Standard Operating Procedure (SOP) for User Onboarding and Job Execution on csis_cluster1

Purpose: -

This SOP provides new users with a definitive guide to accessing the [Your Cluster Name] HPC cluster, understanding its resources, software, and policies, and successfully submitting computational jobs. This ensures consistent, secure, and efficient use of shared resources.

Scope: -

This procedure applies to all institute/department stakeholders who have been granted access to the aforementioned cluster.

Prerequisites: -

- An approved user account with valid credentials.
- An SSH client (For Example: - ssh on Linux/macOS, PuTTY/MobaXterm on Windows).
- Basic familiarity with the Linux CLI mode (Command Line Interface).

Definitions: -

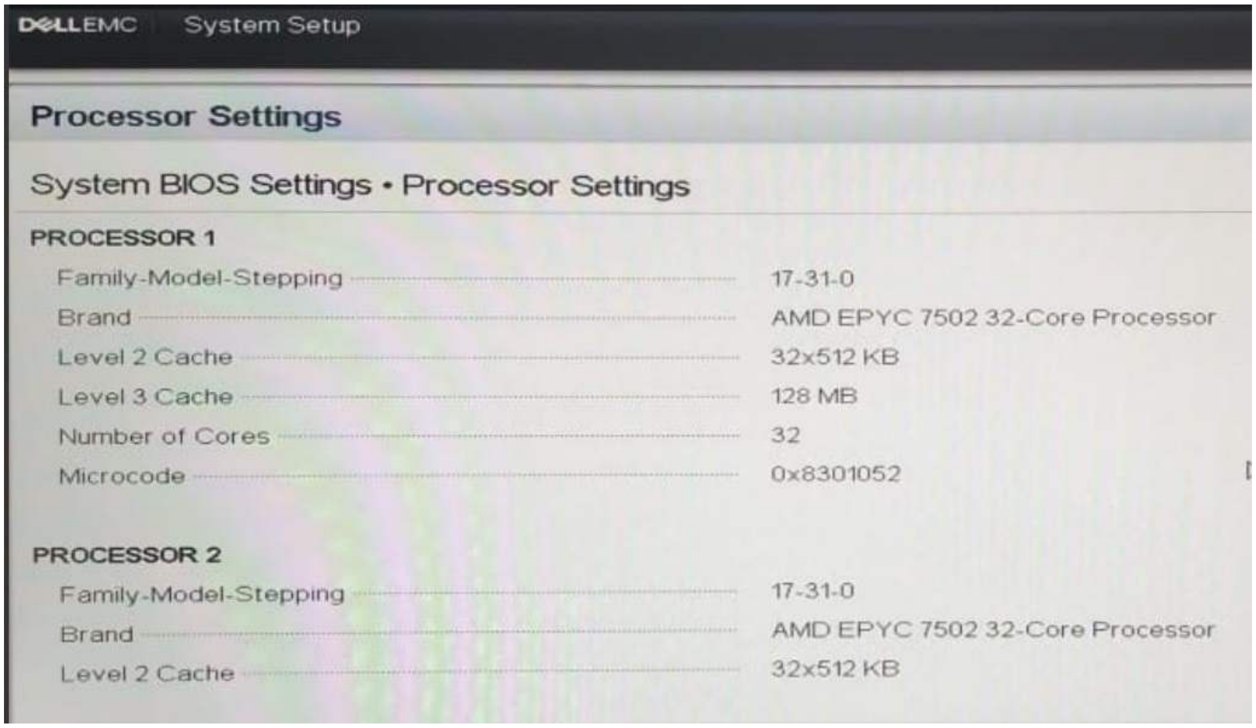
Login Node/Master Node:
The entry point for editing files, compiling code, and submitting jobs.
Slurm:
The job scheduler (Simple Linux Utility for Resource Management) manages all compute resources.

Batch Job:
A computational task is submitted to a queue (partition) to run without user interaction.
QoS (Quality of Service):
A set of limits that override or supplement partition limits, often used to grant different priorities or capabilities (e.g., longer runtimes).

Compute Node:
Worker nodes where jobs are executed. Access is allocated by the scheduler.
Environment Modules:
Software that allows users to dynamically load pre-installed applications and their dependencies into their shell environment.
Partition:
A queue of nodes with specific resource limits (max time, cores/GPU).
SBATCH Script:
A shell script containing directives for Slurm (`#SBATCH` lines) and commands to run your software.

Technical Specifications of Compute Node:

OEM & Model	**DELL PowerEdge R7525**
CPU:	**2× AMD EPYC 7742 (128 cores total per node)**
RAM:	**256 GB DDR4 memory per node**
OS:	**Rocky Linux 8.10**
Kernel:	**4.18.0-553.el8_10.x86_64**
NVIDIA Driver:	**570.158.01**
GPU:	**2× NVIDIA A100 80GB GPUs per node**
CUDA Cores:	**6,912 per GPU**
Tensor Cores:	**432 per GPU**
Total VRAM & CUDA Cores per Node:	**160 GB VRAM; 13,824 CUDA cores**
Workload Manager	**Slurm 23.11.11 (https://slurm.schedmd.com/)**

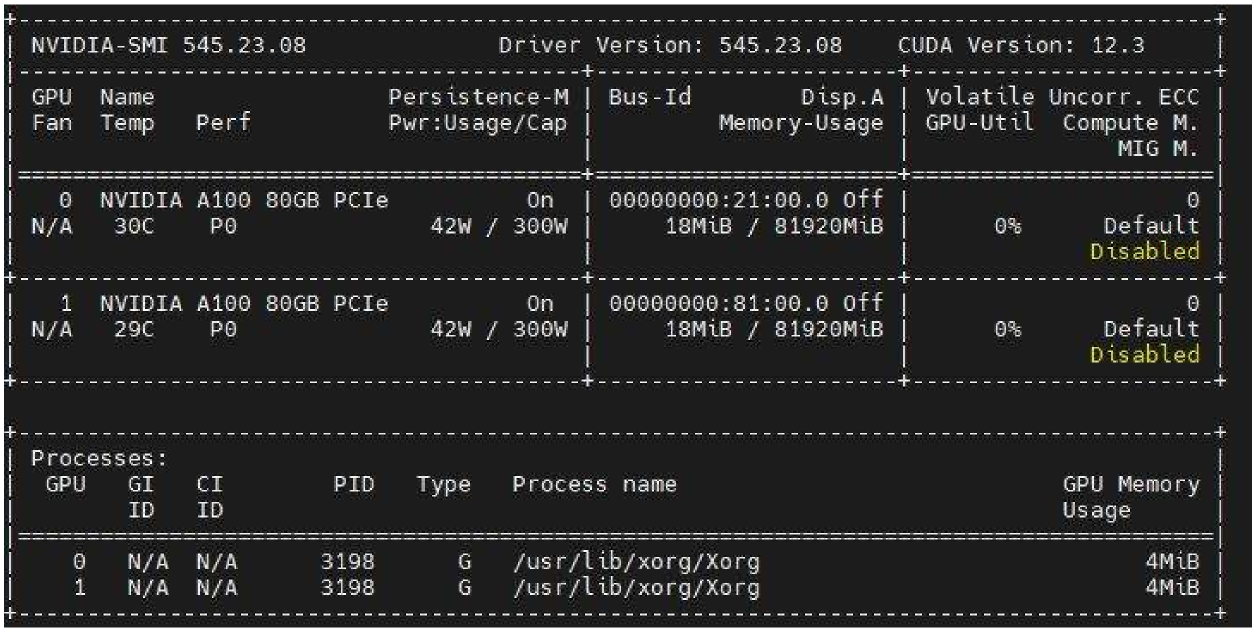


DELLEMC System Setup

Processor Settings

System BIOS Settings • Processor Settings

PROCESSOR 1	
Family-Model-Stepping	17-31-0
Brand	AMD EPYC 7502 32-Core Processor
Level 2 Cache	32x512 KB
Level 3 Cache	128 MB
Number of Cores	32
Microcode	0x8301052
PROCESSOR 2	
Family-Model-Stepping	17-31-0
Brand	AMD EPYC 7502 32-Core Processor
Level 2 Cache	32x512 KB



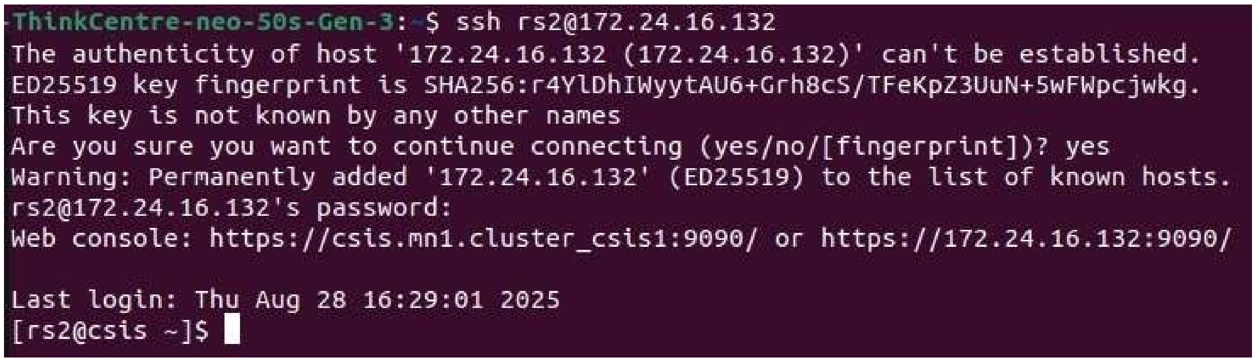
```
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 545.23.08              Driver Version: 545.23.08    CUDA Version: 12.3     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA A100 80GB PCIe          On  | 00000000:21:00.0 Off |                    0 |
| N/A   30C    P0              42W / 300W |      18MiB / 81920MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA A100 80GB PCIe          On  | 00000000:81:00.0 Off |                    0 |
| N/A   29C    P0              42W / 300W |      18MiB / 81920MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A      3198      G   /usr/lib/xorg/Xorg                            4MiB |
|    1   N/A  N/A      3198      G   /usr/lib/xorg/Xorg                            4MiB |
+---------------------------------------------------------------------------------------+
```

Procedure: -

Step 1: Secure Connection (Login) to the Cluster

I. Open your terminal (or SSH client).
II. Connect using the following command:
ssh <username>@172.24.16.132
III. Enter your password and any 2FA token when prompted.



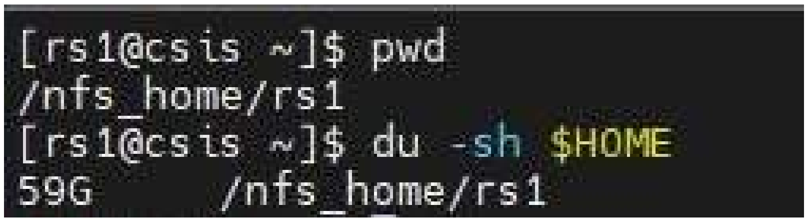
```
ThinkCentre-neo-50s-Gen-3:~$ ssh rs2@172.24.16.132
The authenticity of host '172.24.16.132 (172.24.16.132)' can't be established.
ED25519 key fingerprint is SHA256:r4YlDhIWyytAU6+Grh8cS/TFeKpZ3UuN+5wFWpcjwkg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.24.16.132' (ED25519) to the list of known hosts.
rs2@172.24.16.132's password:
Web console: https://csis.mn1.cluster_csis1:9090/ or https://172.24.16.132:9090/

Last login: Thu Aug 28 16:29:01 2025
[rs2@csis ~]$
```

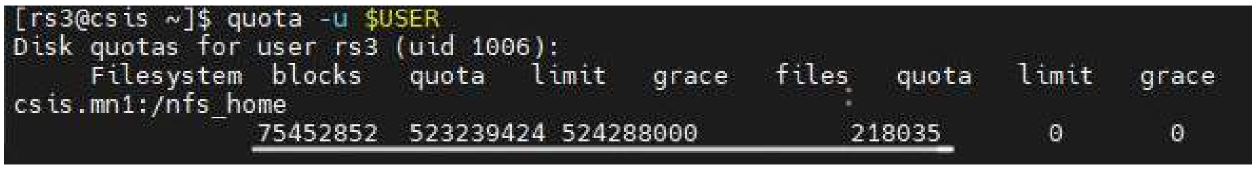
Step 2: Understand the Environment

Initially, users will be on a master node as they successfully make their login. User is supposed to execute the job by preparing and using Slurm files, not directly on the node's terminal console. Each user has a **500 GB** quota limit for their home directory. It may be extended as the cluster is getting more disk availability.

- To check the home directory usage details
  du -sh $HOME
  du -sh /nfs_home/username
- To check the allotted home directory size details
  quota -u $USER
  quota -u /nfs_home/username



```
[rs1@csis ~]$ pwd
/nfs_home/rs1
[rs1@csis ~]$ du -sh $HOME
59G     /nfs_home/rs1
```



```
[rs3@csis ~]$ quota -u $USER
Disk quotas for user rs3 (uid 1006):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
csis.mn1:/nfs_home
                75452852  523239424 524288000          218035       0       0
```

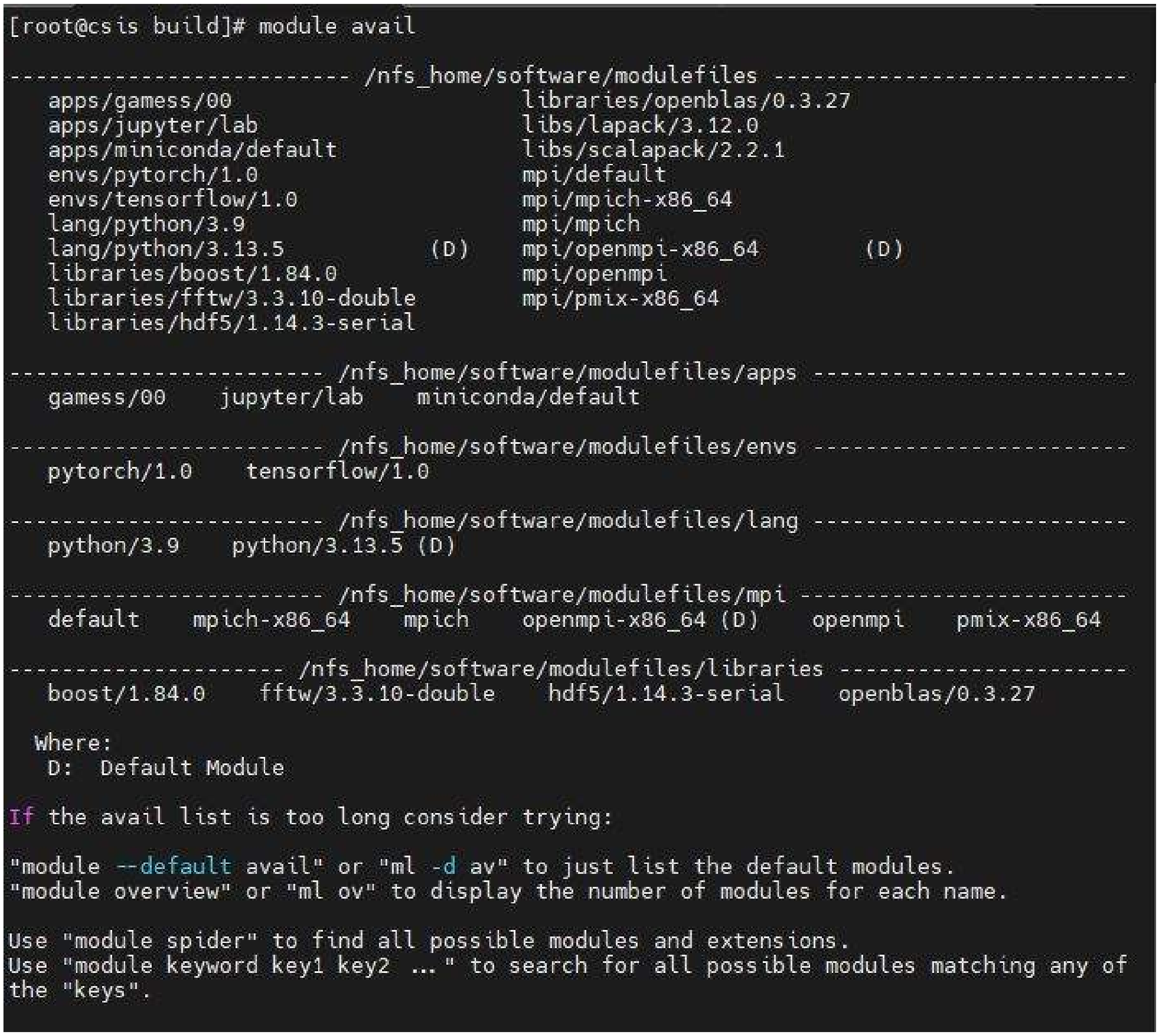
In the above fig, cited user has used around 75GB, **quota** (Soft Limit:- the **warning threshold**) is 499GB, and **limit** (Hard) 500GB

NB: - If the allotted user quota is near or exceeded with stored data, the user must archive or delete files before new jobs can run or data can be saved.

Step 3: Explore Available Software Modules and Resources

A key software stack is pre-installed and managed via Environment Modules.
This simplifies loading complex software and its dependencies.

Key Available Modules Include:



```
[root@csis build]# module avail

-------------------------- /nfs_home/software/modulefiles ---------------------------
   apps/gamess/00                       libraries/openblas/0.3.27
   apps/jupyter/lab                     libs/lapack/3.12.0
   apps/miniconda/default               libs/scalapack/2.2.1
   envs/pytorch/1.0                     mpi/default
   envs/tensorflow/1.0                  mpi/mpich-x86_64
   lang/python/3.9                      mpi/mpich
   lang/python/3.13.5          (D)      mpi/openmpi-x86_64       (D)
   libraries/boost/1.84.0               mpi/openmpi
   libraries/fftw/3.3.10-double         mpi/pmix-x86_64
   libraries/hdf5/1.14.3-serial

------------------------ /nfs_home/software/modulefiles/apps ------------------------
   gamess/00    jupyter/lab    miniconda/default

------------------------ /nfs_home/software/modulefiles/envs ------------------------
   pytorch/1.0    tensorflow/1.0

------------------------ /nfs_home/software/modulefiles/lang ------------------------
   python/3.9    python/3.13.5 (D)

------------------------ /nfs_home/software/modulefiles/mpi -------------------------
   default    mpich-x86_64    mpich    openmpi-x86_64 (D)    openmpi    pmix-x86_64

--------------------- /nfs_home/software/modulefiles/libraries ----------------------
   boost/1.84.0    fftw/3.3.10-double    hdf5/1.14.3-serial    openblas/0.3.27

  Where:
   D:  Default Module

If the avail list is too long consider trying:

"module --default avail" or "ml -d av" to just list the default modules.
"module overview" or "ml ov" to display the number of modules for each name.

Use "module spider" to find all possible modules and extensions.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of
the "keys".
```

Short Notes about available Modules

1) **lang/ (Programming Languages)**
    - **python/3.9**: The Python 3.9 interpreter and core libraries. Useful for projects that require compatibility with this specific, older version.
    - **python/3.13.5 (D)**: The default Python 3.13.5 interpreter. This is the latest version provided by the system and is the one that will load if you just type `module load lang/python`.
2) **envs/ (Pre-configured Environments)**
    - **pytorch/1.0**: A pre-installed environment for the PyTorch 1.0 deep learning framework. Loading this module gives you immediate access to this specific version of PyTorch and its dependencies without needing to install it yourself.
    - **tensorflow/1.0**: A pre-installed environment for the TensorFlow 1.0 deep learning framework. Useful for running older code designed for the TF1 API, which is significantly different from modern TF2.
3) **mpi/ (Message Passing Interface Libraries)**
    - **openmpi-x86_64 (D)**: The default version of the OpenMPI library, compiled for x86_64 architecture. This is the most common high-performance MPI implementation for distributed computing and parallel jobs.
    - **openmpi**: Likely a symbolic link or alternative version of the OpenMPI library.
    - **mpich-x86_64**: The MPICH implementation, another high-performance and widely portable MPI library, compiled for x86_64.
    - **mpich**: Likely a symbolic link or alternative version of the MPICH library.
    - **pmix-x86_64**: The Process Management Interface (PMIx) library, often used as a runtime for MPI implementations and other tools in resource-managed environments (like Slurm).
    - **default**: A fallback module that likely points to the system's default MPI implementation (probably `openmpi-x86_64`).
4) **libraries/ (Scientific & Numerical Libraries)**
    - **openblas/0.3.27**: The OpenBLAS library, an optimized implementation of the BLAS (Basic Linear Algebra Subprograms) API. Dramatically accelerates numerical and linear algebra operations in code like NumPy, R, and many scientific applications.
    - **LAPACK**: Built upon BLAS, this library provides higher-level routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems, forming the core of many quantum chemistry algorithms.
    - **ScaLAPACK** (Scalable LAPACK): A parallel extension of LAPACK for distributed-memory MIMD parallel computers. Building this library was critical for enabling the parallel execution of linear algebra operations across multiple compute nodes, which is necessary for studying large molecular systems with applications like GAMESS.

- **fftw/3.3.10-double**: The FFTW (Fastest Fourier Transform in the West) library, version 3.3.10, compiled for double-precision calculations. The standard library for high-performance Fourier transforms.
- **hdf5/1.14.3-serial**: The HDF5 library, version 1.14.3, compiled for serial (non-parallel) use. Used for managing complex data structures and storing large amounts of numerical data.
- **boost/1.84.0**: The Boost C++ source libraries, version 1.84.0. A large collection of peer-reviewed, portable C++ libraries that provide support for tasks like linear algebra, multithreading, and graph algorithms.

5) apps/ (Applications & Tools)

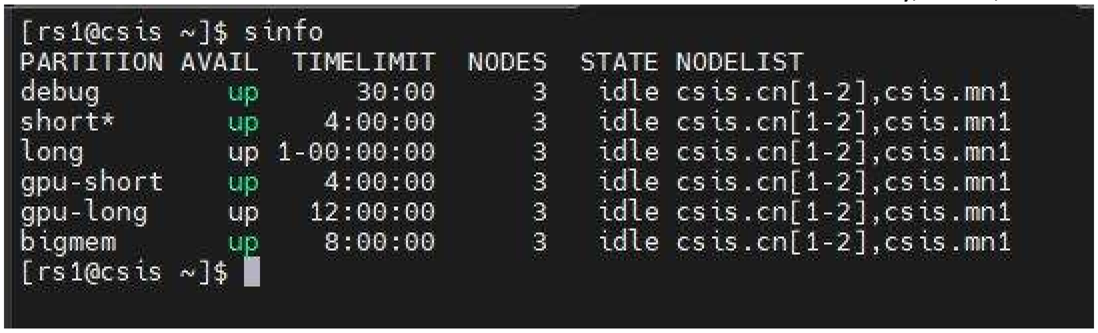
- **miniconda/default**: Provides the `conda` package manager and a minimal base Python environment. Essential for creating and managing your own isolated software environments for Python or R, allowing you to install specific package versions without requiring system administrator privileges.
- **jupyter/lab**: For launching JupyterLab sessions. This module is typically used with a job scheduler (like Slurm) to start an interactive session on a compute node, providing a web-based development environment for code, data, and documentation.
- **GAMESS** (General Atomic and Molecular Electronic Structure System)
  It is a quantum chemistry software package. This is a comprehensive computational chemistry program for ab initio molecular orbital calculations, including methods such as Hartree-Fock, DFT, MP2, CCSD(T), and many other advanced quantum mechanical methods. Essential for researchers in computational chemistry, materials science, and molecular modeling to perform electronic structure calculations, geometry optimizations, frequency calculations, and molecular property predictions.

Step 4: How to View and Load Available Software Modules:

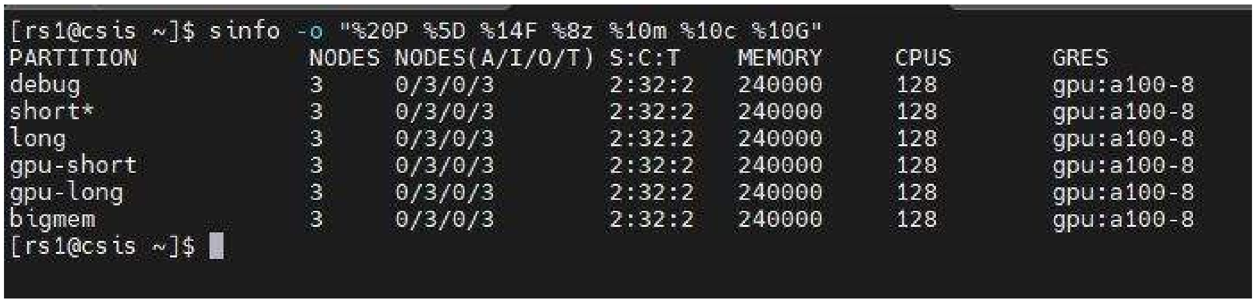
• module avail	# View all available modules
• module avail **python**	# Search for specific modules (e.g., python)
• module show lang/python/3.13.5	# Get information about a specific module
• module load **miniconda/default**	# Load the default Miniconda module *Mention the required module name explicitly for hassle-free setup.
• module load **jupyter/lab**	# Load JupyterLab for interactive work
• module **list**	# See currently loaded modules
• module unload **jupyter/lab**	# To Unload a specified module
• module **purge**	# Unload all modules

Step 5: How to View Cluster Specifications & Partitions:

- sinfo — # Overview of all partitions and node states
- sinfo -o "%20P %5D %14F %8z %10m %10c %10G" — # Detailed view: Partition, Nodes, State, Memory, CPUs, GPUs



```
[rs1@csis ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up      30:00      3   idle csis.cn[1-2],csis.mn1
short*       up    4:00:00      3   idle csis.cn[1-2],csis.mn1
long         up 1-00:00:00      3   idle csis.cn[1-2],csis.mn1
gpu-short    up    4:00:00      3   idle csis.cn[1-2],csis.mn1
gpu-long     up   12:00:00      3   idle csis.cn[1-2],csis.mn1
bigmem       up    8:00:00      3   idle csis.cn[1-2],csis.mn1
[rs1@csis ~]$
```



```
[rs1@csis ~]$ sinfo -o "%20P %5D %14F %8z %10m %10c %10G"
PARTITION            NODES NODES(A/I/O/T) S:C:T    MEMORY     CPUS       GRES
debug                3     0/3/0/3        2:32:2   240000     128        gpu:a100-8
short*               3     0/3/0/3        2:32:2   240000     128        gpu:a100-8
long                 3     0/3/0/3        2:32:2   240000     128        gpu:a100-8
gpu-short            3     0/3/0/3        2:32:2   240000     128        gpu:a100-8
gpu-long             3     0/3/0/3        2:32:2   240000     128        gpu:a100-8
bigmem               3     0/3/0/3        2:32:2   240000     128        gpu:a100-8
[rs1@csis ~]$
```

Step 6: How to View GPU Specifications

• sinfo -o "%20N %10G" # Shows gpu:a100-80gb:2	# To see physical GPU count per node
• nvidia-smi	# To see GPU utilization, including memory
• nvidia-smi -q | grep -E "(Product Name|FB Memory|Count)"	# To see detailed GPU properties

Step 7:
Create a Job Script with Correct Partitions and QoS. The user may refer to the table below to know the most effective parameters with respect to the proposed job.

<u>QOS to Partition Mapping</u>

QOS Name	Max Time	Max CPUs/Job	Max Memory/Job	Max GPUs/Job
debug	**30 min**	**16 CPUs**	**48GB**	**0 GPUs**
short	**4 hours**	**32 CPUs**	**96GB**	**0 GPUs**
long	**24 hours**	**16 CPUs**	**48GB**	**0 GPUs**
gpu-short	**4 hours**	**16 CPUs**	**64GB**	**2 GPUs**
gpu-long	**12 hours**	**8 CPUs**	**80GB**	**1 GPU**
bigmem	**8 hours**	**64 CPUs**	**192GB**	**0 GPUs**

<u>Setup Instructions for Users:</u>

1) Create a job script (`my_job.sh`) that requests appropriate resources and loads necessary modules.

```
nano my_conda_job.sbatch
```

2) Make it executable:

```
chmod +x my_conda_job.sbatch
```

3) Submit the job:

```
sbatch my_conda_job.sbatch
```

4) Monitor the job progress:

- `squeue -u $USER` # View status of your jobs
- `sinfo -o "%20P %10a %10c %10m %10G %60"` # Show resource utilization (CPU, memory, GPU loads)
- `tail -f slurm-<jobid>.out` #Check the output file of the job in progress
- `tail -f slurm-<jobid>.err` #Check the error file of the job in progress
- `cat slurm-<jobid>.out` #Check the output file of the completed job
- `cat slurm-<jobid>.err` #Check the error file of the completed job

5) To know the completed job, to cancel the ongoing job, and to know about pending jobs:

- seff <jobid> # To get a detailed efficiency report after the job completes
- scancel <jobid> # Cancel a pending or running job
- squeue -t pending -o "%.8i %.12P %.20j %.8u %.2t %.10M %.6D %25R" # See all pending jobs and their reasons
- squeue -t pending --start # See estimated start times for pending jobs

6) Quick job submission examples:

- sbatch --partition=debug --time=00:30:00 --mem=16G script.sh
- sbatch --partition=short --time=04:00:00 --gres=gpu:1 script.sh
- sbatch --partition=long --time=24:00:00 --mem=48G script.sh
- sbatch --partition=bigmem --mem=180G script.sh

7) The user may leverage the below given Job Scripts to use the available modules:

1. Basic CPU Job with Miniconda (short partition)

```
#!/bin/bash

#SBATCH --job-name=conda_job

#SBATCH --partition=short

#SBATCH --time=04:00:00

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=8

#SBATCH --mem=32G

#SBATCH --output=conda_%j.out

#SBATCH --error=conda_%j.err


# Initialize conda from NFS shared installation

source /nfs_home/software/miniconda/etc/profile.d/conda.sh


# Activate your environment

conda activate my_env


# Verify environment

echo "Conda environment: $CONDA_DEFAULT_ENV"

echo "Python path: $(which python)"


# Your code here

python my_script.py"=== Job Completed ==="
```

2. GPU Job with Miniconda (gpu-short partition)

```
#!/bin/bash

#SBATCH --job-name=conda_gpu_job

#SBATCH --partition=gpu-short

#SBATCH --gres=gpu:a100-80gb:1

#SBATCH --time=04:00:00

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=8

#SBATCH --mem=32G

#SBATCH --output=gpu_conda_%j.out

#SBATCH --error=gpu_conda_%j.err

echo "=== GPU Job Started ==="

echo "Job ID: $SLURM_JOB_ID"

echo "GPU allocated: $CUDA_VISIBLE_DEVICES"

# Load required modules

module load miniconda3/4.12.0

module load cuda/11.7

# Initialize conda

source /nfs_home/software/miniconda/etc/profile.d/conda.sh

# Activate environment with GPU support

conda activate pytorch_gpu

# Verify GPU access

echo "Python: $(which python)"

nvidia-smi

python -c "import torch; print(f'CUDA available: {torch.cuda.is_available()}'); print(f'GPU count:
{torch.cuda.device_count()}')"

# GPU-accelerated script

python train_model.py --batch-size 32 --epochs 10 --gpu-id 0


echo "=== GPU Job Completed ==="
```

3. Debug Job with Miniconda (debug partition)

```
#!/bin/bash

#SBATCH --job-name=conda_debug

#SBATCH --partition=debug

#SBATCH --time=00:30:00

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=4

#SBATCH --mem=16G

#SBATCH --output=debug_%j.out

#SBATCH --error=debug_%j.err

echo "=== Debug Job ==="

echo "Quick test of conda environment"

# Source conda (adjust path as needed)

source /nfs_home/software/miniconda/etc/profile.d/conda.sh

# List available environments

echo "Available conda environments:"

conda env list

# Test basic environment

conda activate base

python -c "

import sys

print(f'Python {sys.version}')

import numpy as np

print(f'NumPy {np.__version__}')

print('Debug test completed successfully')

"

echo "=== Debug Completed ==="
```

4. Long-running CPU Job with Miniconda (long partition)

```
#!/bin/bash
#SBATCH --job-name=conda_long_job
#SBATCH --partition=long
#SBATCH --time=1-00:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=12
#SBATCH --mem=40G
#SBATCH --output=long_%j.out
#SBATCH --error=long_%j.err

echo "=== Long-running Job ==="
echo "Duration: 24 hours"

# Setup conda
source /nfs_home/software/miniconda/etc/profile.d/conda.sh
conda activate research_env

# Long-running analysis
python extended_analysis.py \
  --threads $SLURM_CPUS_PER_TASK \
  --memory $(($SLURM_MEM_PER_NODE / 1024))G

echo "=== Long Job Completed ==="
```

5. Memory-Intensive Job with Miniconda (bigmem partition)

```
#!/bin/bash
#SBATCH --job-name=conda_bigmem
#SBATCH --partition=bigmem
#SBATCH --time=08:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=32
#SBATCH --mem=180G
#SBATCH --output=bigmem_%j.out
#SBATCH --error=bigmem_%j.err

echo "=== Big Memory Job ==="
echo "Allocated memory: $SLURM_MEM_PER_NODE MB"

# Setup conda for memory-intensive work
source /nfs_home/software/miniconda/etc/profile.d/conda.sh
conda activate bigdata_env

# Memory-intensive processing
python process_large_data.py \
  --workers $SLURM_CPUS_PER_TASK \
  --memory $(($SLURM_MEM_PER_NODE / 1024))G

echo "=== Big Memory Job Completed ==="
```

6. Multi-GPU Job with Miniconda (gpu-short partition)

```
#!/bin/bash

#SBATCH --job-name=conda_multi_gpu

#SBATCH --partition=gpu-short

#SBATCH --gres=gpu:a100-80gb:2

#SBATCH --time=04:00:00

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=16

#SBATCH --mem=64G

#SBATCH --output=multi_gpu_%j.out

#SBATCH --error=multi_gpu_%j.err

echo "=== Multi-GPU Job ==="

echo "GPUs allocated: $CUDA_VISIBLE_DEVICES"

# Setup conda with GPU support

source /nfs_home/software/miniconda/etc/profile.d/conda.sh

conda activate multi_gpu_env

# Verify multi-GPU setup

python -c "

import torch

print(f'Available GPUs: {torch.cuda.device_count()}')

for i in range(torch.cuda.device_count()):

    print(f'GPU {i}: {torch.cuda.get_device_name(i)} -
{torch.cuda.get_device_properties(i).total_memory/10243:.1f}GB')

"

# Multi-GPU training

python -m torch.distributed.launch --nproc_per_node=2 \

    train_multi_gpu.py --batch-size 64 --epochs 20


echo "=== Multi-GPU Job Completed ==="
```

Key Features of These Included Templates:

Users can copy the above templates and modify the required module environment name, Python/Bash script, and parameters as needed for their specific workloads.

1. **Choose the right partition** for their needs
2. **Request appropriate resources** (not too much, not too little)
3. **Stay within time limits** to avoid job termination
4. **Use GPUs efficiently** with proper memory/CPU ratios
5. **Leverage all available partitions** based on their account privileges

8) Getting Help and Support:

- ✓ Users may email their queries or requests for additional module expansions to the support team.
- ✓ In case of any technical error, users should include the corresponding job ID and the relevant section of the slurm-.out file in their email.
- ✓ Support can be reached through the following channels:

- **Email:** vincem@pilani.bits-pilani.ac.in
- **In-Person Assistance:** CSIS Lab (Room 6017).