

# Introduction to Rockfish Cluster

[Help](#)

[High Performance Computing](#)

[Allocations on Rockfish](#)

[Connecting to Rockfish](#)

[Data Transfer](#)

[Module and Software System](#)

[Parallel Computing Jobs](#)

[Job Management by SLURM](#)

[Container and Singularity](#)

[Python and Anaconda](#)

[RStudio](#)

[Jupyter Notebook](#)

[Help](#)

The Advanced Research Computing at Hopkins (ARCH) –formerly known as MARCC– is a shared computing facility at Johns Hopkins University that enables research, discovery, and learning, relying on the use and development of advanced computing. See [About ARCH](#).

Rockfish website: <https://www.arch.jhu.edu/>

Rockfish User Guide: <https://www.arch.jhu.edu/access/user-guide/>

# Help

**Note: We will have maintenance on 4/11/2022 to 4/15/2022.**

## RockFish Web Sites

Read our [User Guide](#) web site for how to use our system or look for specific information.

## RockFish RT System

Send us your questions at [help@rockfish.jhu.edu](mailto:help@rockfish.jhu.edu) and include as much information as possible.  
For example:

- The jobid of the job with problems
- Full path to the batch submission script
- Any specific error messages
- If possible a snapshot with errors

## Frequently Asked Questions

We also have a [FAQ](#) web page where you could find the answers of your questions which most of the users asked.

# High Performance Computing

[HPC Terminology](#)

[Why Use ARCH](#)

## HPC Terminology

### Node

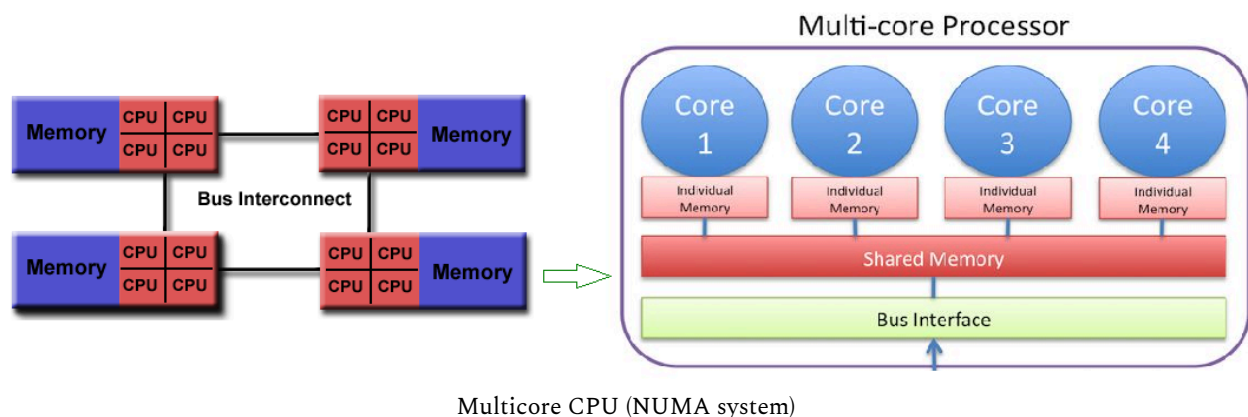
A standalone "computer in a box". Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc.

### Cluster

A group of nodes networked together so a program can run on them in parallel.

### CPU/Processor(Socket)/Core

In the past, a **CPU** (Central Processing Unit) was a singular execution component for a computer. Then, multiple-core CPU is incorporated into a node. It is subdivided into multiple "**cores**" inside processors (or sockets). Each core is a unique execution unit like a CPU in the past. RAM memory between different sockets is connected with a bus interface.



### GPU

Short for Graphics Processing Unit. It is a specialized processor with thousands of small CPU cores. It can run multiple processes and perform many pieces of data in parallel. It is useful for machine learning, video editing, and gaming applications.

### Task/Process

A **process** or **running process** refers to a set of programmed instructions currently being processed by the computer CPU. A process may be made up of multiple threads of execution that execute instructions concurrently.

## Thread

In computer science, a *thread* of execution is the smallest sequence of instructions (from an application) that can be managed independently by a single core.

## SSH

Also known as secure shell. It is used to log into a remote machine and execute commands through a cryptographic network protocol.

## File System

System for storing many files. A local file system is a file system directly connected to a node, such as a hard drive in a computer (in /tmp). **Files saved in a local file system can only be accessed directly through the connected node.** A network file system (NFS) is a distributed file system protocol allowing a user on a client computer to access files over a computer network. **Files saved in a NFS can be directly accessed through the computers in the network.**

## Why Use ARCH

A comparison between your pc and our HPCC:

### Hardwares

	Laptop/Desktop	Rockfish Clusters
Number of Nodes	1	693
Sockets per node	1	2
Cores per node	4 - 16	48
Cores total	4, 8, or 16	33,264
Core Speed	2.7 - 3.5 GHz	3 GHz
RAM memory	8 - 32GB	189GB or 1.4TB
File Storage	250GB - 1TB	50GB(Home), 10TB(data), 10TB(Scratch)
Connection to other computers	Campus ethernet 1 Gbit/sec	"Infiniband"100 Gbit/sec
Users	1	~800
Schedule	On Demand	24/7 via queue

\* Parallel is mostly the key to use ARCH.

### Software

You can use various software with different versions installed in HPCC:

- Compilers — GNU, intel, CUDA, ...
- Parallel — OpenMPI, Intel-MPI, ...
- Bioinformatics — BLAST, Trinity, Mothur, Samtools, Trimmomatic, ...
- Libraries — MKL, OpenBLAS, HDF5ls, FFTW, ...
- Commercial — MATLAB, ABINIT, COMSOL, TotalView, ...

# **Allocations on Rockfish**

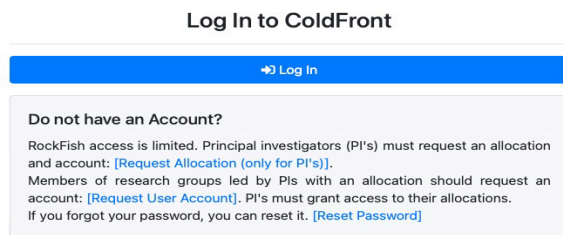
[Account and Allocation Management](#)

[Computing Allocations](#)

## Account and Allocation Management

### Allocation and User Account

The [Rockfish portal](#) (Coldfront) can be used by PIs and users to manage accounts and allocations. PIs can create projects with a short proposal; request allocations (which must be approved by the advanced computing committee); add or remove users to their allocations; upload publications, grants and other items. All projects must submit periodic reports via the Rockfish portal or to XSEDE. Users can see the [video link](#) for how to use Coldfront.



### Account Usage

Rockfish allocations are made in core-hours. The recommended method for estimating your resource needs for an allocation request is to perform benchmark runs. The core-hours used for a job are calculated by multiplying the number of processor cores used by the wall-clock duration in hours.

For example: if you request one core on one node for an hour your allocation will be charged one core-hour. If you request 24 cores on one node, and the job runs for one hour, your account will be charged 24 core-hours. Job accounting is independent of the number of processes actually used on compute nodes. You can request 2 cores for your job for one hour. If you run only one process, your allocation will still be charged for 2 core-hours.

Charge = Number of cores requested x wall-time.

*Note: Each GPU is associated with 12 cores.*

• **Allocations will be reset on a quarterly basis. The model is “use it or lose it”. Unused allocations or part of them will not roll over to the next quarter. Users can use the `sbalance` command to check their usage:**

```
[userid@login01 ~]$ sbalance slurm_account userid
[BALANCE] used/quarter, account:      slurm_account  7073.2 / 100000.0 (SU)
[BALANCE] used/quarter, user:         userid      4077.6 / 100000.0 (SU)
```



## Computing Allocations

### MRI-Related Allocations

- PIs who participated in the proposal to NSF should submit a short [proposal](#) justifying the request for cpu-cycles and storage.
- Regular cpu-cycles. This allocation will allow the use of the regular/standard compute nodes. The limit is 750,000 core-hours per quarter.
- Large memory cpu-cycles. This allocation will allow the use of large memory nodes. Applications and simulations must use more than 192GB of memory. Limit 50,000 core-hours per quarter.
- GPU allocation. This allocation will allow the use of GPU nodes. All scientific applications should make use of the GPUs. Limit 50,000 core-hours.
- Storage (GPFS) allocations. This allocation will assign storage to the group on an annual basis.

### Condo-Related Allocations

- **Deans' Condo allocations**
  - The Dean's office will offer small/startup faculty members allocations. These allocations WILL follow the MRI-topics and will be part of the annual report to the funding agency.
- **Morgan State University allocations**
  - A subset of allocations will follow a model developed at Morgan State University as a partner for this grant.
- **XSEDE allocations**
  - Follows the process established for XSEDE allocations (XRAC)
- PIs who contribute condos will have an allocation (or additional allocation) equivalent to the size of the condo. For example a one node condo is equivalent to 100,000 core-hours per quarter.
- Condo storage and purchased storage. This allocation is composed of the storage given by default plus the size of the storage purchased. It may cover different file sets.
- New faculty members will have allocations equivalent to the size of the condo in their LOI, or as offered by the Dean's office.
- As of April 15, 2022 all "active" PIs using the Bluecrab cluster have a startup allocation on Rockfish (50,000 hours). Use this trial access to benchmark your codes and obtain information to submit a proposal by November 2022.

Please refer to [Allocations](#) web site.

## Connecting to Rockfish

[SSH Client](#)

[Access to login Node](#)

[Check Storage and Account Allocations](#)

## SSH Client

In order to use the Rockfish system robustly by command lines, please have a SSH client and X11 server installed in your computer.

### For Windows machines

You can use [PuTTY](#) or [MobaXterm](#) (Home Edition → Installer edition). MobaXterm provides both a SFTP application for file transfer and a SSH client for command lines with X-Windows (X11 server) system (for graphical user interface (GUI) running on Rockfish login nodes).

### For Mac OS machines

You can use Terminal program (installed within MacOS) for your SSH client. (In the taskbar, search for “terminal”.) However, for running graphical user interface (GUI) programs on Rockfish login nodes, the X11-server program XQuartz needs to be installed. See [XQuartz](#) for download instructions.

## Access to login Node

After you have a SSH client in your local computer, please open the application to run ssh commands. In the opened terminal, please type:

```
> ssh -XY userid@login.rockfish.jhu.edu
password:
```

and hit enter, where `userid` should be your Rockfish account name. Once you establish a connection, SSH will prompt for your password. Please type the password of your Rockfish user account. There will be nothing on your password typing. However, keep finishing your password and hit enter. Once sign into the login node, you should see the welcome messages:

```
Thu Mar 10 09:45:29 2022 from 172.28.3.75

|_ _ \ _ _ _ | | _ | _ ( _ ) _ | | _
| | ) / _ \ / _ | | / | _ | / _ | _ \
| _ < ( _ ) | ( _ | < | _ | | \ _ \ | |
| _ | \ _ \ / \ _ | _ | \ _ \ | _ | _ / _ | |

[STATUS] loading software modules
[STATUS] modules are Lmod (https://lmod.readthedocs.io/en/latest/)
[STATUS] software is Spack (https://spack.readthedocs.io/en/latest/)
[STATUS] the default modules ("module restore") use GCC 9 and OpenMPI 3.1
[STATUS] you can search available modules with: module spider <name>
[STATUS] you can list available modules with: module avail
[STATUS] loading a compiler, MPI, Python, or R will reveal new packages
[STATUS] and you can check your loaded modules with: module list --terse
[STATUS] to hide this message in the future: touch ~/.no_rf_banner
[STATUS] restoring your default module collection now
```

Quota and usage information. Updated hourly. Use 'gpfsquota'

Usage **for** group MyGroup

```
FS          Usage    Quota    Files_Count  File_Quota
---          ---      ---      ---          ---
data        5.07T    10T      2287948      4194304
scratch16   1.939T   10T      1005210      10240000
scratch4     4.177T   10T      1159700      20480000
[userid@login02 ~]$
```

Now you are on the login node of the Rockfish system. Please be aware: **All users are requested to abstain from running jobs and conduct minimal tasks on login nodes. Any extended testing should be done using the debug queue or an interactive session. Please see our [Policies](#) for more details.**

## Check Storage and Account Allocations

Right after access to a login node, users should reach their home space (/home/userid). By default, the results of the usage and quota of their file storages are also displayed. Users can also use `gpfsquota` command to get the results:

```
[userid@login02 ~]$ gpfsquota
Quota Usage for group PI-userid
FS      Usage  Quota  Files_Count  File_Quota
---      ---    ---    ---          ---
data     9.475T  10T    10789187     10240000
scratch16 2.44T   10T    2086208      10240000
scratch4  8.641T  10T    2355444      20480000
```

There should be four file storage places where you can access, read and write files:

File System Path	Size Quota (default)	File Number Quota (default)	Usage	Back-Up	System Type
/home/userid	50GB	N/A	long term file storage	weekly to an off-site location	NMVe SSD
/data/PI-userid	10TB	10240000	long term file storage	N/A	GPFS (parallel file system)
/scratch16/PI-userid	10TB	10240000	temporary large file storage	N/A	GPFS (parallel file system 16MB blocksize)
/scratch4/PI-userid	10TB	20480000	temporary small file storage	N/A	GPFS (parallel file system 4MB block size)

where “userid” is the user’s account name and PI-userid is the account name of the user’s PI. Users can use `cd` and `ls` commands to check if they can be accessed.

Users can use `sbalance` command to check the allocation and usage of their job accounts

```
[userid@login02 ~]$ sbalance PI-userid userid  
[BALANCE] used/quarter, account:      PI-userid 64406.4 / 143750.0 (SU)  
[BALANCE] used/quarter, user:         userid  9468.7 / 143750.0 (SU)
```

If there is anything wrong with the results of the usage and the allocation, please let us know with a message to [Rockfish HELP](#).

## Hardware Resources

Users can use `sinfo` command below to get the partitions and their compute nodes:

```
[userid@login02 ~]$ sinfo -s
PARTITION AVAIL  TIMELIMIT  NODES(A/I/O/T)  NODELIST
a100      up 3-00:00:00      12/2/3/17  gpu[02-09,13-21]
bigmem    up 2-00:00:00      5/10/7/22  bigmem[01-12,17-26]
defq*     up 3-00:00:00      492/66/95/653
c[001-332,337-381,383-424,433-499,501-520,522-524,529-576,625-720]
v100      up 3-00:00:00      0/1/0/1   gpu01
```

Here is the list of compute nodes and their properties:

Partition	Nodes	Processors	Cores	Memory	Local Disk	GPUs	Job Time Limit & Node List
<b>defq</b>	653	Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz	48	189 GB	915 GB	N/A	3 days c[001-332,337-381, 383-424,433-499,501-520,522-524,529-576,625-720]
<b>bigmem</b>	22	Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz	48	1.47 TB	816 GB	N/A	2 days bigmem[01-12,17-26]
<b>a100</b>	17	Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz	48	189 GB	1.4 TB	4*A100	3 days gpu[02-09,13-21]
<b>v100</b>	1	Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz	48	189 GB	1.4 TB	4*V100	3 days gpu01
<b>Total</b>	<b>693</b>		<b>33,264</b>			<b>68*A100 4*V100</b>	

## Data Transfer

[SFTP Application](#)

[Command Lines](#)

[Globus](#)



## SFTP Application

Please make sure you use Rockfish data transfer node: `rfdtn1.rockfish.jhu.edu` for the (remot) host. The following free SFTP applications are available to be installed for file transfer:

### MobaXterm

1. Download and install [MobaXterm](#) (Windows computer only).
2. Start the application. Click on **Sessions** then choose **New session**. In the pop-out window, click on **SFTP** and enter the information:
  - Remote host: `rfdtn1.rockfish.jhu.edu`
  - Username: <your userid>
  - Port: 22
3. Click **OK** You will be asked for password if it is not saved.
4. Once connected, the left column displays files in your local computer, the right column displays files of your Rockfish home.
5. You can select the appropriate directories by double clicking through each tree. Files can be dragged and dropped from one column to the other. (By dragging files from the left column to the right, they are uploaded from your local computer to Rockfish. By dragging files from the right column to the left, they are downloaded from Rockfish to your local computer.)

### FileZilla

1. Download and install [FileZilla](#).
2. Start the application. In the top dialog boxes, enter your information for each item:
  - Host: `rfdtn1.rockfish.jhu.edu`
  - Username: <your userid>
  - Password: <your password>
  - Port: 22
3. Click **connect** or **quickconnect**. The first time you use this, you will have to accept the host certificate.
4. Once connected, the left column displays files in your local computer, the right column displays files of Rockfish home.
5. You can select the appropriate directories by double clicking through each tree. Files can be dragged and dropped from one column to the other. (By dragging files from the left column to the right, you are uploading files to Rockfish from your local computer. By dragging files from the right column to the left, you can download files from Rockfish to your local computer.)

## Command Lines

### Download Data from Web Sites

Users can use `wget` or `curl` command to download data from a website to the Rockfish cluster. Just `cd` to the directory where you would like the data to be saved and execute the command. For example, by running the command:

```
curl -O http://www.examplesite.com/examplefile.txt
```

or

```
wget http://www.examplesite.com/examplefile.txt
```

the file `examplefile.txt` is downloaded to your current directory from the web site:

<http://www.examplesite.com/>.

### Copy Files by scp Command

Users can use `scp` command to copy a file or a directory from/to the Rockfish system to/from a local computer or another cluster. For example, the command

```
scp file-name userid@rfdtn1.rockfish.jhu.edu:/scratch16/PI-id/user-id/
```

copies `file-name` from the local computer (or the cluster) you are using to the directory `/scratch16/PI-id/user-id/` of Rockfish cluster. Since your local computer is not a server, please make sure you always run `scp` command on a terminal before you log into Rockfish cluster (rather than running `scp` on a Rockfish login node). To copy a directory, please specify `-r` option. For example, running the command:

```
scp -r userid@rfdtn1.rockfish.jhu.edu:/data/PI-id/user-id/MyDir ./
```

copies the directory `/data/PI-id/user-id/MyDir` in the Rockfish system to the current directory of your local computer (or the cluster you are using).

### Synchronize Files by rsync Command

With `rsync` command, you can copy and synchronize your files and directories between two different locations. It can be used for mirroring data and transferring only the differences between the source and the destination. The syntax for `rsync` command is similar to `scp` command. To mirror a directory `rockfish_dir` in Rockfish system to a local directory `local_dir`, use the command:

```
rsync -ave ssh userid@rfdtn1.rockfish.jhu.edu:rockfish_dir local_dir
```

To mirror a local directory `local_dir` to a directory `rockfish_dir` in Rockfish cluster, just switch their places in the above command line:

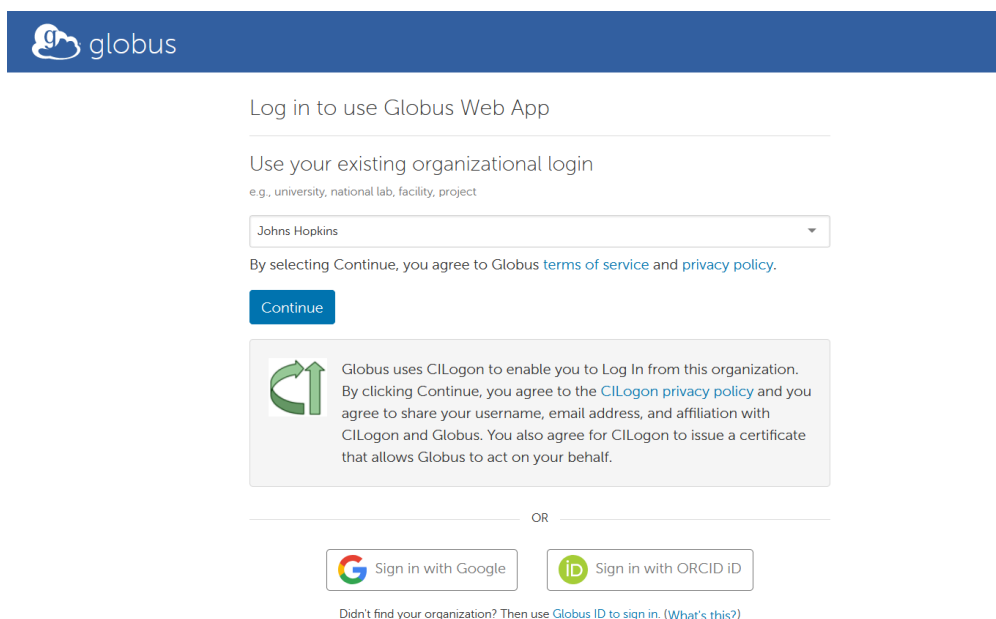
```
rsync -ave ssh local_dir userid@rfdtn1.rockfish.jhu.edu:rockfish_dir
```

Again, the command is considered to be used in the terminal of your local computer (or the cluster you are using) before accessing the Rockfish cluster.

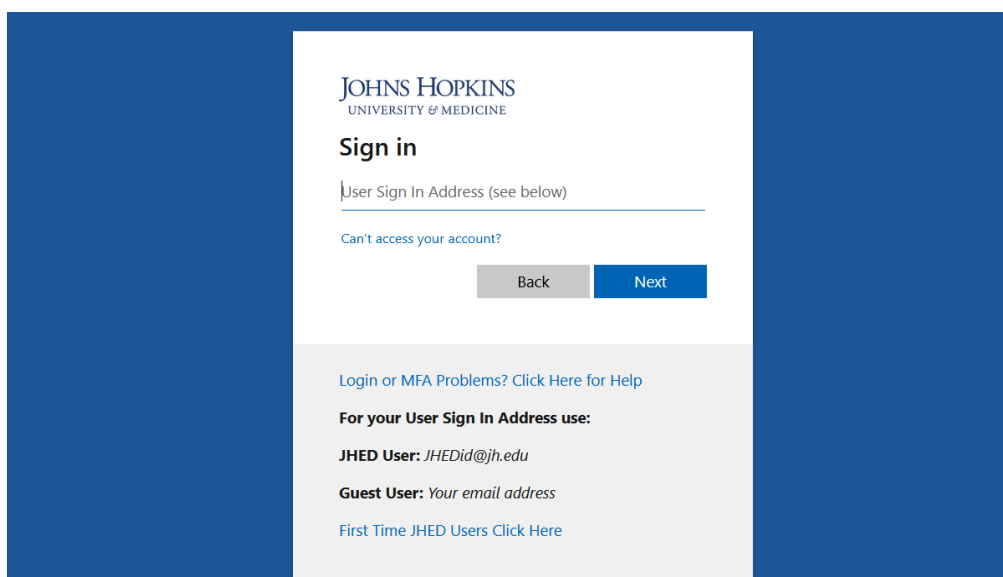
## Globus



The best way to transfer large data files to/from the Rockfish cluster is to use [Globus](https://app.globus.org). Globus manages all data transfers in the background, and makes sure the process goes through even with interruptions. To use Globus for data transfer, you can follow the steps below.

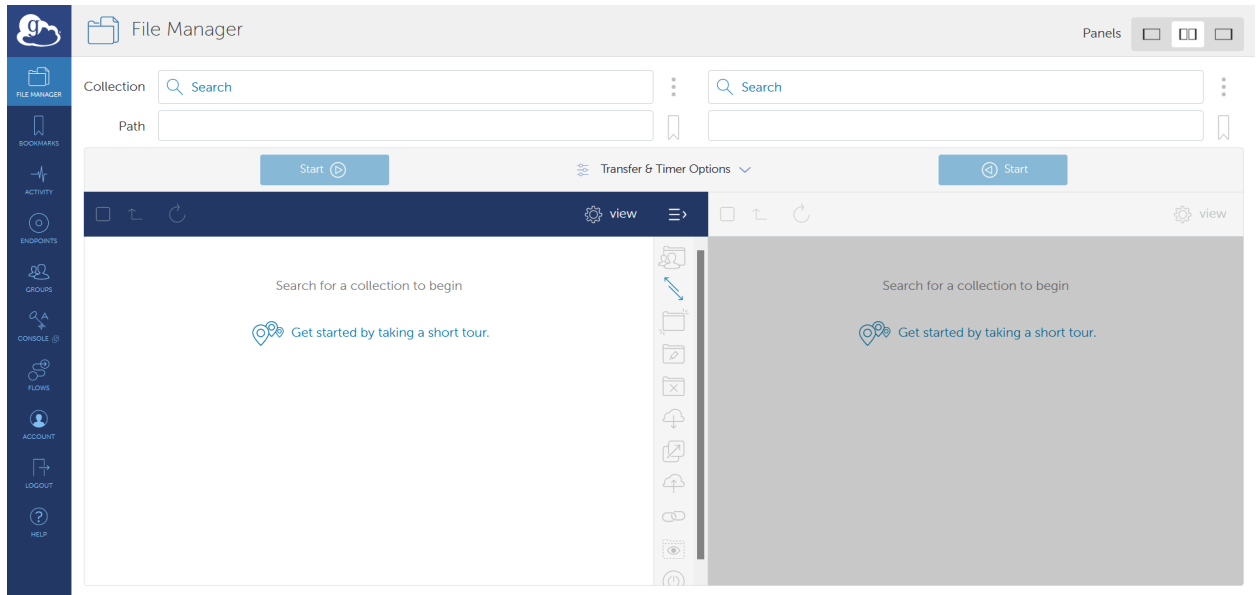
1. Use a web browser to connect to <https://app.globus.org>. If this is the first time you use Globus, you will be asked to sign into an organization. Please type and select “Johns Hopkins” then click **Continue**.



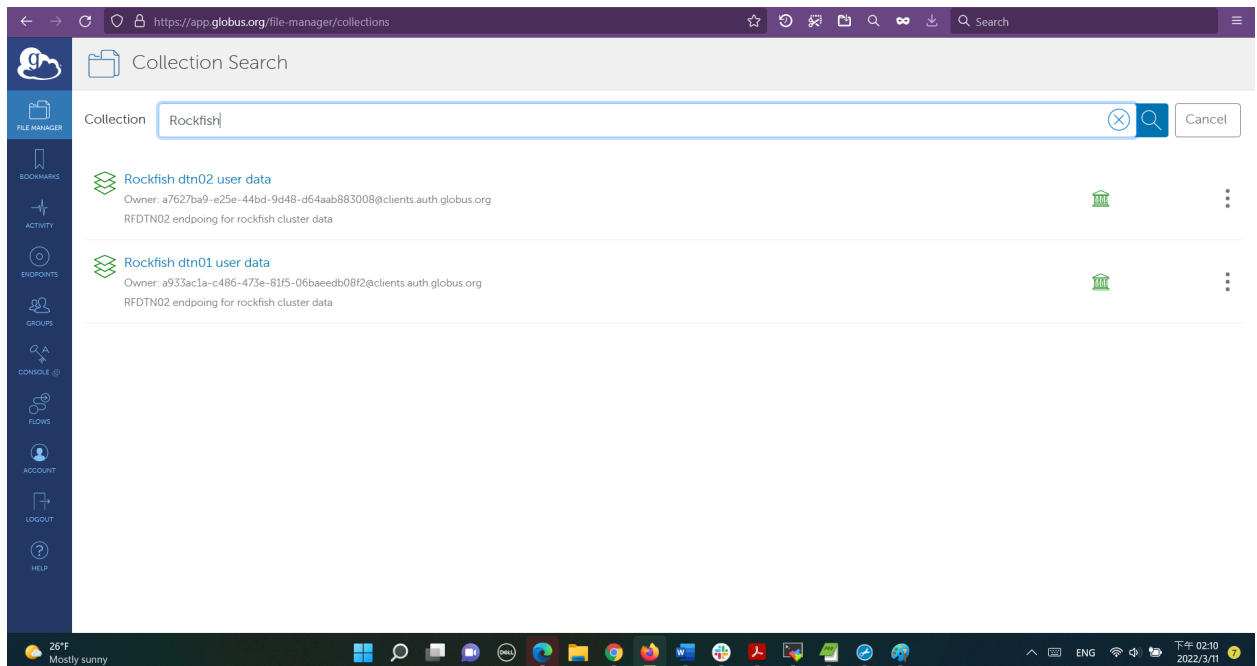
2. Sign in with your JHED ID.



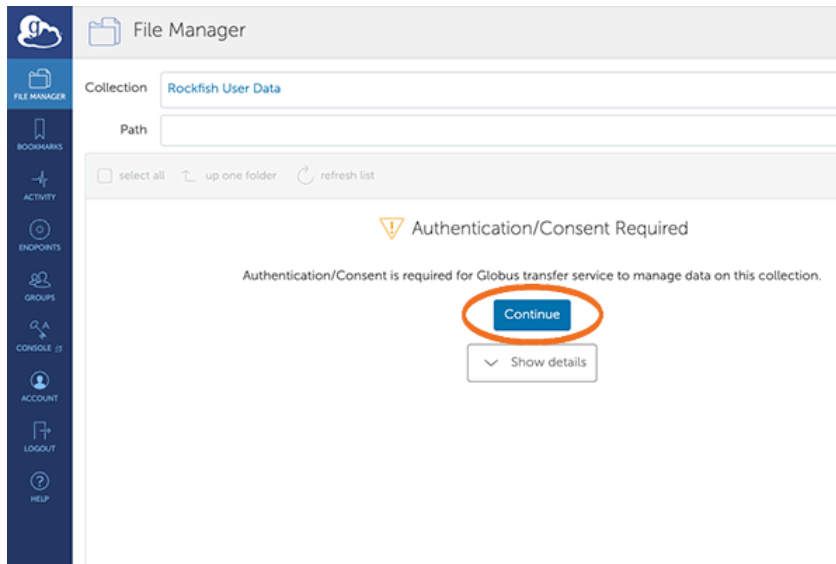
3. After you sign in successfully, you reach the Globus File Manager page. If you only get one column of **Collection**  Search in the screen. You can click  in **Panels** on the top right corner to get two columns.



4. Search for “Rockfish” in the Collection field.



- Click on **Rockfish dtn01 user data**. If this is the first time you access the collection, it will prompt for authentication. Click on **Continue**.



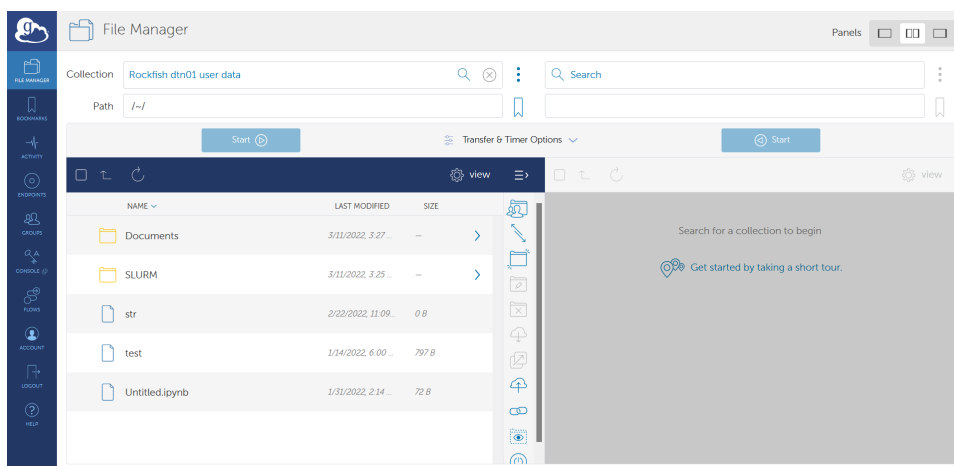
- Enter your Rockfish username and password.

**MyProxy Client Authorization**

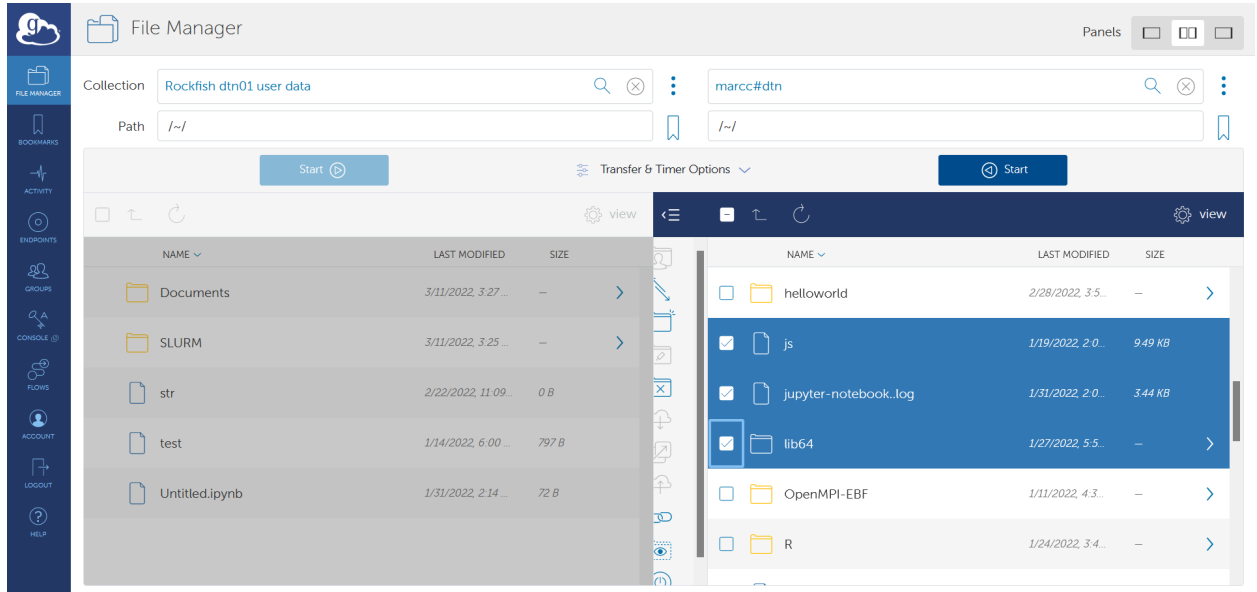
Welcome to the OAuth for MyProxy Client Authorization Page. The Client below is requesting access to your account. If you approve, please sign in with your username and password.

<b>Client Information</b>	<b>Username</b> <input type="text" value="userid"/>
Name: Globus URL: <a href="https://www.globus.org">https://www.globus.org</a>	<b>Password</b> <input type="password" value="....."/>
	<input type="button" value="Sign In"/> <input type="button" value="Cancel"/>

- After that, you access the file system of Rockfish cluster in the left column.



8. In the right column, you can search the endpoint to/from which you wish to transfer data in the Collection field. For example, you can use MARCC data transfer endpoint. Follow from step 3 again and search for "marcc" in the Collection field. After that, click on **marcc#dtn** and follow step 6 to access the file system of MARCC.



9. To transfer files from one side to another, just select files and/or directories on one side and click on **Start** button to transfer them to the other side.

If you would like to use Globus to transfer data to/from your local computer, please install the [Globus Connect Personal](#). During the installation, an endpoint is set for your local computer and you can use it in the Collection field to access your local file system.

# Module and Software System

[LMOD System and Module Commands](#)

[Find, List and Load Software](#)

[Module hierarchy](#)



## LMOD System and Module Commands

There are many software applications installed in Rockfish cluster system. In order to find and use them properly and conveniently, LMOD module system was built in the Rockfish system. Users can simply run the command :

```
[cchan139@login02 ~]$ module -h
Usage: module [options] sub-command [args ...]
...
...
...
...
```

to find out how to use the module commands. The following is a list of commonly used module commands.

### Common Module Commands

<code>module list</code>	List currently loaded modules.
<code>module spider &lt;software name&gt;</code>	Find installed software in HPCC system
<code>module load &lt;software name&gt;</code>	Load an available software module.
<code>module unload &lt;software name&gt;</code>	Unload a currently loaded software module.
<code>module purge</code>	Unload all loaded modules
<code>module swap &lt;software A&gt; &lt;software B&gt;</code>	Unload software A and load software B
<code>module avail</code>	Show currently available software.
<code>module whatis &lt;software name&gt;</code>	Show a description of the software
<code>module show &lt;software name&gt;</code>	Show functions and environment settings of a module.
<code>module use &lt;module directory&gt;</code>	Use modules installed in a directory

## ml Commands

ml can be ...	Description	Examples
<b>module</b>	if a module command is specified behind ml	ml spider ..., ml show ..., ...
<b>module list</b>	if nothing is specified behind ml	ml
<b>module load</b>	if a module name is specified behind ml	ml git, ml matlab
<b>module unload</b>	if a module name with "-" sign is specified	ml -git, ml -matlab

Users can find out more details about `ml` commands by running `ml -h` on a login node.

## Find, List and Load Software

### List Loaded Module

Right after you ssh to a Rockfish node, some modules are automatically loaded by default. You can use `module list` command to see the loaded modules:

```
[userid@login02 ~]$ module list

Currently Loaded Modules:
  1) gcc/9.3.0    2) openmpi/3.1.6    3) slurm/19.05.7    4) helpers/0.1    5)
   git/2.28.0    6) standard/2020.10
```

Once users have a module loaded, its application is ready to be used. For example, `gcc` command of `gcc/9.3.0` and `mpirun` command of `openmpi/3.1.6` can be found by `which` command:

```
[userid@login02 ~]$ which gcc
/data/apps/linux-centos8-cascadelake/gcc-9.2.0/gcc-9.3.0-bnvby67rgbqevwsd26
4rgz44xucnkhpmp/bin/gcc
[cchan139@login02 ~]$ which mpirun
/data/apps/linux-centos8-cascadelake/gcc-9.3.0/openmpi-3.1.6-rk3nyoehbq3pke
4zy4hn7unns3ujtutx/bin/mpirun
```

where commands `gcc` and `mpirun` are pointed to the places where the applications are installed.

### Find Software Module

If you did not find your preferred software loaded, you can use `module spider` command to look for it first. For example, to look for Python application, just run `module spider python`:

```
[userid@login02 ~]$ module spider python

-----
python:
-----

Versions:
  python/3.7.9
  python/3.8.6
  python/3.9.0
-----
... ..
```

and a list of installed python versions is displayed. Before you try to load a version of Python, you might want to find out what needs to be loaded first before you can load it. You can use the same

module spider command with the specified version. For example, to load Python version 3.9.0, use the command `module spider python/3.9.0`:

```
[userid@login02 ~]$ module spider python/3.9.0
```

```
-----  
python: python/3.9.0  
-----
```

You will need to load all module(s) on any one of the lines below before the `"python/3.9.0"` module is available to load.

```
gcc/9.3.0
```

Help:

The Python programming language.

and we found the compiler `gcc/9.3.0` needs to be loaded first. Due to module hierarchy used in our module system, before an application can be loaded, the version of its compiler (or maybe its MPI compiler) needs to be loaded first.

## Load Modules

Now, we can run the `module load` command:

```
[userid@login02 ~]$ module load gcc/9.3.0 python/3.9.0
```

to load both `gcc/9.3.0` and `python/3.9.0` at the same time. Again, we can use `module list` and `which` command to check if they are loaded:

```
[userid@login02 ~]$ module list
```

Currently Loaded Modules:

```
1) slurm/19.05.7   3) standard/2020.10   5) openmpi/3.1.6   7)  
python/3.9.0
```

```
2) helpers/0.1    4) gcc/9.3.0          6) git/2.28.0
```

```
[userid@login01 ~]$ which python
```

```
/data/apps/linux-centos8-cascadelake/gcc-9.3.0/python-3.9.0-yjjcnbjcnwubdg  
kk4d6qy64twbio233/bin/python
```

As we can see, not only are they loaded but also the python command is pointed to the application directory.

## Show Module Functions

The functions of a loaded module are to set up the environment in the running session so the application command can be executed directly and properly. Users can use `module show` command to find out the settings. For example, after the `slurm/19.05.7` module is loaded, we can run

```
[userid@login02 ~]$ module show slurm/19.05.7
-----
/data/apps/lmod/linux-centos8-x86_64/Core/slurm/19.05.7:
-----
whatis("Adds Slurm to your environment ")
setenv("CMD_WLM_CLUSTER_NAME","slurm")
setenv("SLURM_CONF","/cm/shared/apps/slurm/var/etc/slurm/slurm.conf")
prepend_path("PATH","/cm/shared/apps/slurm/current/bin")
prepend_path("PATH","/cm/shared/apps/slurm/current/sbin")
prepend_path("MANPATH","/cm/shared/apps/slurm/current/man")
prepend_path("LD_LIBRARY_PATH","/cm/shared/apps/slurm/current/lib64")
prepend_path("LD_LIBRARY_PATH","/cm/shared/apps/slurm/current/lib64/slurm")
prepend_path("LIBRARY_PATH","/cm/shared/apps/slurm/current/lib64")
prepend_path("LIBRARY_PATH","/cm/shared/apps/slurm/current/lib64/slurm")
prepend_path("CPATH","/cm/shared/apps/slurm/current/include")
help([[ Adds Slurm to your environment
]])
```

to see the setup of the environment variables during module loading. We can also use the results to find the installed location of the application: `/cm/shared/apps/slurm/` and the module file name with the path `/data/apps/lmod/linux-centos8-x86_64/Core/slurm/19.05.7`.

## Module hierarchy

In order to use an application with the linked libraries correctly, the same versions of the compiler and the libraries by which the application was built also need to be loaded. User can see this feature with lists of directly available modules through the `module avail` command:

```
[userid@login02 ~]$ module avail

*** RockFish Software ***
Use "module spider <name>" to search all software.
The available software depends on the compiler, MPI,
Python, and R modules you have already loaded.
https://lmod.readthedocs.io/en/latest/010\_user.html

----- gcc (9.3) + openmpi (3.1) -----
boost/1.74.0    hdf5/1.10.7    openfoam-org/7    py-netcdf4/1.5.3
fftw/3.3.8     hypre/2.20.0    openmm/7.4.1     remora/1.8.5
... ..
... ..
gromacs/2020.5      (D)    netcdf-fortran/4.5.3    petsc/3.14.0

----- gcc (9.3) -----
aspera-cli/3.7.7    htlib/1.10.2    python/3.7.9
bedtools2/2.27.1    intel-mkl/2020.3.279-tbb    python/3.8.6      (D)
... ..
... ..
gmp/6.1.2    openmpi/3.1.6      (L)    vmd/1.9.4

----- external software -----
Aspera-Connect/3.9.6    amber/20    gaussview/gv
... ..
... ..
abinit/9.6.2      (D)    gaussian/16    orca/5.0.2      (D)

----- core -----
anaconda/2020.07    helpers/0.1 (L)    intel/2020.2    own/0.1
standard/2020.10 (L)
gcc/9.3.0      (L)    intel/2020.1 (D)    slurm/19.05.7 (L)

Where:
L:  Module is loaded
D:  Default Module
```

As you can see, all of the available modules are separated into 4 parts. Modules under `gcc (9.3) + openmpi (3.1)` are available since they are compiled by the gcc version 9.3 and OpenMPI version 3.1 and modules under `gcc (9.3)` are built by the gcc version 9.3 only. The application modules under `core` and `external software` can be loaded directly since they only need system libraries. If you try to load a module which is not available, you will get an error message. For examples, loading `quantum-espresso` module will display the module can not be loaded

```
[userid@login02 ~]$ module load quantum-espresso
Lmod has detected the following error: These module(s) or extension(s)
exist but cannot be loaded as
requested: "quantum-espresso"
Try: "module spider quantum-espresso" to see how to load the module(s).
```

and suggest you to use the `module spider` command to find out how to load it. This is exactly the way we mentioned in the previous section. The spider command shows the intel compiler `intel/2020.2` and intel-mpi compiler `intel-mpi/2020.2` also need to be loaded.

```
[userid@login02 ~]$ module spider quantum-espresso/
```

```
-----
quantum-espresso: quantum-espresso/6.6
-----
```

You will need to load all module(s) on any one of the lines below before the `"quantum-espresso/6.6"` module is available to load.

```
intel/2020.2  intel-mpi/2020.2
```

```
... ..
```

By including `intel/2020.2` and `intel-mpi/2020.2` in the `module load` command, the `quantum-espresso` module can be loaded.

```
[userid@login02 ~]$ module load intel/2020.2 intel-mpi/2020.2
quantum-espresso
```

Lmod is automatically replacing `"gcc/9.3.0"` with `"intel/2020.2"`.

Lmod is automatically replacing `"openmpi/3.1.6"` with `"intel-mpi/2020.2"`.

Due to MODULEPATH changes, the following have been reloaded:

1) git/2.28.0

```
[cchan139@login02 ~]$ ml
```

Currently Loaded Modules:

1) slurm/19.05.7	3) standard/2020.10	5) intel-mpi/2020.2	7) quantum-espresso/6.6
2) helpers/0.1	4) intel/2020.2	6) fftw/3.3.8	8) git/2.28.0

Due to conflict between different compilers, loading intel and intel-mpi modules replaces gcc and openmpi. As shown in the module list (ml) command, `quantum-espresso` is successfully loaded and both gcc and openmpi are not in the module list.



## Parallel Computing Jobs

The simplest way to do work in parallel is to run many jobs independently at a time. However, to perform many operations coherently and concurrently on multiple CPUs in one or multiple nodes, an application needs to be programmed with a parallel model and compiled by certain libraries. For HPC software, three basic parallel models: **Shared Memory**, **Distributed Memory** and **Hybrid Model** are usually used.

### Shared Memory with Threads

- It performs some serial work, and then creates a number of threads running by CPU (or GPU) cores concurrently.
- Each thread can have local data, but also, shares the entire resources, including RAM memory of the main program.
- Threads communicate with each other through global memory (RAM). This requires synchronization operations to ensure that no more than one thread is updating the same RAM address at any time.
- Threads can come and go, but the main program remains present to provide the necessary shared resources until the application has completed.

Examples: POSIX Threads, OpenMP, CUDA threads for GPUs

An example of OpenMP resource request:

```
#SBATCH --ntasks=1
#SBATCH --cpu-per-task=8                # run 8 threads
```

### Distributed Memory with Tasks

- A main program creates a set of tasks (processes) that use their own local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.
- Tasks exchange data through communications by sending and receiving messages through a fast network (e.g. infinite band).
- Data transfer usually requires cooperative operations to be performed by each process. For example, a **send** operation must have a matching **receive** operation.
- Synchronization operations are also required to prevent race conditions.

Example: Message Passing Interface (**MPI**)

An example of MPI resource request:

```
#SBATCH --ntasks=8
#SBATCH --cpu-per-task=1
```

```
# mpirun -np 8
```

## Hybrid Parallel

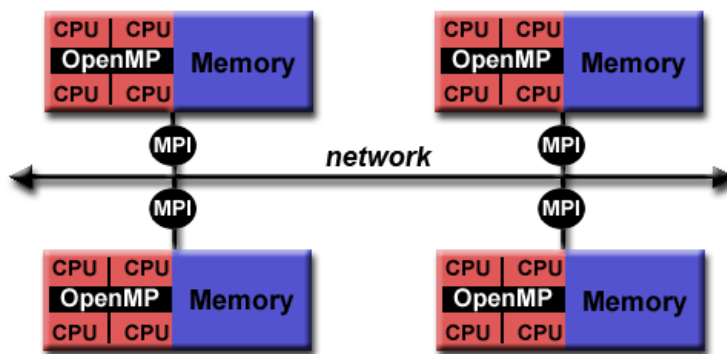
- A hybrid model combines more than one of the previously described programming models.
- A simple example (OpenMPI) is the combination of the message passing model (MPI) with the threads model (OpenMP).
  - Threads perform computationally intensive kernels with the local RAM in a node
  - Communications between processes on different nodes occurs over the network using MPI
- Works well to the most popular HPC clusters with multi/many-core machines.
- Other example: MPI with CPU-GPU (Graphics Processing Unit)

An example of OpenMPI resource request:

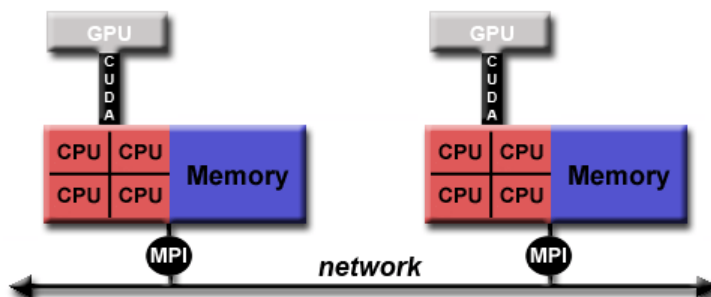
```
#SBATCH --ntasks=6
#SBATCH --cpu-per-task=4
```

```
# mpirun -np 6
# run 4 threads for each task
```

### Hybrid OpenMP-MPI Parallel Model:



### Hybrid CUDA-MPI Parallel Model:



## Job Management by SLURM

In order to have fair usage of the **shared** cluster, SLURM (Simple Linux Utility for Resource Management) is used to manage users' jobs and computing resources in the Rockfish system. SLURM is an open-source, fault-tolerant, and highly scalable scheduling system. It has been employed by a large number of national and international computing centers. Users can use SLURM commands to submit jobs with resource requests, monitor job status and collect resource usages.

[Request Interactive jobs](#)

[List of Job Options](#)

[Batch Job Script](#)

[SLURM Environment Variables](#)

[Submit and Monitor Jobs](#)

## Request Interactive jobs

It is helpful to run your work and get the response of the commands right away to see if any error is in your workflow. If an interactive job is required for job testing, users can use the `interact` command to request one. Simply use the command on a login node to display the usage.

```
[userid@login02 ~]$ interact

usage: interact [-n tasks or cores] [-t walltime] [-r reservation] [-p
partition] [-a Account] [-f featurelist] [-h hostname] [-g ngpus]

Starts an interactive job by wrapping the SLURM 'salloc' and 'srun'
commands.

options:
  -n tasks           (default: 1)
  -m memory           memory in K|M|G|T (if m > max-per-cpu * cpus, more cpus
are requested)
  -t walltime         as hh:mm:ss (default: 30:00)
  -r reservation      reservation name
  -p partition        (default: 'defq')
  -a Account          If users needs to use a different account. Default is
primary PI
  -f featurelist      SLURM features (e.g., 'haswell'),
combined with '&' and '|' (default: none)
  -h hostname         only run on the specific node 'hostname'
                      (default: none, use any available node)
  -g gpus             specify GRES for GPU-based resources
```

As mentioned in the results, this command is related to 'salloc' and 'srun' commands. We can see how it works by requesting an interactive job:

```
[userid@login03 ~]$ interact -n 1
Tasks:      1
Cores/task: 1
Total cores: 1
Walltime: 30:00
Reservation:
Queue:      defq
Command submitted: salloc -J interact -N 1-1 -n 1 --time=30:00 -p defq srun
--pty bash
```

```
salloc: Granted job allocation 3624855
```

```
... ..
```

```
... ..
```

```
[userid@c003 ~]$
```

where the real command executed is

```
salloc -J interact -N 1-1 -n 1 --time=30:00 -p defq srun --pty bash
```

In other words, the `interact` command actually uses the syntax:

```
salloc <Job Options> srun --pty bash
```

to request an interactive job. A list of available job options is mentioned in the next section and we can use them for job submission.

## List of Job Options

The following is a list of basic job specifications. To see the complete options, please refer to the SLURM [sbatch command page](#).

### Computing Resource Options

Job Options	Description	Examples in Job Script
<b>-c,</b> <b>--cpus-per-task=&lt;ncpus&gt;</b>	Require <i>ncpus</i> number of processors per task	#SBATCH -c 3 (3 cores per node)
<b>--gres=&lt;list&gt;</b>	Specifies a comma delimited list of generic consumable resources. The format of each entry on the list is "name[:type]:count", where name is that of the consumable resource. To request for GPUs, --gres=gpu:3 is an example to request 3 A100 GPUs.	#SBATCH --gres=gpu:2 (request 2 GPUs per node)
<b>--mem=&lt;size[units]&gt;</b>	Specify the real memory required per node.	#SBATCH --mem=2G (M or G bytes)
<b>--mem-per-cpu=&lt;size[units]&gt;</b>	Minimum memory required per allocated CPU	#SBATCH --mem-per-cpu=2G (M or G bytes)
<b>-N,</b> <b>--nodes=&lt;minnodes[-maxnodes]&gt;</b>	Request that a minimum of <i>minnodes</i> nodes be allocated to this job. A maximum node count may also be specified with <i>maxnodes</i> . If only one number is specified, this is used as both the minimum and maximum node count.	#SBATCH --nodes=2-4 (Request 2 to 4 different nodes)
<b>-n,</b> <b>--ntasks=&lt;number&gt;</b>	Request total <i>number</i> of tasks. The default is one task per node, but note that the <b>--cpus-per-task</b> option will change this default.	#SBATCH -n 4 (All tasks could be in 1 to 4 different nodes)

<b>--ntasks-per-node=&lt;ntasks&gt;</b> <b>--tasks-per-node=&lt;ntasks&gt;</b>	Request that <i>ntasks</i> be invoked on each node. This is related to <b>--cpus-per-task=<i>ncpus</i></b> , but does not require knowledge of the actual number of cpus on each node.	
<b>-p,</b> <b>--partition=&lt;partition_names&gt;</b>	Request a specific partition for the resource allocation. If not specified, the default behavior is used.	#SBATCH -p a100 #SBATCH -p bigmem
<b>-A,</b> <b>--account=&lt;account&gt;</b>	Charge resources used by this job to specified account.	#SBATCH -A <account>
<b>-t, --time=&lt;time&gt;</b>	Set a limit on the total run time of the job allocation. The total run time in the form: HH:MM:SS or DD-HH:MM:SS	#SBATCH -t 00:20:00
<b>-w, --nodelist=&lt;node name list&gt;</b>	Request a specific list of your buy-in nodes. The job will contain <i>all</i> of these hosts and possibly additional hosts as needed to satisfy resource requirements. The list may be specified as a comma-separated list of hosts, a range of hosts, or a filename. The host list will be assumed to be a filename if it contains a "/" character.	#SBATCH --nodelist=c001,c004,c011,.. .. #SBATCH -w c[011-015,...] #SBATCH -w /home/userid/nodelist
<b>-x, --exclude=&lt;node name list&gt;</b>	Explicitly exclude certain nodes from the resources granted to the job.	
<b>-J,</b> <b>--job-name=&lt;jobname&gt;</b>	Specify a name for the job allocation.	#SBATCH -J MyJob

## Other Job Options

Job Options	Description	Examples in Job Script
<b>--mail-type=&lt;type&gt;</b>	Notify user by email when certain event types occur. Valid <i>type</i> values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit), TIME_LIMIT_50 (reached 50 percent of time limit) and ARRAY_TASKS (send emails for each array task).	#SBATCH --mail-type=BEGIN,END
<b>--mail-user=&lt;user&gt;</b>	User to receive email notification of state changes as defined by <b>--mail-type</b> . The default value is the submitting user.	#SBATCH --mail-user=user@msu.edu
<b>--export=&lt;environment variables [ALL]   NONE&gt;</b>	Identify which environment variables are propagated to the launched application, by default all are propagated. Multiple environment variable names should be comma separated.	#SBATCH --export=EDITOR=/bin/emacs,ALL
<b>--begin=&lt;time&gt;</b>	Submit the batch script to the Slurm controller immediately, like normal, but tell the controller to defer the allocation of the job until the specified time. Time may be of the form <i>HH:MM:SS</i> to run a job at a specific time of day (seconds are optional).	#SBATCH --begin=16:00



## Options Disabled for Interactive Job

Job Options	Description	Examples in Job Script
<b>-e,</b> <b>--error=&lt;filename&gt;</b>	Instruct Slurm to connect the batch script's standard error directly to the file name specified. By default both standard output and standard error are directed to the same file. See <b>-o,</b> <b>--output</b> for the default file name.	<pre>#SBATCH -e /home/username/myerrorfile</pre>
<b>-o,</b> <b>--output=&lt;filename pattern&gt;</b>	<p>Instruct Slurm to connect the batch script's standard output directly to the file name specified in the "<a href="#">filename pattern</a>".</p> <p>The default file name is "slurm-%j.out", where the "%j" is replaced by the job ID. For job arrays, the default file name is "slurm-%A_%a.out", "%A" is replaced by the job ID and "%a" with the array index.</p>	<pre>#SBATCH -o /home/username/output-file</pre> <p>Need a <i>file name</i> or <i>filename pattern</i> not just a directory.</p>
<b>-a,</b> <b>--array=&lt;indexes&gt;</b>	<p>Submit a job array, multiple jobs to be executed. The <i>indexes</i> specification identifies what array ID values should be used. Each job has the same job ID (\$SLURM_JOB_ID) but different array ID (\$SLURM_ARRAY_TASK_ID variable). Can use step function with ":" separator. A maximum number of simultaneously running jobs may be specified with "%" separator.</p>	<pre>#SBATCH -a 0-15 #SBATCH --array=0,6,16-32 #SBATCH --a 0-15:4 (same as #SBATCH -a 0,4,8,12) #SBATCH --array=0-15%4 (4 jobs running simultaneously)</pre>

More options can be found on the [sbatch option](#) website.

## Batch Job Script

To submit a batch job and run it on a compute node, users need to use `sbatch` command with a job script file. The job script is supposed to contain three parts. The first part is the first line of the file which specifies the shell to run the script. By default, bash shell is generally used to run our command lines. The first line should be like

```
#!/bin/bash
```

The second part contains the lines of resource requests and job options. Each of the lines must start with the words `#SBATCH` so the job scheduler (SLURM) can read and manage the resources. For example, the following `SBATCH` lines:

<code>#SBATCH --job-name=MyTest</code>	<code># Job name (-J MyTest)</code>
<code>#SBATCH --time=4:00:00</code>	<code># Time limit (-t 4:00:00)</code>
<code>#SBATCH --nodes=1</code>	<code># Number of nodes (-N 1)</code>
<code>#SBATCH --ntasks=2</code>	<code># Number of processors (-n 2)</code>
<code>#SBATCH --cpus-per-task=6</code>	<code># Threads per process (-c 6)</code>
<code>#SBATCH --partition=defq</code>	<code># Used partition (-p defq)</code>
<code>#SBATCH --mem-per-cpu=4GB</code>	<code># Define memory per core</code>

specify the job name to be `MyTest`. It can use 2 processes simultaneously (in parallel) on one node in the `defq` partition with 6 threads in each process. The maximum memory usage is 4GB per CPU and maximum running time is 4 hours.

The third part is the command lines which will be run on the compute nodes when the job starts. The command lines should include all commands of job workflow after logging into a node, such as module loading, environment setting and running application commands. An example of the command lines are

```
module load intel/2020.2 intel-mpi/2020.2
module load quantum-espresso/6.6

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
mpirun -n $SLURM_NTASKS pw.x < scf.in > scf.out

scontrol show job $SLURM_JOBID
```

The first 2 command lines load the necessary modules to run the QuantumESPRESSO software. The `export` command sets the environment variable `OMP_NUM_THREADS` as the SLURM environment variable `$SLURM_CPUS_PER_TASK` which is the requested number of CPUs per task. The setting allows the application to run with multiple threads. The `mpirun` command starts to run the `pw.x` command in parallel with the number of the processes the same as the SLURM variable `$SLURM_NTASKS` set to be the requested number of tasks. The last command will print the job

information to the SLURM output file, where the environment variable `$SLURM_JOBID` is set to be the job ID of the job. More SLURM variables can be seen in the [SLURM Environment Variables](#) section.

By default, the job standard output and standard error will be sent to the SLURM output file `slurm-<JobID>.out` in the directory where you run the job submission command. Users can use the `-o` or `-e` option to specify a different output or a different error file name with a preferred location. If the `-e` option is not specified, both messages are sent to the output file. Users can also use the [filename pattern](#) to name the file. For example, using the specifications:

```
#SBATCH -o /home/userid/%j/%x.out
#SBATCH -e /home/userid/%j/%x.err
```

will send the output to the file `/home/userid/<JobID>/<JobName>.out` and the error to the file `/home/userid/<JobID>/<JobName>.err`, where `<JobID>` and `<JobName>` are the ID and name of the job respectively. If there exists a file with the same file name as the output file name, the job output will append to it.

## Other Job Options and Usages

As mentioned in the [Hardware Resources](#) section, there are 3 partitions: `defq`, `bigmem` and `a100` for Rockfish compute nodes. Users can use the `sinfo` command to see them:

```
[cchan139@login02 ~]$ sinfo -s
PARTITION AVAIL  TIMELIMIT  NODES(A/I/O/T)  NODELIST
a100      up 3-00:00:00      8/0/2/10  gpu[01-10]
bigmem    up 2-00:00:00      3/7/3/13  bigmem[01-13]
defq*     up 3-00:00:00    167/307/22/496
c[001-232,241-280,289-328,337-376,385-424,433-440,625-720]
```

By default, submitted jobs will use the `defq` partition if no partition is specified. If users would like to use large memory nodes or A100 GPUs, they need to get the allocation first and use the partition with the related account. For example, to run jobs on `bigmem` nodes, put the job options in the job script:

```
#SBATCH --partition=bigmem
#SBATCH -A PI-userid_bigmem
```

Please also make sure the memory request is more than 192GB. To run jobs with A100 GPUs, users can use the options

```
#SBATCH --gres=gpu:4
#SBATCH -p a100
#SBATCH -A PI-userid_gpu
```

where the first line is to request the number of GPUs and the last 2 lines specify the partition and the account options respectively.

More job script examples can be found on the [Running Jobs](#) website.

## Job Array

If users would like to submit multiple jobs with similar job scripts, use of the job array option could be a good idea. Submission of a job array will use the same job script for all jobs but with a different SLURM environment variable `$SLURM_ARRAY_TASK_ID` for each different job. Users can apply the variable on the command lines to create different job runnings. A simple job array example could be as

```
[userid@login02 ~]$ cat ArrayJob.sb
#!/bin/bash

#SBATCH --array=1-5

echo "SLURM job $SLURM_ARRAY_TASK_ID starts on $HOSTNAME"

[userid@login02 ~]$ sbatch ArrayJob.sb
Submitted batch job 516540
```

Submitting this script `ArrayJob.sb` returns a parent job ID `516540` as `$SLURM_ARRAY_JOB_ID` and generates 5 jobs with job IDs from `516540_1` to `516540_5`. Each job has the output file as `slurm-516540_${SLURM_ARRAY_TASK_ID}.out` and has different output as

```
[userid@login02 ~]$ head slurm-516540_*.out
==> slurm-516540_1.out <==
SLURM job 1 starts on c007
==> slurm-516540_2.out <==
SLURM job 2 starts on c007
==> slurm-516540_3.out <==
SLURM job 3 starts on c007
==> slurm-516540_4.out <==
SLURM job 4 starts on c007
==> slurm-516540_5.out <==
SLURM job 5 starts on c007
```

Users can also change the output or error file name with the `-o` or `-e` options in the job script. The following is the list of the SLURM variables related to the [file patterns](#):

SLURM Variables	File Pattern	Description
SLURM_ARRAY_JOB_ID	%A	Job array's parent job allocation number
SLURM_ARRAY_TASK_ID	%a	Job array ID (index) number
SLURM_JOB_ID	%j	Job identifier ID
SLURM_JOB_NAME	%x	Name of the job

## Estimating Computing Resources

When users try to set the job options on computing resources, they might have problems estimating how much to request. If users request too much, the job could wait in the queue for a long time to get the available resources. However, if users request too less, the job could over-utilize them and get canceled before it finishes. If you have no experience in job resource usage, it is suggested to request more to make sure it can finish running for the first time. Check the resource usages of the job with the `seff` command (see [Submit and Monitor Jobs](#) section) and adjust the computing resources for the next jobs.

## SLURM Environment Variables

The following is a list of SLURM environment variables available for the job script or during an interactive job session. Users are able to get the variable settings by using the command `env | grep SLURM` in an interactive job session or in a batch job script.

SLURM Variables	File Pattern	Description
<b>SLURM_CPUS_PER_TASK</b>		Number of cpus requested per task. Only set if the <b>--cpus-per-task</b> option is specified.
<b>SLURM_JOB_ACCOUNT</b>		Account name associated of the job allocation
<b>SLURM_JOBID</b> <b>SLURM_JOB_ID</b>	%j	The ID of the job allocation
<b>SLURM_JOB_NAME</b>	%x	Name of the job
<b>SLURM_NODELIST</b> <b>SLURM_JOB_NODELIST</b>		List of nodes allocated to the job
<b>SLURM_NNODES</b> <b>SLURM_JOB_NUM_NODES</b>		Total number of different nodes in the job's resource allocation
<b>SLURM_MEM_PER_NODE</b>		Same as <b>--mem</b>
<b>SLURM_MEM_PER_CPU</b>		Same as <b>--mem-per-cpu</b>
<b>SLURM_NTASKS</b> <b>SLURM_NPROCS</b>		Same as <b>-n, --ntasks</b>
<b>SLURM_NTASKS_PER_NODE</b>		Number of tasks requested per node. Only set if the <b>--ntasks-per-node</b> option is specified.
<b>SLURM_SUBMIT_DIR</b>		The directory from which <b>sbatch</b> was invoked
<b>SLURM_ARRAY_TASK_ID</b>	%a	Job array ID (index) number
<b>SLURM_ARRAY_JOB_ID</b>	%A	Job array's master job ID number

More variables are displayed on the [SLURM environment variables](#) website.

## Submit and Monitor Jobs

### Job Submission

After you have a job script, you can use the `sbatch` command with the file name of the job script to submit the job. For example, the command

```
[userid@login02 MyJob]$ sbatch jobfile
Submitted batch job 3679746
```

submits the script file `jobfile`. If it is submitted successfully, you will see the output with a job ID. Please notice some facts about job submission:

- In case the job script does not specify any job options, users can also specify job options mentioned in the [List of Job Options](#) section with `sbatch` command.
- If there is any option conflict between the specifications in the job script and on the command line, the job scheduler will take the options specified with the `sbatch` command.
- By default, if the job scheduler is not able to find any job option, the submitted job will use 1 node, 1 task and 1 hour for the job running.
- After a job starts, it will run the command lines and create a SLURM output file in the directory where you execute `sbatch` command. Please make sure you use the right directory to run `sbatch` command.

### Job Listing and Information

To list all your jobs in the SLURM queue, you can run `squeue` command:

```
[userid@login02 MyJob]$ squeue
      USER      ACCOUNT      JOBID PARTITION      NAME  NODES   CPUS
TIME_LIMIT      TIME NODELIST ST REASON
      userid    mygroup      3679746 defq      node_work      1      1
1:00:00      0:00      PD None
```

The `squeue` command is the same as the SLURM job listing command `squeue` :

```
squeue -u $USER -o '%.12u %.9a %.12i %9P %.10j %.5D %.5C %.10l %.8M %.8N %t
%r '
```

Users can also learn about how to use the command on the [squeue](#) website.

More information about a specific job can also be displayed by the `scontrol show job` command:

```
[userid@login02 MyJob]$ scontrol show job 3679746
```

```

JobId=3679746 JobName=node_work
  UserId=userid(1495) GroupId=mygroup(1002) MCS_label=N/A
  Priority=2630 Nice=0 Account=mygroup QOS=normal
  JobState=COMPLETED Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:00:01 TimeLimit=01:00:00 TimeMin=N/A
  SubmitTime=2022-03-15T09:36:31 EligibleTime=2022-03-15T09:36:31
  AccrueTime=2022-03-15T09:36:31
  StartTime=2022-03-15T09:36:31 EndTime=2022-03-15T09:36:32 Deadline=N/A
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=2022-03-15T09:36:31
  Partition=defq AllocNode:Sid=login02:2912087
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=c005
  BatchHost=c005
  NumNodes=1 NumCPUs=1 NumTasks=0 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=1,node=1,billing=1
  Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
  MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
  Features=(null) DelayBoot=00:00:00
  OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=./jobfile
  WorkDir=/home/userid/MyJob
  StdErr=/home/userid/MyJob/slurm-3679746.out
  StdIn=/dev/null
  StdOut=/home/userid/MyJob/slurm-3679746.out
  Power=

```

At any time, you would like to cancel jobs in the queue, use `scancel` command with job IDs behind it. For example, to cancel the job 3679746, run `scancel 3685853`

```

[userid@login02 MyJob]$ scancel 3685853
[userid@login02 MyJob]$ $ sqme
      USER   ACCOUNT      JOBID PARTITION      NAME  NODES   CPUS
TIME_LIMIT   TIME NODELIST ST  REASON

```

and it is canceled.

## Job States and Pending Reasons

After a job is submitted, it could be in many different states (ST). The basic job states are these:

- Pending (PD): job is in the queue, waiting to be scheduled.



- Running (R): job has been granted an allocation and is running.
- Completing (CG): job is in the process of completing.
- Held (H): job was submitted, but was put in the held state (held by users or system).
- Timeout (TO): job was terminated for running longer than its wall clock limit.
- Failed (F): job terminated with a non-zero status.
- Node Fail (NF): job terminated after a compute node reported a problem.

More job states can be found on the [SLURM JOB STATE](#) web site.

If a job is in the pending state, it could be due to many reasons. Some basic reasons are these:

- Resources - The job is waiting for resources to become available.
- Priority - The job needs to wait and get higher priority for this partition.
- ReqNodeNotAvail - Nodes specifically required by the job are not currently available.
- AssociationResourceLimit - The job's association has reached some resource limit.
- QOSResourceLimit - The job's QOS has reached some resource limit.
- PartitionDown - The queue is currently closed to running any new jobs.

Please see more reasons on the [SLURM JOB REASON](#) web site.

## Resource Usages

After a job finishes, users might need to check the resource usages for the next job submission. Since it is no longer in the job queue, `sqme` or `scontrol show job` command can not list or display the job information. Users will need to use the `sacct` command to see a list of finished jobs.

To show the jobs submitted or in running today, you can run `sacct -X` command:

```
[userid@login02 MyJob]$ sacct -X
      JobID      JobName  Partition      Account  AllocCPUS      State
ExitCode
-----
-----
3679742          FastRun      defq      mygroup          1  COMPLETED
0:0
3679746      node_work      defq      mygroup          1  COMPLETED
0:0
```

to get the job list. The `-X` option will prevent the display of each detailed job step. If users would like to see the jobs over a period of time in the past, the `-S` option for the start time and `-E` option

for the end time can be used with the command. For example, to get the jobs running during the days between March 1st, 2022 and March 5th, 2022, the command displays the results:

```
[userid@login02 MyJob]$ sacct -X -S 2022-03-01 -E 2022-03-05
      JobID      JobName Partition      Account AllocCPUS      State
ExitCode
-----
3342147      interact      defq      mygroup          1      FAILED
1:0
3342150      interact      defq      mygroup          1      FAILED
2:0
3345156          test2      defq      mygroup         16      COMPLETED
0:0
3345497      interact      defq      mygroup          1      COMPLETED
0:0
```

where the date format after -S and -E option is year-month-day. If -E is not specified, the end of time is the current time. More usage about the command can be found on the [sacct](#) website.

After you find the job of which you would like to know the resource usages, you can run the seff command with the job ID behind it to get the results. For example, to get the resource usage of the job 3345156, you can run

```
[userid@login02 MyJob]$ seff 3345156
Job ID: 3345156
Cluster: slurm
User/Group: userid/mygroup
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 16
CPU Utilized: 00:17:45
CPU Efficiency: 96.81% of 00:18:20 core-walltime
Job Wall-clock time: 00:01:09
Memory Utilized: 28.72 GB
Memory Efficiency: 89.75% of 32.00 GB
```

to find the CPU, memory and wall-time usages with the efficiencies. The core-walltime with orange background is the CPU time which will be charged to your group account.

## Container and Singularity

Some applications are not able to run on the Rockfish cluster due to system conflict. Containers allow developers to package up an application with all of the dependencies so we can use it as an application package in our system. Docker is a popular tool to create, deploy and run application containers. However, due to its root privilege requirement, most HPC systems do not allow users to install and use it. In order to use Docker containers, we have Singularity installed in compute nodes. Singularity is compatible with Docker and able to run its containers.

To use Singularity, users can request an interactive job. After the job starts, `singularity` command can be used directly:

```
[userid@c011 ~]$ which singularity
/usr/bin/singularity
```

To pull a docker container image, users can use the [Docker Hub](#) web site to look for it. Once the preferred container is found, use the docker pull address and `singularity pull` command to download the image file in the current directory. For example, to pull the Docker [python container](#) of image file `3.9.6-slim-buster` (docker pull `python:3.9.6-slim-buster`), we can run

```
[userid@c011 Singularity]$ singularity pull python-3.9.6.sif
docker://python:3.9.6-slim-buster

INFO:      Converting OCI blobs to SIF format
INFO:      Starting build...
... ..
... ..
Getting image source signatures
INFO:      Creating SIF file...
```

and the image file is saved with the name `python-3.9.6.sif`:

```
[userid@c011 Singularity]$ ls
python-3.9.6.sif
```

To run the image, just use `singularity run` command with the image file name:

```
userid@c011 Singularity]$ singularity run python-3.9.6.sif
Python 3.9.6 (default, Aug 17 2021, 02:38:04)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

where, python version 3.9.6 is ready to use inside the container. We can check the container shell and environment with the `singularity shell` command:

```
[userid@c011 Singularity]$ singularity shell python-3.9.6.sif
Singularity> echo $PATH
/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
Singularity> ls /usr/local/bin
2to3      idle  idle3.9  pip3     pydoc    pydoc3.9  python-config
python3-config  python3.9-config
2to3-3.9  idle3  pip      pip3.9  pydoc3   python    python3
python3.9      wheel
```

If users would like to run other commands installed in the container image, such as `2to3`, the `singularity exe` command can be used:

```
[userid@c011 Singularity]$ singularity exec python-3.9.6.sif 2to3 --help
Usage: 2to3 [options] file|dir ...
```

Options:

-h, --help	show this help message and exit
-d, --doctests_only	Fix up doctests only
-f FIX, --fix=FIX	Each FIX specifies a transformation; default: all
-j PROCESSES, --processes=PROCESSES	Run 2to3 concurrently
-x NOFIX, --nofix=NOFIX	Prevent a transformation from being run
-l, --list-fixes	List available transformations
-p, --print-function	Modify the grammar so that <code>print()</code> is a function
-e, --exec-function	Modify the grammar so that <code>exec()</code> is a function
-v, --verbose	More verbose logging
--no-diffs	Don't show diffs of the refactoring
-w, --write	Write back modified files
-n, --nobackups	Don't write backups for modified files
-o OUTPUT_DIR, --output-dir=OUTPUT_DIR	Put output files in this directory instead of overwriting the input files. Requires -n.
-W, --write-unchanged-files	Also write files even if no changes were required (useful with --output-dir); implies -w.
--add-suffix=ADD_SUFFIX	Append this string to all output filenames.

To find out more usages of singularity commands, simply run `singularity --help` command:

```
[cchan139@c011 ~]$ singularity --help
```

Linux container platform optimized for High Performance Computing (HPC) and Enterprise Performance Computing (EPC)

Usage:

```
singularity [global options...]
```

Description:

Singularity containers provide an application virtualization layer enabling

mobility of compute via both application and environment portability.

With

Singularity one is capable of building a root file system that runs on any

other Linux system where Singularity is installed.

Options:

-c, --config string	specify a configuration file (for root or unprivileged installation only) (default "/etc/singularity/singularity.conf")
-d, --debug	print debugging information (highest verbosity)
-h, --help	help for singularity
--nocolor	print without color output (default False)
-q, --quiet	suppress normal output
-s, --silent	only print errors
-v, --verbose	print additional information
--version	version for singularity

Available Commands:

build	Build a Singularity image
cache	Manage the local cache
capability	Manage Linux capabilities for users and groups
config	Manage various singularity configuration (root user only)
delete	Deletes requested image from the library
exec	Run a command within a container
help	Help about any command
inspect	Show metadata for an image
instance	Manage containers running as services
key	Manage OpenPGP keys
oci	Manage OCI containers
plugin	Manage Singularity plugins

pull	Pull an image from a URI
push	Upload image to the provided URI
remote	Manage singularity remote endpoints, keyservers and OCI/Docker registry credentials
run	Run the user-defined default <code>command</code> within a container
run-help	Show the user-defined <code>help</code> for an image
search	Search a Container Library for images
shell	Run a shell within a container
sif	siftool is a program for Singularity Image Format (SIF) file manipulation
sign	Attach digital signature(s) to an image
test	Run the user-defined tests within a container
verify	Verify cryptographic signatures attached to an image
version	Show the version for Singularity

Examples:

```
$ singularity help <command> [<subcommand>]
$ singularity help build
$ singularity help instance start
```

For additional `help` or support, please visit <https://www.sylabs.io/docs/>

## Python and Anaconda

There are several python versions installed in the Rockfish cluster. Users can use `module spider` command to check the installed versions:

```
[userid@login03 ~]$ ml spider python
```

```
-----  
python:  
-----
```

```
  Versions:
```

```
    python/3.7.9
```

```
    python/3.8.6
```

```
    python/3.9.0
```

Once you have a version of python loaded, you can use `module avail` command to check if any python packages are available to load:

```
[userid@login03 ~]$ module load python/3.8.6
```

```
[userid@login03 ~]$ module avail
```

```
*** RockFish Software ***
```

```
Use "module spider <name>" to search all software.
```

```
The available software depends on the compiler, MPI,
```

```
Python, and R modules you have already loaded.
```

```
https://lmod.readthedocs.io/en/latest/010\_user.html
```

```
----- python (3.8) -----  
py-cython/0.29.21   py-pip/20.2         py-scipy/1.5.3  
py-joblib/0.14.0   py-pybind11/2.5.0   py-setuptools/50.1.0  
py-numpy/1.18.5    py-scikit-learn/0.23.2  py-threadpoolctl/2.0.0
```

If more packages are needed, you can also use the `pip` command to install them. Since users are not able to install packages in the python installed directory, You could use the `pip` command with `--user` option to install in the hidden directory `~/.local` of your home space. However, as more python packages are installed, It is difficult to meet all of their requirements by one global installation. Users are strongly suggested to create virtual environments and install python packages in a self-contained directory. In this way, different applications can use different virtual environments to avoid their conflict.

## Virtual Environment

To create a virtual environment, users can first check the version of the used python and create a directory.

```
[userid@login03 ~]$ module list python

Currently Loaded Modules Matching: python
  1) python/3.8.6
[userid@login03 ~]$ mkdir python3.8
[userid@login03 ~]$ cd python3.8/
[userid@login03 python3.8]$
```

Once enter the directory, users can run the `python3 -m venv math-packages` command to create a virtual environment. For example, to create a virtual environment with a name `math-packages`, we can use the command:

```
[userid@login03 python3.8]$ python3 -m venv math-packages
[userid@login03 python3.8]$ ls math-packages/
bin  include  lib  lib64  pyvenv.cfg
```

and the environment is created under the `math-packages` directory. Now we can use the environment by sourcing the script file `activate` under the `bin` directory:

```
[userid@login03 python3.8]$ source math-packages/bin/activate
(math-packages) [userid@login03 python3.8]$
```

Once it is activated, the name of the environment (`math-packages`) is displayed in front of the prompt.

Now, we can install python packages under the environment by the `pip install` command.

```
(math-packages) [userid@login03 python3.8]$ pip install numpy
Collecting numpy
  Downloading
    numpy-1.22.3-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.8 MB)
    |████████████████████████████████████████| 16.8 MB 85 kB/s
Installing collected packages: numpy
Successfully installed numpy-1.22.3
```

We can also install multiple python packages with your specific versions.



```
(math-packages) [userid@login03 python3.8]$ pip install mpmath==1.1.0
sympy==1.4
Collecting mpmath==1.1.0
  Downloading mpmath-1.1.0.tar.gz (512 kB)
    |████████████████████████████████████████| 512 kB 18.2 MB/s
Collecting sympy==1.4
  Downloading sympy-1.4-py2.py3-none-any.whl (5.3 MB)
    |████████████████████████████████████████| 5.3 MB 103.3 MB/s
Using legacy 'setup.py install' for mpmath, since package 'wheel' is not
installed.
Installing collected packages: mpmath, sympy
  Running setup.py install for mpmath ... done
Successfully installed mpmath-1.1.0 sympy-1.4
```

To see all installed python packages in the environment, use the `pip freeze` command:

```
(math-packages) [cchan139@login03 python3.8]$ pip freeze
mpmath==1.1.0
numpy==1.22.3
sympy==1.4
```

At any time you would like to exit the environment, simply run `deactivate` command:

```
(math-packages) [cchan139@login03 python3.8]$ deactivate
[cchan139@login03 python3.8]$
```

and the environment name in front of the prompt disappears.

## Anaconda

The Rockfish cluster also has versions of anaconda installed. After you load a version of anaconda, you can use conda command to create conda environments and install python packages.

```
[userid@login03 conda]$ module load anaconda
[userid@login03 conda]$ conda -V
conda 4.8.3
```

Users are also suggested to use conda environments for installing and running python packages as mentioned in the python section. To create a conda environment, the conda create command is used. For example, to create a `my_conda` environment, execute the command with `-p` option:

```
[userid@login03 conda]$ conda create -p my_env
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.8.3
  latest version: 4.11.0
```

Please update conda by running

```
$ conda update -n base -c defaults conda
```

```
## Package Plan ##
```

```
environment location: /home/userid/conda/my_env
```

```
Proceed ([y]/n)? y
```

After you agree to proceed, the `my_env` environment is created in `my_env` directory under the current directory.

```
[cchan139@login03 conda]$ ls
my_env
```

Users can now activate the environment by the `conda activate` command with the directory path:

```
[userid@login03 conda]$ conda activate ./my_env
(/home/userid/conda/my_env) [userid@login03 conda]$
```

As the same as the python virtual environment, the environment name is displayed in front of the prompt. Now, users can search for available conda packages from the [Anaconda web site](#) and use the `conda install` command to install your preferred conda packages under the environment. For example, we can install `pip` by the command:

```
(/home/userid/conda/my_env) [userid@login03 conda]$ conda install pip
Collecting package metadata (current_repodata.json): done
Solving environment: done
... ..
... ..

(/home/userid/conda/my_env) [userid@login03 conda]$ which pip
~/conda/my_env/bin/pip
```

and the `pip` command becomes available. With the `pip` package installed, all python packages from [PyPI](#) can be installed just as mentioned in the section of python virtual environment.

At any time users would like to quit the environment, simply run `conda deactivate` command

```
(/home/userid/conda/my_env) [userid@login03 conda]$ conda deactivate
[userid@login03 conda]$
```

and the environment name in front of the prompt disappears.

## Install a List of Packages

Some python applications require many python packages as dependencies. We can use a file with a list of them inside and do the installations once. To install multiple packages by the pip command, we can make a text file with the package names and their versions. For example, the installation of a alphafold version requires several python packages and the file [requirements.txt](#) displays a list of them

```
(math-packages) [userid@login03 python3.8]$ cat requirements.txt
abs1-py==0.13.0
biopython==1.79
dm-haiku==0.0.4
dm-tree==0.1.6
docker==5.0.0
immutabledict==2.0.0
jax==0.2.14
ml-collections==0.1.0
numpy==1.19.5
pandas==1.3.4
scipy==1.7.0
tensorflow-cpu==2.5.0
```

A simple pip command with `-r` option installs them all in the environment:

```
(math-packages) [userid@login03 python3.8]$ pip install -r requirements.txt
```

For Anaconda, users can also use a yaml file to create a conda environment and install dependencies all for once. For example, we can create a yaml file as `reqs.yaml` :

```
[userid@login03 conda]$ cat reqs.yaml
name: conda_list
channels:
- conda-forge
dependencies:
- python=3.7
- matplotlib
- scipy
- numpy
```

Use the conda create command:

```
[userid@login03 conda]$ conda env create -f reqs.yaml
```

The listed dependencies can all be installed under the created `conda_list` environment.

## RStudio

To use RStudio installed in the Rockfish cluster, users can simply run the script `r-studio-server.sh` on a login node:

```
[userid@login02 ~]$ r-studio-server.sh
```

```
Creating slurm script: R-Studio-Server.slurm.script
```

```
The Advanced Research Computing at Hopkins (ARCH)
SLURM job script for run RStudio into Singularity container
Support:  help@rockfish.jhu.edu
```

```
Nodes:          1
Cores/task:     1
Total cores:    1
Walltime:       00-02:00
Queue:          defq
```

The R-Studio-Server is ready to run.

1 - Usage:

```
$ sbatch R-Studio-Server.slurm.script
```

2 - How to login see login file (after step 1):

```
$ cat rstudio-server.job.<SLURM_JOB_ID>.out
```

3 - More information about the job (after step 1):

```
$ scontrol show jobid <SLURM_JOB_ID>
```

A job script with the filename `R-Studio-Server.slurm.script` is created in your current directory. To request a job and start a RStudio session interactively, simply follow the output instruction and submit the job script:

```
[userid@login02 ~]$ sbatch R-Studio-Server.slurm.script
Submitted batch job 3549484
```

and an interactive job with the name `rstudio_container_userid` is in the queue, which can be listed with `sqme` command:

```
[userid@login02 ~]$ sqme
      USER   ACCOUNT      JOBID PARTITION      NAME  NODES   CPUS
TIME_LIMIT   TIME NODELIST ST REASON
      userid   mygroup    3549484 defq    rstudio_co    1     1
2:00:00      0:21      c278 R None
```

Now, follow the output instruction and see the output file `rstudio-server.job.<JOBID>.out` for the information of login to R-Studio server:

```
[userid@login02 ~]$ cat rstudio-server.job.3549484.out
```

1. SSH tunnel from your workstation using the following **command**:

```
ssh -N -L 53609:c278:53609 cchan139@login.rockfish.jhu.edu
```

2. **log in** to RStudio Server **in** your web browser using the Rockfish cluster credentials (username and password) at:

```
http://localhost:53609
```

```
user: userid
```

```
password: < ARCH password >
```

3. When **done** using RStudio Server, terminate the job by:

- a. Exit the RStudio Session ("**power**" button **in** the top right corner of the RStudio window)

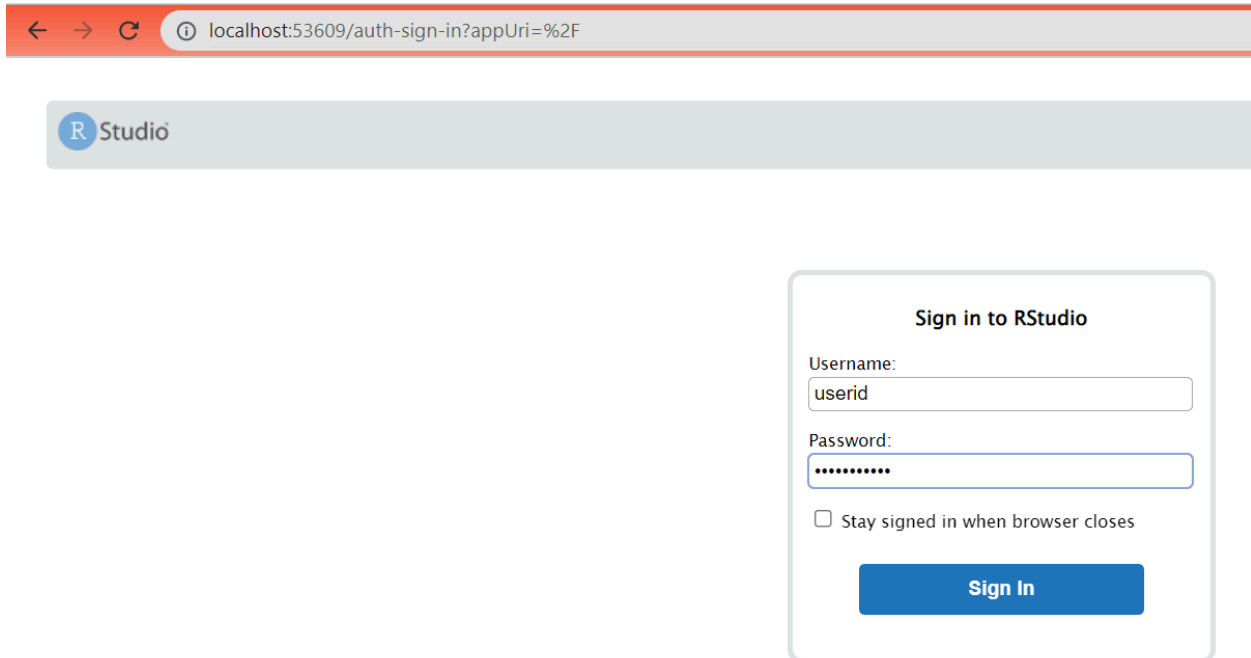
- b. Issue the following **command** on the login node:


```
scancel -f 3549484
```

The first step is to open a new terminal with your ssh client and run the command `ssh -N -L 53609:c278:53609 userid@login.rockfish.jhu.edu` with your password.

```
> ssh -N -L 53609:c278:53609 userid@login.rockfish.jhu.edu
Password:
```

Once it is logged in, follow the second step. Use your web browser to open the address `http://localhost:port-number` and input your username and password to sign in. Once you successfully sign in, the RStudio session is started in the web browser.



If you would like to end the session, follow step 3. Click on the power button  on the top right corner and cancel your job.

```
[userid@login02 ~]$ scancel -f 3549484
```

To find out how to request more resources for RStudio session, use the script command `r-studio-server.sh` with `-h` option on a login node:

```
[userid@login02 ~]$ r-studio-server.sh -h
User Menu
```

```
usage: r-studio-server.sh [options]
                        [-n nodes] [-c cpus] [-m memory] [-t walltime] [-p
partition] [-a account] [-q qos] [-g gpu] [-e email]
```

Starts a SLURM job script to run R-Studio server into singularity container.

options:

```

?, -h help      give this help list
-n nodes        how many nodes you need (default: 1)
-c cpus         number of cpus per task (default: 1)
-m memory       memory in K|M|G|T          (default: 4G)
                (if m > max-per-cpu * cpus, more cpus are requested)
                note: that if you ask for more than one CPU has, your
account gets    charged for the other (idle) CPUs as well
-t walltime     as dd-hh:mm (default: 00-02:00) 2 hours
-p partition    (default: defq)
-a account      if users needs to use a different account. Default is
primary PI      combined with '_' for instance: 'PI-userid'_bigmem
                (default: none)
-q qos          quality of Service's that jobs are able to run in your
association     (default: qos_gpu)
-g gpu          specify GRES for GPU-based resources (eg: -g 1 )
-e email        notify if finish or fail (default: <userid>@jhu.edu)

```



## Jupyter Notebook

To use Jupyter Lab installed in the Rockfish cluster, users can submit a slurm job script to start a job and run Jupyter Lab interactively. If users do not have the script, they can simply run the script `jupyterlab.sh` on a login node to get one.

```
[userid@login02 ~]$ jupyterlab.sh
```

The `jupyterlab.sh` script will create a slurm script `for` multiple environments with `jupyterlab` and `#SBATCH` with default parameters.

Use `jupyterlab.sh --help` `for` more details.

- 1) Slurm script to run jupyterlab (`jupyter_lab.slurm.script`)
- 2) File with login information (`Jupyter_lab.job.<JOBID>.login`)
- 3) File related to slurm INPUT ENVIRONMENT VARIABLES and HTTPS server information (`Jupyter_lab.info`)
- 4) Notebook server file (`.jupyter/jupyter_notebook_config.py`)
- 5) The jupyter-lab, ipykernal, pip will be installed/updated `in`:  
`/home/cchan139/jp_lab`

`<Ctrl+C>` to cancel

Sign `in` with your Rockfish Login credentials:

```
Enter the userid password:
Attempt 1 of 3
?
```

After you enter the password, the job script `jupyter_lab.slurm.script` is saved in the current directoy.

Submit the job script with `sbatch` command and wait for the job to start:

```
[userid@login03 ~]$ sbatch ./jupyter_lab.slurm.script
Submitted batch job 3550093
[userid@login03 ~]$ sqme
```

USER	ACCOUNT	JOBID	PARTITION	NAME	NODES	CPUS
userid	mygroup	3550093	defq	Jupyter_la	1	1
2:00:00	0:53	c005	R None			

Once it starts, check the output file `Jupyter_lab.job.<JOBID>.login` in the directory and use `cat` command to see the 3 steps for how to connect to the Jupyter Lab session.

```
[userid@login03 ~]$ cat Jupyter_lab.job.3550093.login
```

1. SSH tunnel from your workstation using the following `command`:

```
ssh -N -L 49607:c005:49607 cchan139@login.rockfish.jhu.edu
```

2. `log in` to Jupyter Lab `in` your web browser using the Rockfish cluster credentials (username and password) at:

```
http://localhost:49607
```

```
user: cchan139
```

```
password: < ARCH password >
```

3. When `done` using Jupyter Lab, terminate the job by:

- a. Exit the Jupyter Lab ("`file`" button `in` the top left corner of the Jupyter Lab and the shut down)

- b. Issue the following `command` on the login node:

```
scancel -f 3550093
```

The steps are the same as the steps for connecting to RStudio interactive session mentioned in the previous section. Follow the steps and the session is started with your web browser.

To find out more resource requests for Jupyter Lab job script, simply run the script `jupyterlab.sh` with `-h` option and the usage is displayed:

```
[userid@login03 ~]$ jupyterlab.sh -h
```

```
usage: jupyterlab.sh [options]
               [-n nodes] [-c cpus] [-m memory] [-t walltime] [-p
partition] [-a account] [-q qos] [-g gpu] [-e email]
```

Starts a SLURM job script to run Jupyter Lab.

options:

```
?, -h help      give this help list
  -n nodes      how many nodes you need (default: 1)
```

```

-c cpus      number of cpus per task (default: 1)
-m memory    memory in K|M|G|T        (default: 4G)
              (if m > max-per-cpu * cpus, more cpus are requested)
              note: that if you ask for more than one CPU has, your
account gets charged for the other (idle) CPUs as well
-t walltime  as dd-hh:mm (default: 00-02:00)
-p partition (default: defq)
-a account   if users needs to use a different account. Default is
primary PI   combined with '_' for instance: 'PI-userid'_bigmem
              (default: none)
-q qos       quality of Service's that jobs are able to run in your
association (default: qos_gpu)
-g gpu       specify GRES for GPU-based resources (eg: -g 1 )
-e email     notify if finish or fail (default: <userid>@jhu.edu)

```

# Help

**Note: We will have maintenance on 4/11/2022 to 4/15/2022.**

## RockFish Web Sites

Read our [User Guide](#) web site for how to use our system or look for specific information.

## RockFish RT System

Send us your questions at [help@rockfish.jhu.edu](mailto:help@rockfish.jhu.edu) and include as much information as possible.

For example:

- The jobid of the job with problems
- Full path to the batch submission script
- Any specific error messages
- If possible a snapshot with errors

## Frequently Asked Questions

We also have a [FAQ](#) web page where you could find the answers of your questions which most of the users asked.

# Thanks

[help@rockfish.jhu.edu](mailto:help@rockfish.jhu.edu)





[ ]