The Emory Rollins School of Public Health

High Performance Computing Getting Started Guide

(Rev. 9/16/20)

“Welcome to the RSPH Compute Cluster!”

This guide is to help new users of the (new) RSPH High Performance Computing (HPC) Cluster quickly get up and running on the cluster. This document is meant to be a gentle introduction to HPC cluster usage and intended to be supplemented by other information sources that cover topics in greater depth. For other resources, **please see the final section on “Additional Resources.”**

About the HPC Cluster

The HPC cluster is a collection of computers connected via a high speed network and sharing a common storage filesystem. It is not uncommon to refer to a cluster as a “supercomputer”, although the term today is usually reserved for very large systems.

The RSPH HPC cluster is a system that consists of (at the time of this writing) 25 *compute nodes,* 24 of which have 32 compute cores and 196GB of RAM each. The last node is a “large memory node” with 1.5 TB of RAM. These systems are connected together via 25GB Ethernet network, and all have access to a shared 1 Petabyte Panasas parallel file system. The system can support 800 concurrent running jobs which are managed by a job scheduler.

In addition to the hardware, the system runs the CentOS Linux operating system (currently version 8), which is a “white-box” implementation of the Red Hat Enterprise Linux OS that purports to be 100% binary compatible with the commercial version. Job scheduling is handled by the SLURM job scheduler, which is an application that currently runs on the majority of the Top 500 supercomputing sites in the world.

Access to the HPC cluster is free for all members of the RSPH community, although some fees may be required for storage beyond the initial allotment of 25GB per user. All RSPH faculty may request access to the cluster for themselves, and non-faculty may request accounts via sponsorship by an RSPH faculty member. Accounts are requested by emailing “help@sph.emory.edu". An active Emory NetID is required for all account recipients and should be included in the account request.

Connecting to the Cluster

To connect the the HPC cluster, one first requires access to the **Emory VPN** HIPAA-core. All users can self manage access to the general VPN by following the instructions at (http://it.emory.edu/vpntools/access.html). Once the VPN has been configured by the user and a general connection has been successfully made, the secondary access to the HIPAA-core will be granted that will allow access to the HPC cluster login node. This access is requested on behalf of the user from LITS.

All connections to the cluster are made via an encrypted connection using the **Secure Shell**, or more commonly, **ssh.** To use ssh to access the cluster, one must use a local ssh *client.* Fortunately, all major operating systems now come with natively supported ssh clients by default (!). The following operating systems have ssh clients that can be accessed easily:

1) On Mac OS X system, open a terminal app and type `ssh`
2) Under Linux, open a terminal and type `ssh`
3) Under Windows 10, open up a Powershell (search for "Powershell" in the search tool next to the Start menu) and type `ssh`. (Older versions of the operating system may require the installation of a third party ssh client, like Putty.)

All user accounts on the cluster use their respective Emory NetID as their account name. To login to the cluster, type in the respective terminal

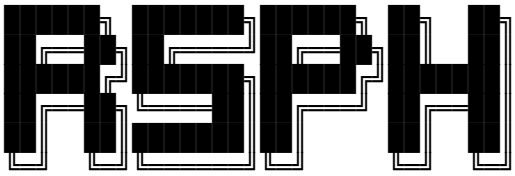
```
ssh <your_Emory_NetID>@clogin01.sph.emory.edu
```

(The '0' above in "clogin01" is a zero, not a capital 'O'.) You will be prompted for a password, and then immediately prompted to change that password to a new hard-to-guess password. **All passwords are required to be at least 10 characters long and contain at least one character each from lower case letters, upper case letters, digits and punctuation.** If your new password is rejected for any reason, the system will end your login attempt and you will have to try again.

Once you have successfully logged into the system, you will see the banner screen and a command prompt:

```
$ ssh testuser@clogin01.sph.emory.edu
testuser@clogin01.sph.emory.edu's password:

Welcome to the
```



```
High Performance Computing (HPC) Cluster

***  AUTHORIZED USE ONLY ***


.
Last login: Tue Aug 11 16:02:19 2020 from 10.110.100.86
[testuser@clogin01 ~]$
```

You may then start submitting commands to the system.

Copying Data to the Cluster

The supported method for copying data to the cluster is with the **scp** or "secure copy" terminal command, which provides encrypted transport of data to and from the cluster. All three major client operating systems (in their up-to-date versions) have the scp client installed alongside ssh by default. To use the **scp** command you may either *push* or *pull* data to and from the cluster securely over remote networks. The general format for copying a file to a remote system takes the format of

scp <path_to_file> <username>@<remote_system_name>:<destination_path>

So for example to copy a tile "testscript.sh" from your current directory on, say, a Mac laptop to the cluster (as the user "testuser"), you'd use

```
scp testfile.sh testuser@clogin01.sph.emory.edu:.
```

which after authentication would copy the file testfile.sh to testuser's home directory (designated here by the dot "." after the colon) on the cluster. (Substitute your Emory NetID for "testuser" in the above example for your own copying, of course.)

Available Software

There are many available software packages on the cluster, not including those that you may wish to install locally in your home directory. Some of the larger, more popular software we have configured to use a system called **modules**. The modules system allows users to select both software packages and their particular versions in a way that automatically sets the path and other environment variables for that application.

Assume, for instance, that a user would like to use the R programming language:

```
testuser@clogin01 ~]$ R
bash: R: command not found…
```

By using the module spider command, we can see what packages are installed cluster-wide:

```
testuser@clogin01 ~]$ module spider

----------------------------------------------------------------
The following is a list of the modules and extensions currently available:
----------------------------------------------------------------
  R: R/4.0.2
    R is a free software environment for statistical computing and graphics.
```

```
  intel: intel/2020.2.254

  intelmpi: intelmpi/20200624
    Intel MPI

  julia: julia/1.0.5
    Julia is a high-level, high-performance dynamic language for technical
computing.

  lmod: lmod
    Lmod: An Environment Module System

  openmpi/gcc/4.8.5: openmpi/gcc/4.8.5/4.0.3
    OpenMPI open source Message Passing Interface implementation

  openmpi/intel/2020.2.254: openmpi/intel/2020.2.254/4.0.3
    OpenMPI open source Message Passing Interface implementation

  python: python/3.8
    Python is an interpreted, high-level, general-purpose programming
language.
```

We can then use the **`module load`** command to load the R module, and then run R:

```
[testuser@clogin01 ~]$ module load R
[[testuser@clogin01 ~]$ R

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
…
```

In this example, we simply loaded the 'R' module as there was only one present and it was unambiguous. If for instance we had more than one version of R listed, such as R/3.6.0, we could have specified a version, e.g., '`module local R/3.6.0`'.

You can use the "module spider" command for time to time to see what new (versions of) software are installed on the system.

SLURM and Submitting Jobs

In order to run a computation on the HPC cluster, you must submit the work as a “job”. This is in contrast to work done on a traditional computer like a laptop or an office workstation, where work is done primarily interactively and via a graphical user interface (GUI). Large jobs or numbers of computations are submitted in “batches”.

The job scheduler SLURM does the work of running all jobs submitted to the cluster. It is very similar in function to the Grid Engine scheduler seen on other clusters. SLURM is a fault-tolerant cluster management and job scheduling system that allocates resources to compute nodes so that they can run programs or jobs. It also manages contention for cluster resources by managing a queue of pending work.

A List of Basic SLURM Commands

sbatch command used to submit jobs (like 'qsub' in Grid Engine)

squeue used to display information about the run queue

scancel used to cancel or kill a job

scontrol used to show information about running or pending jobs

srun used to run an interactive instance

sinfo used to report the state of the cluster partition and nodes

Once logged into the system via ssh, you can do a quick check on the system via the **sinfo** command:

```
[testuser@clogin01 ~]$ sinfo
PARTITION       AVAIL  TIMELIMIT  NODES  STATE NODELIST
month-long-cpu     up 31-00:00:0     24   idle node[1-24]
week-long-cpu      up 7-00:00:00     24   idle node[1-24]
day-long-cpu       up 1-00:00:00     24   idle node[1-24]
short-cpu*         up      30:00     24   idle node[1-24]
interactive-cpu    up 2-00:00:00     24   idle node[1-24]
```

This display shows all of the partitions (i.e., “queues” in GE) on the system with the number nodes, their availability and status. The cluster isn’t being used at all at the moment. Here is a more typical **sinfo** output:

```
$ sinfo
PARTITION       AVAIL  TIMELIMIT  NODES  STATE NODELIST
month-long-cpu     up 31-00:00:0      1   comp node13
month-long-cpu     up 31-00:00:0     18    mix node[1-12,14-19]
month-long-cpu     up 31-00:00:0      5   idle node[20-24]
week-long-cpu      up 7-00:00:00      1   comp node13
week-long-cpu      up 7-00:00:00     18    mix node[1-12,14-19]
week-long-cpu      up 7-00:00:00      5   idle node[20-24]
day-long-cpu       up 1-00:00:00      1   comp node13
day-long-cpu       up 1-00:00:00     18    mix node[1-12,14-19]
day-long-cpu       up 1-00:00:00      5   idle node[20-24]
short-cpu*         up      30:00      1   comp node13
….
```

The command **sbatch** is used to submit jobs to the job scheduler. There are a number of parameters that can be passed to **sbatch** via the command line or separately in a shell script. Both methods can be used simultaneously, but any settings on the command line will prevail.

Here is a basic example of a python script that simply prints a line containing the hostname of the compute node on which it ran:

```
$ cat HelloWorld.py
#! /apps/bin/python3
import socket

hostname = socket.gethostname()

print("Hello World from " + hostname + "!\n")
```

If we submit this job without any additional arguments, we can see the results almost immediately in the directory from which we submitted it:

```
$ sbatch HelloWorld.py
Submitted batch job 3026
$ ls
HelloWorld.py  sleeper.sh  slurm-3026.out
$ cat slurm-3026.out
Hello World from node1!
```

When a job is submitted, **sbatch** returns a job ID, which is then associated with all aspects of the job. In this case, the output of the HelloWorld.py program was written to the output file “slurm-3026.out”, which is the default behavior. To change the default behavior, we can add arguments to the sbatch command:

```
$ sbatch --partition=short-cpu --output=hello-%j.out
HelloWorld.py
Submitted batch job 3027
$ ls
hello-3027.out  HelloWorld.py  sleeper.sh  slurm-3026.out
$ cat hello-3027.out
Hello World from node1!
```

Here we told the job scheduler to change the name of the output file to “hello-<jobid>.out” and designated the “short-cpu” queue as the place to run the very short program. It is a good idea to specify the appropriate queue for running jobs based on the anticipated time to complete the computation.

To check on the system details of a submitted job, we can use the **scontrol** command:

```
$ scontrol show job 3027
JobId=3027 JobName=HelloWorld.py
   UserId=testuser(3004) GroupId=hpcusers(3001) MCS_label=N/A
   Priority=4294901740 Nice=0 Account=(null) QOS=(null)
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:00 TimeLimit=00:30:00 TimeMin=N/A
   SubmitTime=2020-08-17T11:42:38 EligibleTime=2020-08-17T11:42:38
   AccrueTime=2020-08-17T11:42:38
   StartTime=2020-08-17T11:42:38 EndTime=2020-08-17T11:42:38 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2020-08-17T11:42:38
   Partition=short-cpu AllocNode:Sid=clogin01:556227
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=node1
   BatchHost=node1
   NumNodes=1 NumCPUs=1 NumTasks=0 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=192079M,node=1,billing=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:1 CoreSpec=*
   MinCPUsNode=1 MinMemoryNode=192079M MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/testuser/HelloWorld.py
   WorkDir=/home/testuser
   StdErr=/home/testuser/hello-3027.out
   StdIn=/dev/null
   StdOut=/home/testuser/hello-3027.out
   Power=
   MailUser=(null) MailType=NONE
```

This output is sometimes useful in determining what may have happened to a job during its submission.

You can also use **scontrol** to display the configuration of a partition:

```
$ scontrol show partition day-long-cpu
PartitionName=day-long-cpu
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=1-00:00:00 MinNodes=0 LLN=NO
MaxCPUsPerNode=UNLIMITED
   Nodes=node[1-24]
   PriorityJobFactor=20000 PriorityTier=20000 RootOnly=NO ReqResv=NO
OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=768 TotalNodes=24 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

To cut down on typing and for more complicated parameters, one can set the arguments to **sbatch** in a wrapper or *job submission script*, which uses a combination of shell commands and SLURM directives, which begin with "#SBATCH". These can get very complex, but here is an example that covers some basic parameters:

```
#!/bin/bash

#SBATCH --job-name=normal.R
#SBATCH —partition=day-long-cpu

## This puts all output files in a separate directory.
#SBATCH --output=Out/normal.%A_%a.out
#SBATCH —error=Err/normal.%A_%a.err

## Submitting 100 instances of srun commands listed below
#SBATCH —array=0-100

## For notification purposes.  Use your Emory email address only!
#SBATCH —mail-user=<your_email_address>@emory.edu.
#SBATCH --mail-type=END,FAIL


module purge
module load R

srun /home/<user>/normal.R
```

This example shows how one might submit an R computation named "normal.R". This job submission script not only sets the values for several parameters such as job name, run partition, and output file names, but it allows for submission of an "array" of jobs, which is useful if one is running a large number of identical or similar computations. This script also demonstrates the use of modules in a script to ensure that your program has the appropriate path set.

Once we have our job submission script in place, we can use sbatch to submit the job (assuming we have named the above script “normal.sh”):

```
$ sbatch normal.sh
Submitted batch job 3028
```

And use the command **squeue** to monitor the progress:

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
          3028_22 day-long- normal.R  testuse CG       0:01      1 node23
          3028_24 day-long- normal.R  testuse CG       0:01      1 node10
           3028_2 day-long- normal.R  testuse CG       0:02      1 node3
           3028_4 day-long- normal.R  testuse CG       0:02      1 node5
          3028_11 day-long- normal.R  testuse CG       0:02      1 node12
          3028_14 day-long- normal.R  testuse CG       0:02      1 node15
          3028_19 day-long- normal.R  testuse CG       0:02      1 node20
          3028_20 day-long- normal.R  testuse CG       0:02      1 node21
     3028_[36-100] day-long- normal.R  testuse PD       0:00      1
(Resources)
          3028_25 day-long- normal.R  testuse  R       0:01      1 node13
          3028_26 day-long- normal.R  testuse  R       0:01      1 node17
          3028_27 day-long- normal.R  testuse  R       0:01      1 node18
          3028_28 day-long- normal.R  testuse  R       0:01      1 node11
          3028_29 day-long- normal.R  testuse  R       0:01      1 node22
          3028_30 day-long- normal.R  testuse  R       0:01      1 node4
```

Here, under “ST” for job state, we see that some array jobs are running (R), others are completing (CG), and one is still pending (PD).

Finally, but perhaps most importantly, to cancel a job submission, one uses the **scancel** command:

```
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              3231 short-cpu  sleeper testuser  R       0:44      1 node1
              3232 short-cpu  sleeper testuser  R       0:36      1 node1
              3233 short-cpu  sleeper testuser  R       0:33      1 node1
              3234 short-cpu  sleeper testuser  R       0:33      1 node1
[testuser@clogin01 ~]$ scancel 3232
[testuser@clogin01 ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              3231 short-cpu  sleeper testuser  R       0:52      1 node1
              3233 short-cpu  sleeper testuser  R       0:41      1 node1
              3234 short-cpu  sleeper testuser  R       0:41      1 node1
```

The **scancel** command takes a job id as its argument, and is pretty straight forward.

For more detailed options and explanations about SLURM commands, please consult the referenced documentation in “Additional Resources” below.

Interactive Terminal Sessions (via `srun`)

Sometimes one would like to test code before submitting jobs widely to the cluster. In this case and interactive terminal sessions may be useful. To run an application on the cluster in an interactive session you can use **srun** on the **interactive-cpu queue**, which will start a terminal session for you on a compute node. Here is an example of running R in an interactive session:

```
[user@clogin01 ~]$ module load R
[user@clogin01 ~]$ srun -p interactive-cpu --pty bash
[user@node1 ~]$ R

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> quit()
Save workspace image? [y/n/c]: n
```

When you have finished with your session, simply exit the shell from the execution node:

```
[user@node1 ~]$ exit
exit
[user@clogin01 ~]$
```

Interactive applications like R and MATLAB are not permitted on the login node clogin01. Please always use an interactive session on a compute node when running interactive applications.

Storage on the Cluster

The storage is presented via three volumes for access by users on the cluster:

- The `/home` file system, where every cluster user has a directory for their account
- The `/scratch` file system, where users may load data for immediate computation
- And the `/projects` file system, where groups may store project data for a period of time and also run computations.

There is also a fourth volume for applications, `/apps`, which is readable by cluster users. This is where binaries for cluster-provided software can be found, although it is preferred that applications be accessed via the `module` command.

The Isilon volumes are presented in the `/isilon` top-level directory on the login node only (and are therefore not suitable for cluster computation). At present the directories `/isilon/home` and `/isilon/projects` (the `/home` and `/projects` directories for the old cluster, respectively) are available for copying of data. Other Isilon directories may be mounted by arrangement with the HPC Cluster management.

Where do I put my data?

By default, all users of the cluster have a 25GB quota for their home directory located in `/home/<userid>`. There are no time restrictions on data in your home directory, so data may reside in your home directory as long as the account is active. The home directories reside in the PanFS file system which is mounted across all nodes of the cluster, so it is suitable to run computations from your home directory location provided you have enough space in your quota to write your output files.

For computations where a larger cluster-wide computation space is needed, the `/scratch` file system is available. This shared space is usable for large computations that are limited in time scope, as there is a **two-week maximum retention policy** for files in `/scratch`. Space in /scratch is available on a first-come-first-use basis for users that request quota space there. (Please contact the HPC management for possible policy exceptions.)

For reserved space exceeding the home directory quota, and for shared project space, the `/projects` directory is recommended. Faculty may request quota in `/projects` for use for their research or their sponsored accounts in increments of 1 TB. Files stored in `/projects` may remain for one year, depending on the quota arrangement.

Performance-wise, there is no difference between the /home, /scratch and /projects volumes. All are PanFS volumes mounted across the cluster, and all take advantage of the features of the Panasas storage. The major differences are in the storage policies.

Quotas and Cluster Policy

Below is a table of the user-writable storage volumes on the HPC Cluster.

Volume	Quota	Purge Policy	Suitable for Job I/O
/home	25 GB/user	None	Yes
/project	1 TB/PI or Group*, additional fee for use option $75/TB/year	Annual renewal	Yes
/scratch	100GB/usr default quota*	2-week	Yes
/isilon	Purchased from LITS	None	NO

*Increases available via request

Monitoring your Storage Quota

The parallel file system allows users to self-monitor their quota via the **pan_quota** command:

```
[user@clogin01 MyTest]$ pan_quota

  <GB> <soft> <hard> : <files>    <soft>    <hard> :     <path to volume> <pan_identity(name)>
  9.30  24.00  25.00 :  121845 unlimited unlimited : /home/user/MyTest uid:99999(user)
```

The first number listed above is the current amount of storage consumed by the user. This number is computed on the fly by the storage system and may change if storage is currently being written to the system. The third number, under <hard>, is the actual limit on the storage. Upon consuming this amount, the user will no longer be able to write to the file system. The second number (<soft>), is the amount at which the system will send the user email warnings.

NOTICE: **Data on the PanFS file systems on the cluster are not backed up.** *All users are encouraged to keep and manage their own backups of any vital data.* While a great deal of effort and expense have gone to provide a very redundant and robust

cluster file system, accidents do occur. If you overwrite a file you may be able to recover it if discovered immediately via the system snapshots feature (see the section on **Snapshots** below). If you have further questions about how to best protect data used on the HPC Cluster, please contact the HPC Cluster management.

Data Archiving Options

Users have a number of options for archiving data from the cluster filesystem, and once a project or job run is completed it is highly recommended to migrate data off of the cluster file system.

A highly responsive and University-supported archive location is the LITS Isilon storage which is mounted on the login node at `/Isilon`. Quota in the `/Isilon/projects` directory can be negotiated with the IT group for suitable long term archiving of data in a secured, redundant storage environment.

Researchers may also independently manage off-site or cloud-based storage such as AWS S3 or Backblaze B2 “buckets”, which are sometimes cost effective and come with high-availability guarantees. **Researchers are reminded to follow all university guidelines with regards to research data management, especially with respect to Federally-protected and sensitive data.**

Snapshots

The cluster file system currently generates and retains two snapshots of data in the /home file system in the hidden directory “`.snapshot`”. For instance:

```
[user@clogin01 ~]$ ls /home/.snapshot
2020.07.28.23.59.01.Daily_Home  2020.07.29.23.59.01.Daily_Home
```

Shows the two most recent snapshots (as of the date in this example) of the home directory file system. If a user accidentally deletes a file, they can restore one of two possible copies from the snapshots of their home directories in the above locations.

Additional Resources

To receive cluster assistance or request an account, email help@sph.emory.edu

The Emory VPN:

http://it.emory.edu/vpntools/access.html

Modules:

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

SLURM:
https://slurm.schedmd.com/quickstart.html

Linux:

Learning the Linux Command Line, a video training module accessible to all Emory-affiliates persons via LinkedIn learning.
(*http://it.emory.edu/linkedinlearning/help/index.html*)