Clusters 101

Purdue University - ITaP

Gladys Andino
Dan Dietz
Jieyu Gao
Lev Gorenstein
Erik Gough
Stephen Harrell
Randy Herban
Steve Kelley
Boyu Zhang
Xiao Zhu

rcac-help@purdue.edu

March 6-10, 2017

Slides available:
www.rcac.purdue.edu/tutorials/clusters101/
A few additional acknowledgments to the many people who have helped make this workshop possible.

- The material in this workshop was prepared by the Purdue University ITaP Research Computing team.
- Special thanks to Megan Dale for organizing the workshop sessions.
Vocabulary

Logging In

Modules

Types of Work

Good Citizenship

Jobs, Queues, and PBS

Running Jobs

Job Monitoring

Interactive Jobs and Thinlinc

Storage

Globus

Workflows
A cluster is a cluster is a cluster?

- Hardware (compute nodes + interconnect + storage)
- Software (OS + compilers + libraries + apps + queue manager)
- Infrastructure (front-ends + power + cooling + data center + staff)
Node:

- Network card
- Memory
- Processors
- Memory
- HDD
- Intel Xeon Phi accelerator
- Power supply
Vocabulary

Logging In
Modules
Types of Work
Good Citizenship
Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and Thinlinc
Storage
Globus
Workflows

CPU:
Vocabulary

Processor Core: individual compute unit ("slot") on the chip

- You see:
  2 physical processors, 8 cores ea.

- Queuing system sees:
  16 logical processors
  (hence 'nodes=X:ppn=16' effectively stands for "processor cores per node").

- From now on, we will be mostly concerned with cores (logical processors), not physical chips
  ("I ran on 5 CPUs" == "on 5 processors" == "on 5 cores")
Vocabulary

Front-end vs compute node:

“Front-end” shared by many

“Back end”

Running Jobs: The goal is getting to the compute nodes
IBM SYSTEM 360 (1 Computer, 1 Processor):
Community clusters:

**HPC (Halstead):** Multiple cores or nodes, probably MPI. Benefit from high-performance network and parallel filesystem. The vast majority of campus - 80% of all work!

**Data-Intensive Life Science (Snyder):** Use entire node to get large amounts of memory. Less need for high-performance network. Needs large, fast storage.
Vocabulary

Need help with ITaP Research Computing resources?

Check our User Guides and documentation:
www.rcac.purdue.edu/knowledge

You can always send us an email:
rcac-help@purdue.edu

Come to our coffee hour consultations:
www.rcac.purdue.edu/coffee
Logging In

- Windows
- Mac
- Activity Files
Logging In

We will be using the Radon cluster:

- [www.rcac.purdue.edu/compute/radon/](http://www.rcac.purdue.edu/compute/radon/)
- Everyone has been given an account on the cluster for the duration of the workshop
- If you wish to continue using Radon or other cluster after the workshop concludes, please make a request under your advisor or PI’s name: [https://www.rcac.purdue.edu/account/request/](https://www.rcac.purdue.edu/account/request/)
Many clients are available for Windows:

- We will use the PuTTY SSH client
- Download PuTTY, no install required
- [http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html](http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) (or Google search *putty*)
- Download `putty.exe` for Intel x86 to your desktop
Host Name for Radon is radon.rcac.purdue.edu
Logging In

Mac

Connect using:

```
ssh myusername@radon.rcac.purdue.edu
```
Linux also has a built in terminal client, similar to Mac:

```
ssh myusername@radon.rcac.purdue.edu
```
We’ll need a few files for some of the hands-on activities

```
$ cd
$ cp -r /depot/itap/clusters101 .
```
Vocabulary
Logging In

Modules

Types of Work
Good Citizenship
Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and Thinlinc
Storage
Globus
Workflows
All ITaP Research Computing clusters use a module system

- Module system provides for the dynamic modification of a user’s environment
- Module commands allow you to add applications and libraries to your environment
- This allows us to simultaneously and safely provide several versions of the same software
- You can set up the environment for running applications using simple commands:

```bash
$ module load intel
$ module load openmpi
```

- All ITaP Research Computing clusters have a default programming environment (compiler and MPI library) loaded for you when you log in.
Show all modules currently loaded in my environment:

```bash
$ module list
```

Currently Loaded Modules:
1) intel/16.0.1.150 2) openmpi/1.8.1 3) rcac

Load a new module:

```bash
$ module load matlab
$ module list
```

Currently Loaded Modules:
1) intel/16.0.1.150 2) openmpi/1.8.1 3) rcac
4) matlab/R2016a
Module commands:

Shows you all software modules currently available to you:

$ module avail

Filters software modules currently available to you by a search term. In this case, show all Python versions available:

$ module avail python

--------- Applications built with Intel 16.0.1.150 ---------
   python/2.7.8

--------------------- Core Applications ---------------------
   python/anaconda (D)   python/2.7.2   python/3.4.1
All ITaP Research Computing clusters use an hierarchical software configuration. Modules are organized by the programming environment (compiler and libraries) that they were built with and depend on. You will only see (with avail) modules compatible with the programming environment currently loaded.

As you swap in other programming environments (compiler and libraries), you may find additional software available to you. Any libraries or software dependent on the programming environment will be automatically swapped with compatible versions as you change environments.
We can look for software not immediately available to us - that is, things that may not show up in `avail` output:

```
$ module avail gromacs
$ module spider gromacs
```

```
--------------------------------------------------------------
gromacs: gromacs/5.0
--------------------------------------------------------------
```

You will need to load one of the set of module(s) below before the "gromacs/5.0" module is available to load.

```
intel/13.1.1.163 openmpi/1.8.1
```

This tells us we must swap into a new programming environment to load gromacs:

```
$ module load intel/13.1.1.163 openmpi/1.8.1 gromacs/5.0
```
Unload a currently loaded module:

```bash
$ module unload matlab
$ module list
```

Currently Loaded Modules:
1) intel/13.1.1.163   2) openmpi/1.8.1   3) rcac
4) gromacs/5.0

Unload and purge all currently loaded modules:

```bash
$ module purge
$ module list
No modules loaded
```
What exactly does `module load` do?

It will change the environmental variables defined for that module. Typically these are `$PATH` and `$LD_LIBRARY_PATH`. Also a few extra module-specific environmental variables could be set.

```bash
$ module show vmd
```

```
/opt/modulefiles/core/vmd/1.9.1.lua:

whatis          invoke VMD 1.9.1
setenv          VMD_HOME  "/apps/rhel6/vmd-1.9.1"
prepend_path    PATH  "/apps/rhel6/vmd-1.9.1/bin"
prepend_path    LD_LIBRARY_PATH  "/apps/rhel6/vmd-1.9.1/lib"
setenv          LIBGL_ALWAYS_INDIRECT  "y"
```
A few tips from us:

- Don’t load modules in `.bashrc/.profile/.login/.cshrc`
- Don’t load more than needed for current task
- Do `module purge` when in doubt
- Do check with `module show` what a given module does
- Do check with `module list` what's actually loaded
- Do script and automate your builds - even if it's just a 3-line snippet! Reproducibility is good!
Modules

Exercises:

1. List currently loaded modules
2. List available modules
3. Search available modules for matlab
4. Show what one of the matlab modules does
5. Load the gromacs module (hint: spider)
6. Purge all modules
7. List currently loaded modules
Exercises:

1. List currently loaded modules
   
   ```
   $ module list
   
   Currently Loaded Modules:
   1) intel/16.0.1.150  2) openmpi/1.8.1  3) rcac
   ```

2. List available modules
   
   ```
   $ module avail
   ...
   ```

3. Search available modules for matlab
   
   ```
   $ module avail matlab
   
   ---------------- Core Applications ----------------
   matlab/R2013a  matlab/R2016a (D)
   ```
4. Show what one of the matlab modules does

$ module show matlab

5. Load the gromacs module (hint: spider)

$ module spider gromacs
$ module load intel/13.1.1.163 openmpi/1.8.1 gromacs
Due to MODULEPATH changes the following have been reloaded:
  1) openmpi/1.8.1

The following have been reloaded with a version change:
  1) intel/16.0.1.150 => intel/13.1.1.163

6. Purge all modules

$ module purge

7. List currently loaded modules

$ module list
No modules loaded
Types of Work

- Serial vs Parallel
- Parallel Computations
Types of Work

Serial vs Parallel

More than one mower!
Mow the lawn faster ...
... or mow a bigger lawn
Lawn mowers = cores
Types of Work
Serial vs Parallel

More than one mower!
Mow the lawn faster ...
... or mow a bigger lawn
Lawn mowers = cores
Parallel computations come in a few flavors:

- MPI - Message Passing Interface, multiple nodes
- OpenMP - confine yourself to one node
- Hybrid - mix MPI and OpenMP
- Accelerators/coprocessors - GPU, Intel Xeon Phi
- Parallel R, Matlab, Hadoop and Spark
Good Citizenship

Vocabulary
Logging In
Modules
Types of Work

Good Citizenship

Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and Thinlinc
Storage
Globus
Workflows
Good Citizenship

**Front-end vs compute node:**

- **Front-end** e.g. `radon-fe01`
  - Internet
  - `ssh`
  - `batch job`
  - `queue`

- **Compute Nodes** e.g. `radon-e000`

**File system** (\$HOME, \$RCAC_SCRATCH)

**Running Jobs:** The goal is getting to the compute nodes
Good Citizenship

What does this mean?

- Do not do science on the front end
- Front-ends are very limited resources and you'll annoy the system administrators and other users
- Watchdog will monitor and quickly stop any inappropriate resource intensive process
- Instead: submit a job. We'll get to that
Good Citizenship

What can I do on the front-ends?

- Building/compiling software
- Managing files: editing, transferring, `tar`, `gzip`, `hsi`
- Submitting, monitoring, and managing batch jobs
- Launching interactive jobs
- Modest post-processing and analysis
Jobs, Queues, and PBS

- Jobs
- Queues
- PBS
- Life of a Job
What is a job?

- A job is simply a set of tasks to be performed by a cluster
- A script to instruct the cluster precisely what to do to complete your work
- Self contained to be executed without any interaction
- Submit and forget!
Remember, cluster front-end nodes are shared resources for
- Creating, submitting, and monitoring jobs
- File transfers
- Preparing inputs
- Editing and compiling code
- Small-scale testing

May be used by 50+ people simultaneously
- Check out the `who` command
- Jobs should have no interference by other people
- Want jobs carefully arranged on compute nodes
Jobs, Queues, and PBS

Jobs

Jobs are submitted to the cluster:

- Cluster executes jobs on back-end compute nodes
- Jobs are carefully scheduled and arranged on the compute nodes
Jobs need to specify the resources they require:

- Three basic units:
  - Number of nodes
  - Number of cores
  - Time

- Memory

- Node access policy

- Other resources

Cluster will allocate requested resources once they are available. Job starts once resources are allocated.
Jobs, Queues, and PBS

Jobs

The most basic unit you can request is nodes:

- Serial applications probably only need one node
- Parallel applications may need 10s or 100s of nodes

Job should also request how many cores per node it needs:

- Cluster allows your job to use only the cores you request
- Often your job should request all of the cores in a node
- Remember, we want to avoid other people!
Nodes are physically different pieces of hardware. A job with 16 nodes and 16 cores each, is **not** 256 threads

- 16 individual computers!
- Program needs to be written with MPI to tie all the nodes together

![Diagram of nodes and processors](image)
Nature of the job: how much inter-process communications? 1 core on 16 nodes, or for 16 cores on 1 node?

- Communication inside a node is faster than between nodes
- Collisions with other people’s jobs (16 nodes x 15 jobs = 240 jobs)
- Most performant, most predictable, least vulnerable, least interfering

Tight packing preferred. Most ITaP clusters enforce exclusive nodes unless explicitly specified for single node jobs.
Jobs, Queues, and PBS

Jobs

The third basic unit jobs can request is walltime. The cluster won’t let your job run forever!

Cluster needs to have a time limit for your job:

- Helps the cluster efficiently schedule you on the nodes
- Make request accurate with some safety buffer
- Tune your walltime request through testing and experimenting
Jobs, Queues, and PBS

Jobs

Jobs need memory, but...

- Clusters can not limit your job to an amount memory
- All of a compute node’s memory is available to your job
- All of that memory is also available to any other job on the node!
- 1 core job needs $16 \text{ GB} \times 16 \text{ jobs} = 256 \text{ GB}$
  
  But wait, my node only has 16 GB!!

Jobs crash and burn
Jobs, Queues, and PBS

Jobs

Memory across multiple compute nodes is distributed. Memory is discrete chunks of memory in different address spaces.

Can’t just ”add more nodes” to get more memory:

- 16 nodes with 16 GB memory each is not 256 GB of memory
- 16 discrete chunks of 16 GB memory
- Can’t allocate 256 GB of memory to a single process
- Need MPI to tie memory together
Jobs, Queues, and PBS

Jobs

Best bet is to always request all cores of a node or request exclusive access with the node access policy. You get full control of the node and all of its memory. You can try requesting proportionate numbers of cores but not a guarantee!!

Another option is to use the node access policy to request that only your own jobs can share nodes. You don’t have to worry about somebody else getting in the way, but you do have to be mindful of what resources your own job is using.
There are a number of other resources you can request in a job:

- Accelerators (GPUs, Xeon Phis)
- Specific types of nodes
- Certain licenses: MATLAB and its Toolboxes
Cluster can’t run all jobs at once so sometimes you must wait

- Jobs are submitted into queues
- Jobs wait until cluster and queue has free resources
- Queues set constraints on jobs that can be submitted
  - Max nodes
  - Max walltime
- Sets initial priorities
- Our clusters have several queue types
Jobs, Queues, and PBS

Queues

Owner queues:

- Named for your PI, lab, or group
- Number of nodes set by what your PI buys into
- Allows for longest jobs - 336 hours
- Jobs should start in 4 hours or less (assuming your queue isn’t full of other jobs in the queue)

Queues are **not** tied to specific nodes in the cluster. They allow you to use the number of nodes somewhere in the cluster - whichever nodes are free, functioning, and of the type the queue can access.
Standby queue:

- Uses idle nodes from owner queues
- Everyone on cluster gets access
- Limited to 4 hours (owners must be able get quick access from their queue)
- No limit on number of nodes (probably won’t get whole cluster though)
- Lowest priority jobs, no promises on turnaround time (can be minutes or days)
- If you can run under 4 hours, go for it!
Debug queue:

- Allows for running small jobs for testing and debugging
- Up to 2 nodes for 30 minutes at a time
- Highest priority
- Starting time <1 minute
In this workshop we are using Radon, a general access cluster. It has none of the above queues!

workq:

- Sort of like standby
- Not competing with owner queues
- Longer jobs than standby

Downside to a more open and available queue is Radon has an order of magnitude less nodes than Community Clusters. Each node is also less powerful than its bigger siblings. It is free, but isn’t as capable.

For this workshop we have set up a special workshop queue. Jobs in it are given priority and have several nodes reserved.
What is PBS?

- Portable Batch System
- Original developed for NASA in 1991
- Open source fork and commercial fork remain
- We use the open source fork, TORQUE

[en.wikipedia.org/wiki/Portable_Batch_System](en.wikipedia.org/wiki/Portable_Batch_System)
What is PBS, really? It is the resource manager for the cluster:

- Job submission
- Job status
- Executes jobs
- Manages queues
- Manages nodes
The scheduler is actually separate software. We use a scheduler called Moab. It decides which jobs to run and where.

"PBS" == "Scheduler" == "Batch system" == "Server"

- We often interchange them in conversation
- All refer to the whole system of servers, schedulers, and managers
You prepare a job submission script
You submit job script into PBS. The job is placed into a queue.
Scheduler iteratively asks PBS for new jobs, status of old jobs, and status of nodes. Each cycle can last several minutes.
Jobs, Queues, and PBS

Life of a Job

Scheduler considers jobs, acts as a handler. Tells PBS to start jobs one at a time. Goes back and does this over and over.
PBS checks health of the node. Send job script to node(s) and executes it.
PBS watches for the job script to complete on nodes. Collects job output files from nodes and sends files to you.
Running Jobs

- Submission File
- Submitting
- Environment Variables
- Tips
- Example
Running Jobs

Submission File

To run a job you must first create a submission file. This file instructs the cluster everything it needs to know to run your job.

Components of a submission file:

- PBS directives: specify resources needed such as number of nodes, cores, and time
- Module load to set up paths, libraries
- PBS environment variables
- Customized commands to make your job run
 PBS directives set PBS job attributes. They appear at the top of your submission file and are prefaced with #PBS in your submission file or could be specified at submit time as arguments.

Common PBS job attributes include the following. None are absolutely required, but you should always set the following three. Without them, PBS will assume defaults which are likely not useful to you.

<table>
<thead>
<tr>
<th>PBS Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-l nodes=2:ppn=8</td>
<td>Number of nodes and cores</td>
</tr>
<tr>
<td>-l walltime=00:10:00</td>
<td>Maximum walltime allowed</td>
</tr>
<tr>
<td>-q workshop</td>
<td>Destination queue</td>
</tr>
</tbody>
</table>
Running Jobs
Submission File

There are also several other directives you can optionally set. Some of the more common ones follow. As usual `man qsub` will document all available options.

<table>
<thead>
<tr>
<th>PBS Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-N jobname</td>
<td>Give your job a custom name</td>
</tr>
<tr>
<td>-o /path/to/output.out</td>
<td>Specify the job output file</td>
</tr>
<tr>
<td></td>
<td>Can use <code>$PBS_JOBID</code> to make unique</td>
</tr>
<tr>
<td>-e /path/to/error.err</td>
<td>Specify the job error file</td>
</tr>
<tr>
<td></td>
<td>Can use <code>$PBS_JOBID</code> to make unique</td>
</tr>
<tr>
<td>-l naccesspolicy=policy</td>
<td>Node access policy</td>
</tr>
<tr>
<td>-n</td>
<td>Request exclusive access to the node</td>
</tr>
</tbody>
</table>
naccesspolicy can be set to one of the following:

- **shared**: allowing sharing of a node between jobs (default on Radon)
- **singlejob**: only allow one job per node (default on most other ITaP clusters)
- **singleuser**: only allow one user’s jobs on a node

Node access policies are explained in more detail on our user guide:

[www.rcac.purdue.edu/knowledge/radon/run/pbs/naccesspolicy](http://www.rcac.purdue.edu/knowledge/radon/run/pbs/naccesspolicy)
Running Jobs

Submission File

After requesting resources, your submission script should load required modules to set up the environment for your application or code. Pretend you have just logged in and have nothing set up yet.

It may be a good idea to purge any currently loaded modules (whether from your login scripts or system default) so you get a predictable and reproducible environment.

```bash
#PBS -l nodes=1:ppn=1
#PBS -q workshop
module purge
module load r
```
Now that you have resources requested and your environment set up, `cd` to where your files are, then run the commands necessary to do your work.

A job to run an R script:

```
#PBS -l nodes=1:ppn=1
#PBS -q workshop
module purge
module load r

cd ~/myproject/  # PBS starts job in home directory
    # so need to move where project files are
Rscript my_script.r arg1 arg2
```
Once your job submission script is written you can submit it with the `qsub` command:

```bash
$ qsub myjob.sub
1234567.radon-adm.rcac.purdue.edu
```

If submission was successful, you should get back the Job ID that PBS has assigned to your job. Only the numerical part is strictly required - the rest of the ID identifies which cluster the job is on but is not required for further operations on the job.

Job scripts do not require the `.sub` extension, however, it is convenient to make it easy to identify submission files.
You can also specify PBS directives directly to `qsub` rather than in the job file. Any directives that will never change should be put in the file, however, by specifying directly to `qsub` you can use the same submission file with slight variations on one or more of the directives.

Here we can specify different job names for the same job script. We’ll see in one of the following examples how we can take advantage of this.

```
$ qsub -N input_1 1234567.radon-adm.rcac.purdue.edu
$ qsub -N input_2 1234568.radon-adm.rcac.purdue.edu
$ qsub -N input_3 1234569.radon-adm.rcac.purdue.edu
```
PBS defines several variables automatically for you. They can be used within your job submission file. Some common variables include:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_O_WORKDIR</td>
<td>Absolute path of the current working directory when you submitted this job</td>
</tr>
<tr>
<td>PBS_JOBID</td>
<td>Job ID assigned to this job by PBS</td>
</tr>
<tr>
<td>PBS_JOBNAME</td>
<td>Job name (defined by you or PBS)</td>
</tr>
<tr>
<td>PBS_NODEFILE</td>
<td>Path to a file containing the list of nodes assigned to this job</td>
</tr>
</tbody>
</table>
$PBS_O_WORKDIR contains the path to the directory form which you submitted the job. $PBS_O_WORKDIR can be used to cd into that directory within the job.

When a job starts it will default to your home directory, so you likely need to cd somewhere first.

```bash
#PBS -l nodes=1:ppn=1
#PBS -q workshop

# Change to the directory from which you submitted this job
cd $PBS_O_WORKDIR

# Print out the current working directory path
pwd
```
Running Jobs

Environment Variables

$PBS_JOBID can be used to store output from a program in a uniquely named file. In this way you can reuse the same job file across multiple inputs and run several instances simultaneously without overwriting output from the other jobs.

```
#PBS -l nodes=1:ppn=1
#PBS -q workshop

# Change to the directory from which you submitted this job
cd $PBS_O_WORKDIR

# Make a directory for each job to store the job output and error file
mkdir $PBS_JOBID

./myprogram 1> $PBS_JOBID/myprogram.out 2> $PBS_JOBID/myprogram.err
```
PBS_JOBNAME allows you to use the same script for multiple inputs/arguments. You can submit the same submission file with different job names (specify directly to qsub rather than hard code in the file) and then feed the job name into your program as an argument.

```bash
#PBS -l nodes=1:ppn=1
#PBS -q workshop

# Change to the directory from which you submitted this job
cd $PBS_O_WORKDIR

# Use PBS_JOBNAME as the input argument to my program
./myprogram $PBS_JOBNAME
```
PBS_NODEFILE provides a path to a file containing the list of nodes assigned to your job (one entry for each core). This is typically passed into an MPI program telling it where to run.

MPI programs run inside a PBS job will automatically detect the nodefile. However, you may want to change the nodefile and run your MPI program with some complex geometry.

```bash
#PBS -l nodes=2:ppn=8
#PBS -q workshop
#PBS -N input_800
module load intel openmpi

# Change to the directory from which you submitted this job
cd $PBS_O_WORKDIR

# Use PBS_JOBNAME (-N) as the input argument to my program
mpiexec -n 16 -machinefile $PBS_NODEFILE ./mpi_hello
```
If you have lots of small single core tasks, you have two options: Option 1: one process per PBS job. Drawback: overhead! Option 2: pack processes into a single PBS job. Drawbacks: need to be careful not to overwhelm the node at once, slightly more complicated submission script.

```bash
#PBS ...
command1 &
command2 &
...
command16 &
wait
```

& will let the command run in the background and will move onto the next command. If you do this you must have a wait at the end. Otherwise PBS will think your job has completed the instant all commands are started and exit prematurely.
Running Jobs

Tips

When scripting PBS submissions, use some `sleep` between `qsub`s. Rapid submissions may cause server to trip and drop some submissions.

```bash
$ for i in {1..10}; do qsub -N input_$i; sleep 0.5; done
```

```
#PBS -l nodes=1:ppn=1
#PBS -q workshop

# Change to the directory from which you submitted this job
cd $PBS_O_WORKDIR

# Use PBS_JOBNAME (-N) as the input argument to my program
./myprogram $PBS_JOBNAME
```
Running Jobs

Example

Real life example:

```bash
#!/bin/sh -l
#PBS -N blastx_sp
#PBS -q workshop
#PBS -l nodes=1:ppn=8
#PBS -l walltime=03:00:00

# set up modules
module purge
module load bioinfo
module load blast
module list

# move to our working directory
cd $PBS_O_WORKDIR
pwd

# log the start time
date +"%d %B %Y %H:%M:%S"
```
Real life example (continued):

```
# run blast with our inputs
blastx -query sequences.fasta -out seq.blastx_sp.fmt6 \
 -db swissprot -num_threads 8 -outfmt 6 -evalue 1E-06 \
 -max_target_seqs 5

# log completion time
echo ""
date +"%d %B %Y %H:%M:%S"
```

```
$ qsub subjob4cluster101.sub
787233.radon-adm.rcac.purdue.edu
```
Job Monitoring
Job Monitoring

$q\text{list}$ can be used to list all queues you can use and their current status and limits:

```
$ q\text{list}

<table>
<thead>
<tr>
<th>Queue</th>
<th>Current Number of Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>workq</td>
<td>352 61 211 149 336:00:00</td>
</tr>
<tr>
<td>workshop</td>
<td>8 0 0 8 4:00:00</td>
</tr>
</tbody>
</table>
```

This is a custom command unique to ITaP clusters, so this may not work on other systems.
Job Monitoring

qstat -u myusername can be used to list all current jobs from the user myusername. Typically this would be your own username and your own jobs:

```
$ qstat -u myusername

radon-adm.rcac.purdue.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time S Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>787233</td>
<td>gandino</td>
<td>workshop blastx_sp</td>
<td>--</td>
<td>1</td>
<td>8</td>
<td>--</td>
<td>03:00:00</td>
<td>Q</td>
</tr>
</tbody>
</table>
```
### Job Monitoring

Some of the `qstat` fields explained in more detail:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessID</td>
<td>Process ID of running job. Will show up shortly after job starts.</td>
</tr>
<tr>
<td>NDS</td>
<td>Number of nodes assigned to job</td>
</tr>
<tr>
<td>TSK</td>
<td>Total processors assigned to job</td>
</tr>
<tr>
<td>Req’d Memory</td>
<td>Requested memory (if specified)</td>
</tr>
<tr>
<td>Req’d Time</td>
<td>Requested maximum walltime</td>
</tr>
<tr>
<td>S</td>
<td>Status: Queued (Q), Running (R), Held (H), Complete (C)</td>
</tr>
<tr>
<td>Elap Time</td>
<td>Elapsed walltime of running job</td>
</tr>
</tbody>
</table>
qstat -a myqueue can be used to check jobs in your lab’s queue and see how busy your queue is:

```bash
$ qstat -a myqueue
...
```
checkjob is useful to see why your job hasn’t started:

```
$ checkjob 787233
job 787233
Aname: blastx_sp
State: Idle
...
BecameEligible: Mon Oct 19 17:15:04
    # If a queue is running capacity, your job will not
    # be immediately eligible for scheduling
SubmitTime: Mon Oct 19 17:13:34
...
NOTE: job req cannot run in partition radon-adm (available procs do not meet requirements : 0 of 8 procs found)
idle procs: 27  feasible procs:  0
    # if there are no nodes available to run your job you will
    # get a message similar to this. There may be free (idle)
    # processors scattered across the cluster, but may not be
    # on contiguous nodes to fit your job
```

checkjob queries the scheduler directly, so may not respond while it is busy
showstart will give you an estimate of when your job should start:

```
$ showstart 787233

job 787233 requires 8 procs for 3:00:00
*Estimated Rsv based start in 17:14:47 on Tue Oct 20 10:39:31
```

**This is a very rough estimate** and is only accurate in the current state of the cluster (assuming no more jobs will be submitted). It will not be accurate at all for low-priority queues (such as standby), as jobs there are constantly pushed back in line as owner queue jobs come are submitted.
Job Monitoring

qdel myjobid will stop and delete the specified job:

```bash
$ qdel 787233
```

Occasionally the node running your job will become unresponsive (might be why you’re trying to delete the job!) and you will receive an error message from qdel. In this case, give it a few minutes to clear on its own, and if it still doesn’t clear email us at rcac-help@purdue.edu and we will manually purge the job.
Job Monitoring

Exercise:

Change to your working space $RCAC_SCRATCH:

$ cd $RCAC_SCRATCH

Copy the lab files into your scratch directory:

$ cp -r /depot/itap/clusters101/sub_job .
$ cd sub_job
$ pwd
$ ls

Open the job script in an editor to see if you need to change it (fill in blanks):

$ nano subjob4cluster101.sub # or your favorite editor
Submit the job:

```
$ qsub subjob4cluster101.sub
```

Monitor the job’s status:

```
$ qstat -u <myusername>
$ checkjob <myjobid>
$ qstat -a
```

When job completes, take a look at results:

```
$ ls # Note presence/names of new output files
$ less jobname.oxxxxx # "xxxxx" is your job’s jobid
$ less jobname.exxxxx # "xxxxx" is your job’s jobid
```
Interactive Jobs and Thinlinc
Interactive Jobs and Thinlinc

Jobs are typically run without user interaction, but some do need interaction. An interactive job will give you an interactive session on a (dedicated to you if so requested) compute node where you can do heavy computations (vs a shared front-end).

- Test code without impacting others
- Quicker develop / test / debug cycle
- Run GUI apps on a dedicated compute node (Matlab, IGV, Fluent, etc)
You can submit an interactive job:

```
$ qsub -I
qsub: waiting for job 7867033.radon-adm.rcac.purdue.edu to start
```

The job will be queued like any other job, and may take some time to start. Eventually the job will start and you will be placed on a compute node:

```
qsub: job 7867033.radon-adm.rcac.purdue.edu ready
$ hostname
radon-e044
```
To have graphics support in an interactive job, you need to enable X11 forwarding to your local machine, and then enabled in your interactive job:

```
$ ssh -Y radon.rcac.purdue.edu
$ qsub -I -X
```

qsub: waiting for job 7867033.radon-adm.rcac.purdue.edu to start

If you don’t enable X11 forwarding you might encounter an error like this:

```
Exception in thread "main" java.awt.HeadlessException:
No X11 DISPLAY variable was set, but this program performed an
operation which requires it.
    at java.awt.GraphicsEnvironment.checkHeadless(GraphicsEnvironment.java:204)
    at java.awt.Window.<init>(Window.java:536)
    at java.awt.Frame.<init>(Frame.java:420)
    ...
```
We recommend using Thinlinc:

- Can be accessed from web browser: `thinlinc.rcac.purdue.edu` or stand-alone client
- Gives you a remote desktop environment
- Thinlinc keeps session alive after disconnect
- Faster than X11 forwarding directly to your laptop, especially if you are off-campus
Interactive Jobs and Thinlinc

Start a browser and navigate to thinlinc.rcac.purdue.edu:
Interactive Jobs and Thinlinc

Log in and wait for remote desktop to start:
Open terminal from Applications Menu and connect to cluster:
Interactive Jobs and Thinlinc

Welcome to the Radon Cluster

Radon is composed of hyperthreaded 4-core Intel Xeon E3-1284 CPUs and
10 Gbit Ethernet. Your scratch is mounted at the path specified in
the environment variable $RCAC_SCRATCH.

New mailing list for researchers interested in high performance
computing for their work.
Info: https://lists.purdue.edu/mailman/listinfo/hpc
Subscribe with email to 'hpc-subscribe@lists.purdue.edu'

Latest module updates:
texlive 20160520 was updated on Fri Feb 24 11:04
intel 17.0.1.132:lua20170308-61748-118pomm was removed on Fri Mar 3 16:03
intel 17.0.1.132 was updated on Fri Mar 3 16:33
impi 2017.1.112 was updated on Fri Mar 3 16:33

gandino@radon-fe80:~ $
Interactive Jobs and Thinlinc
Interactive Jobs and Thinlinc
But wait, this just ran on the front-end! Let’s get onto a node and try again:

```bash
$ ssh -Y radon.rcac.purdue.edu
$ echo $DISPLAY
radon-fe00.rcac.purdue.edu:11.0
$ qsub -I -X -q workshop -l nodes=1:ppn=8,walltime=00:10:00
qsub: waiting for job 945541.radon-adm.rcac.purdue.edu to start
qsub: job 945541.radon-adm.rcac.purdue.edu ready
$ echo $DISPLAY
localhost:50.0
$ hostname
radon-e004.rcac.purdue.edu
$ module load matlab
$ matlab
```
Interactive Jobs and Thinlinc

Vocabulary
Logging In
Modules
Types of Work
Good Citizenship
Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and Thinlinc
Storage
Globus
Workflows
Interactive Jobs and Thinlinc

Exercise:

Using thinlinc:

- start an interactive session, 1 node, 1 core
- load the bioinfo and fastqc modules
- open fastqc and examine the fastq files used in previous exercise subjob4clusters101.sub
Exercise: select the sequence file you want to check
Storage

Vocabulary
Logging In
Modules
Types of Work
Good Citizenship
Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and Thinlinc
Storage
Globus
Workflows
ITaP Research clusters have several types of storage available - each has its own purpose and use case.

- $\text{HOME} \text{ vs } \text{RCAC}_\text{SCRATCH}$
- Local disk (/tmp) on each individual node
- Research Data Depot
- Fortress archive
<table>
<thead>
<tr>
<th></th>
<th>$HOME</th>
<th>Scratch</th>
<th>/tmp</th>
<th>/depot</th>
<th>Fortress</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Capacity</strong></td>
<td>25 GB</td>
<td>100 TB</td>
<td>~250 GB</td>
<td>100 GB+</td>
<td>unlimited</td>
</tr>
<tr>
<td><strong>Resilient to hardware failures</strong></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td><strong>Resilient to human errors</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><strong>Subject to purging</strong></td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>medium</td>
<td>high</td>
<td>medium</td>
<td>medium</td>
<td>very slow</td>
</tr>
<tr>
<td><strong>Designed for running jobs</strong></td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>in moderation</td>
<td>no</td>
</tr>
<tr>
<td><strong>Common within cluster</strong></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes (hsi/htar)</td>
</tr>
<tr>
<td><strong>Common across clusters</strong></td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes (hsi/htar)</td>
</tr>
<tr>
<td><strong>Globus endpoint</strong></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Data Depot - a high-capacity, fast, reliable and secure data storage service.

- Basically, a ”home directory for labs”
- Redundant, fast, flexible, accessible from anywhere
- Don’t have to own nodes
- Data Depot space is mounted and accessible from all nodes and clusters

To learn more: www.rcac.purdue.edu/storage/depot/
Storage

Fortress - a large, long-term storage system

- Robotic tape library - offers vast amounts of storage for cheap
- HSI - FTP-style interface without requiring any special user knowledge.
- HTAR - a utility to aggregate a set of files into a single tar archive directly into Fortress.
- Does not support direct FTP, SFTP or SCP transfers

To learn more: [www.rcac.purdue.edu/storage/fortress/](http://www.rcac.purdue.edu/storage/fortress/)
Globus – a powerful and easy to use file transfer service. Transfer large amounts of research data to and from campus, or share data with collaborators around the country or around the globe.

Purdue Globus Portal: transfer.rcac.purdue.edu
Types of storage and when to use them: capacity, speed, longevity – pick any two

ITaP Research Computing offerings are designed around these scenarios:

- Codes, executables, scripts – develop and store in $HOME
- Lab-wide codes, executables, scripts, settings in /depot
- Produce/analyze data in $RCAC_SCRATCH
- Permanently store results in Fortress and/or in /depot

myquota command
Use variables instead of explicit paths

Snapshots are not a substitute for backups!
Some more tips:

- Avoid running jobs from $HOME
- Run demanding jobs from $RCAC_SCRATCH
- Avoid frequent I/O when possible
- Minimize simultaneous I/O from many processes
- Learn to recognize/avoid other stressors (default `ls` on big directories)
- Know when its time to learn/use parallel I/O
Globus

Vocabulary
Logging In
Modules
Types of Work
Good Citizenship
Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and ThinLinc
Storage
Globus
Workflows
Globus enables file transfers between two sites ("endpoints")

- Like hiring movers to pack your house and get it to new location
- You don’t have to be involved
- Get an update when it completes / fails
- Goes directly between the two fast sites
- Doesn’t use your laptop’s bandwidth!
- Endpoints on all of our clusters, many other Universities/labs and even on your own laptop!
- Sites must set up Globus on their side, so not everyone has it but worth checking!
Globus

Exercise:

- Log into Globus:

  - transfer.rcac.purdue.edu
Log in to use Purdue Globus Web App

Use your existing organizational login

e.g. university, national lab, facility, project, Google or Globus ID

(Your Globus username and password used prior to February 13, 2016 is now Globus ID)

Purdue University Main Campus

Continue

Globus uses CIlogon to enable you to Log In from this organization. By clicking Continue, you agree to the CIlogon privacy policy and you agree to share your username, email address, and affiliation with CIlogon and Globus. You also agree for CIlogon to issue a certificate that allows Globus to act on your behalf.

Didn't find your organization? Then use Globus ID to sign up.
Globus

Vocabulary
Logging In
Modules
Types of Work
Good Citizenship
Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and Thinlinc
Storage
Globus
Workflows
Globus

Vocabulary
Logging In
Modules
Types of Work
Good Citizenship
Jobs, Queues, and PBS
Running Jobs
Job Monitoring
Interactive Jobs and Thinlinc
Storage
Globus
Workflows

Transfer Files

Transfer Files | Activity | Endpoints | Bookmarks | Publish | Console

Endpoint: Purdue Research Computing - Home Dire
Path: /-

Endpoint: Purdue Radon Cluster
Path: /scratch/radon/

select all  up one folder  refresh list

Folder
unix101-2016
Can share select folders with outside collaborators: no need to have Purdue ID.

Can set up an endpoint on your personal laptop:
www.globus.org/globus-connect-personal
Workflows
Unless you are running a one-off simulation, you should stop and think about the workflow:

- What’s the optimal resource request?
- Experimentation - scaling studies
- Do I break my workflow into multiple jobs?
- How does my data flow?
  - From original source
  - Working spaces
  - Post-processing/analysis
  - Archives
Sample workflow:

- Grab dataset before batch of jobs:
  \[\text{rsync -rav /depot/group/data/experiment1/scratch/radon/u/user/}\]

- Submit batch of jobs

- Save to /scratch/radon/u/user/results

- Copy results back to Depot or Fortress:
  \[\text{ssh user@cli.globusonline.org transfer -- purdue#radon:/scratch/radon/u/user/results purdue#depot:/depot/group/data/experiment1-results}\]

- Requires ssh key setup with Globus

- Happens in the background; doesn't waste job time
This requires SSH keys to script unattended Globus transfers:

- Create GlobusID
- Go to globus.org and create Globus ID
- Log in with it and link your Purdue Credentials
- Get SSH key and pair it with your Globus ID
Workflows

Can use existing key:

```
$ cd ~
$ ls -las .ssh
$ cat .ssh/id_rsa.pub
```

Or generate new key:

```
$ ssh-keygen -t rsa -f .ssh/id_globus -N ""
$ cat .ssh/id_globus.pub
```

Copy the public key.
Manage SSH and X.509 Keys

You have the following key attached to your Globus ID account:

SSH Public Key

expand delete rename
Workflows

Add a New Key

Alias: My Laptop's Key

Type:
- SSH Public Key
  Allows you to use the Globus command line interface via any standard ssh client. Some Globus-based applications may also use this SSH authorized key to provide access to their resources.

- X.509 Credential
  Allows you to use the Globus command line interface via a gssish client, and to activate endpoints with a delegated proxy of this X.509 credential via gssish to the CLI.

Body

Add Key  Cancel