

- The `read` command takes a line of input from the user and stores it in the given variable

`bash hello.sh`

- Running a script is as easy as invoking it in the shell
- It is also possible to run a script as if it were a program by marking it executable using `chmod +x`

Shell and Shell Scripts (cont'd)

- **Need to learn more about the shell and CLI?**
  - Study "The Missing Semester of your CS education" by MIT: https://missing.csail.mit.edu/
- **Alternatively, a quick overview is here**
  - https://uwmadison.box.com/s/tq8a3sid2i97k1m57zoym9gbiphj9xz3
- **Finally, if you want to learn quickly, ChatGPT it**
  - Just ask what you need to do at the CLI
  - ChatGPT is amazingly good at helping with bash scripts.
    - Explain in plain words what you need, it'll get a bash script done for you. Try it!

Slurm Batch Scripts

Content of file `hello_slurm.sh`

```
#!/usr/bin/env zsh
#SBATCH --job-name=HelloScript
#SBATCH --partition=instruction
#SBATCH --ntasks=1 --cpus-per-task=1
#SBATCH --time=0-00:00:10
#SBATCH --output=hello_output-%j.txt

cd $SLURM_SUBMIT_DIR

name_str=World
echo “Hello, $name_str!”
```

```
$ sbatch hello_slurm.sh
Submitted batch job 1975385
$ cat hello_output-1975385.txt
“Hello, World!”
$
```

A batch script is passed to the **`sbatch`** `Slurm` utility, which processes the commands/code one line at a time

`#SBATCH ...`

- `#SBATCH` directives can accept any flag that can be passed to **`sbatch`** on the command line.
  - `--job-name` : the name of the job in the scheduler database
  - `--cpus-per-task` : the number of CPU threads that will be allocated for each task
- The # character is recognized as a comment by bash, so these directives don’t interfere with the script itself

`cd $SLURM_SUBMIT_DIR`

- Slurm sets some environment variables when it executes a script
- In this case, we cd to the directory where the script is submitted from

`sbatch` `hello_slurm.sh`

- Submit a script for scheduling by Slurm
- **IMPORTANT**: To use `slurm`, you invoke `sbatch`
  - This is what it means to use “slurm”

Slurm Concept

- **Partition**
  - Each partition has its own priority rules, job queue, hardware, and time limit
  - Euler uses partition to ensure user priority
    - Ex: owner of the GPU typically has the highest priority to run jobs on that GPU
- **Tasks describe a single invocation of a script command**
  - For many jobs, there is only one task that sequentially executes each line of the job script
  - Distributed jobs such as those using MPI or client-server protocols sometimes have multiple tasks which run different parts of the script in tandem
- **Thread: a concurrent logical path of execution run by a CPU**
  - Slurm will usually allocate one hardware thread per CPU that you request
  - Slurm will not assign threads from different jobs to the same CPU core

Breakdown

- The #! header tells the system to interpret our file as a zsh script
- The next thing that goes in the file are the Slurm directives specified via #SBATCH
  - **You must use the instruction partition for this class**
  - It can be helpful to give your job a name
  - Specifying an output file is helpful too

```
#!/usr/bin/env zsh

#SBATCH -p instruction

#SBATCH --job-name=MyAwesomeJob

#SBATCH --output=job_output-%j.txt
```

Breakdown

Choosing a Time Limit

- The Slurm scheduler is much more efficient if users give accurate estimates for how long their jobs will take
- If you don't know for sure, just add 50% onto your best guess
- If you leave it blank, your job will try to request the maximum possible time
  - 60 minutes for jobs on **instruction**
  - Using too long of a time limit may cause unnecessary delays in launching your job
  - Ideally, all your PAs should pass within 1 hour

```
#!/usr/bin/env zsh

#SBATCH -p instruction

#SBATCH --job-name=MyAwesomeJob

#SBATCH --output=job_output-%j.txt

#SBATCH --time=0-00:01:00
```

Breakdown

Requesting CPU Cores

- For multithreaded jobs, request however many cores your program will use with --cpus-per-task or -c
  - Slurm does the dirty work needed to pick out threads and physical cores for you
- For MPI software, you will need to request some number of ranks using --ntasks or --nodes with --ntasks-per-node
- Do not request more cores than your program can use; it is wasteful and can prevent other cluster users from getting their work done

```
#!/usr/bin/env zsh

#SBATCH -p instruction

#SBATCH --job-name=MyAwesomeJob

#SBATCH --output=job_output-%j.txt

#SBATCH --time=0-00:01:00

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=4
```

Breakdown

Requesting Special Hardware

- If you need a specific number (N) of things such as graphics cards, request a **G**eneric **RES**ource
  - --gres=type[:model]:N
  - e.g., gpu:gtx1080:3 or infiniband:1
- Resource selection can be limited to nodes with particular features using constraints
  - NOTE: a constraint is just a hint to the scheduler and does not guarantee any particular resource will be allocated

```
#!/usr/bin/env zsh

#SBATCH -p instruction

#SBATCH --job-name=MyAwesomeJob

#SBATCH --output=job_output-%j.txt

#SBATCH --time=0-00:01:00

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=4

#SBATCH --gres=gpu:1
```

Breakdown

- Slurm invokes the script from your **home directory (place where you log in)**
  - You can invoke your program with an absolute path or cd to an arbitrary directory
  - You can also cd to the directory where you ran sbatch
- You can capture the output of many commands in a single job
- Slurm sets environment variables with information about the running job

```
#!/usr/bin/env zsh

#SBATCH -p instruction

#SBATCH --job-name=MyAwesomeJob

#SBATCH --output=job_output-%j.txt

#SBATCH --time=0-00:01:00

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=4

#SBATCH --gres=gpu:1
#SBATCH --constraint=haswell

cd $SLURM_SUBMIT_DIR

date
hostname
srun awesome_MPI_CUDA_program
```

Breakdown

- Control flow, substitutions, and arguments all work like with normal shell scripts
- We can still process the result of our program or substitute output as we would in a standard bash script

```
#!/usr/bin/env zsh

#SBATCH -p instruction

#SBATCH --job-name=MyAwesomeJob

#SBATCH --output=job_output-%j.txt

#SBATCH --time=0-00:01:00

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=4

#SBATCH --gres=gpu:1
#SBATCH --constraint=haswell

cd $SLURM_SUBMIT_DIR

date
hostname
srun awesome_MPI_CUDA_program
result=$?

if [ $result -ge 0 ]; then
    echo “mpirun in job $SLURM_JOB_ID reported error upon finishing”
    exit $result
fi
```

Example Slurm Script to your HW01

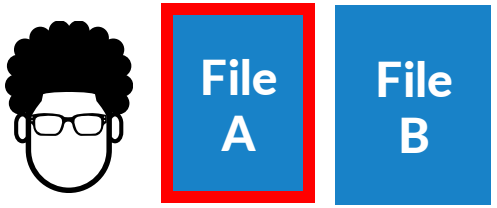
```
#!/usr/bin/env zsh
#SBATCH --partition=instruction
#SBATCH --time 00:01:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --gpus-per-task=1
#SBATCH --output="example.output”

# log in the submission directory
cd $SLURM_SUBMIT_DIR

# load the gcc for compiling C++ programs
module load gcc/13.2.0

# load the nvcc for compiling cuda programs
module load nvidia/cuda

# clone (replace the github link to yours)
git clone https://github.com/tsung-wei-huang/repo455.git
cd repo455/HW01
g++ example.cpp -o example
./example
```

You can use up to 8 CPUs for this course.

You should always use just one GPU for this course.

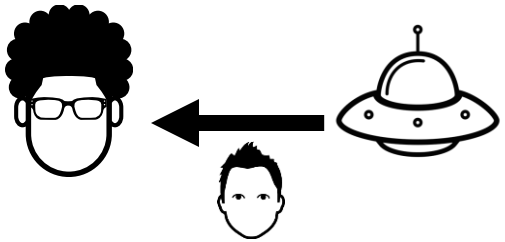
Git Flow

git **add** “File A”

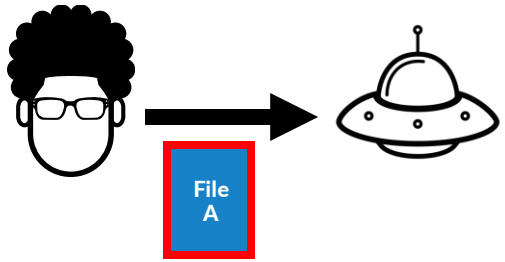


git **commit** –m “New features for project A ”

Pull files from GitHub to your workspace (e.g., Euler)



git **pull**



git **push**

Push files from your local workspace (e.g., CAE/your machines) to GitHub

CAE General Rule (1/3)

- **Euler is a shared resource**
  - Please be mindful of how you use it, there're other folks using it at the same time
  - Allow yourself enough time to get your work done even if Euler is busy
    - Busy: there is a long queue of jobs lined up for execution; you are in the line somewhere…
    - **As a result, don't work on your assignment in the last minutes**
- **Euler is a part of CAE, so CAE rules apply**
  - Use critical thinking before doing things on Euler
  - Euler is here for you to learn, but it isn't a playground
  - Again, work on your or CAE machines for assignment and only test it on Euler

CAE General Rule (2/3)

- **CAE monitors Euler activity**
- **If CAE suspends your account, you can't access your files**
  - It might take time to reactivate your account
  - Be mindful around homework deadlines
  - We will NOT accept late assignment due to this excuse
- **95% of suspensions are for running your code on head node**
  - Happens when you compile and run your program directly from the log-in node
  - Happens when you don't use slurm to schedule the execution of your program

CAE General Rule (3/3)

The first two times a student is caught doing something they shouldn't be doing, [CAE] will send them a warning. They won't need to do anything about it other than to stop doing whatever it is CAE saw.

If, for whatever reason, the student continues misusing the supercomputer after two warnings, we will then reach out to their instructor(s) so that they can intervene and teach the student how to use it properly. We will do this up to two times as well. It is at this point that the instructor should determine whether a punitive deterrent is necessary to prevent or correct the problematic behavior.

Finally, if after being notified twice, the instructor fails to remedy the student's behavior, the case will be referred to the CAE director who will consider further action. […] This is a last resort, but it if the instructor is unable to adequately manage student behavior, the instructor may be barred from using the supercomputer in future courses."

CAE Euler DON'Ts

- **Don't run program on log-in node**
- **Don't run interactive jobs**
- **Don't try to gain administrative access to the cluster**
  - Ex: **sudo** command
- **Don't request resources you don't need**
  - Ex: If your program doesn't need GPU, don't ask for a GPU
- **Don't leave idle session running when you're not around**
  - Ex: Log out when you walk out for lunch
- **Don't use Euler as your personal storage server**
- **Don't do anything which you think will be against the CAE rules**

CAE Euler DOs

- **Do use Slurm via `sbatch` to submit and run your programs**
- **Do use "–p instruction" for all of your jobs**
- **Do provide inputs to your programs using command line arguments or input files**
- **Do think about what your program needs**
  - Ex: if your program is designed to run on two CPU cores, then request only two CPU cores per task
- **Do log out when you are not using Euler**
- **Do contact Euler support if need help**
  - Please reach out to Colin via euler-support@engr.wisc.edu
  - More information: https://kb.wisc.edu/cae/page.php?id=138004

The Module Utility

```
$ gcc --version
gcc (GCC) 8 ...

$ module load gcc/11.3.0
$ gcc --version
gcc (GCC) 11.3.0 ...

$ nvcc main.cu -o cudaprogram
bash: nvcc: command not found

$ module avail nvidia/cuda

-------------- /opt/apps/lmod/modulefiles ---------------------
nvidia/cuda/10.2   nvidia/cuda/11.3  nvidia/cuda/11.6

$ module load nvidia/cuda/11.6
$ nvcc main.cu -o cudaprogram

$ module list
Currently Loaded Module files:
1) gcc/11.3.0   2) nvidia/cuda/11.6


$ module unload nvidia/cuda gcc
$ nvcc
bash: nvcc: command not found
```

- The module command allows you to add utilities to your environment
- "module" provides a method to manage multiple versions of the same utility
- Most common module commands:
  - load, unload, list, and avail
  - You'll use these commands on Euler to handle your assignment

Typical Work Cycle of Assignments

1) Write your solution using your favorite editor/IDE on **your** computer or CAE workstation
2) Build & test the code on **your/CAE** computer until you are convinced that it works
3) Get your files over to Euler using `scp` or via `git`
4) Build your executable on Euler, on the log-in node
5) Write/Recycle `slurm` scripts to define how your executable will be managed by Slurm
6) Run `slurm` script to place your executable in the Slurm queue
7) Once it finished, examine the output of your job
   - By default, output of your program is written to a file starting with "slurm-"
8) If there are any small problems, use a terminal text editor on Euler (like `vim`) to make minor changes to your code and then rerun your `slurm` script to test your program again on Euler

   NOTE: If the results look bad, you might need to go back to 1)
9) Once you are convinced that program works as expected on Euler, push solution into your `git` repo
   - Note: Your code will be graded based on how it runs **on Euler**, not how it works on your computer