HPC3 - Goals

1. Enables users to have access to a larger compute/analysis system than they could reasonably afford “on their own”

2. Enables access to specialized nodes (large memory, 64bit GPU)

3. Fosters a growing community across UCI to utilize scalable computing (HPC and HTC)* for their scientific research program and teaching

4. Provides a well-managed software environment that forms the basis of a reproducible and more secure research environment

* HPC – High-Performance Computing
  HTC – High-Throughput Computing
What does HPC3 look like?

- HPC3 is 100s of servers, 1000s of compute cores
  - 9104 x86 Cores
  - 207 servers
  - 4 different brands of hardware
  - Grows every year

- And Many Petabytes (1 PB = 1000 TB = 1,000,000 GB) of Storage
  - Seven Parallel File Systems (BeeGFS)
  - One home area file system
  - ~900 hard drives
  - Every node has two local drives

This is “Sea” of physical resources.
Interconnected by high-speed networking.
Current Specs for a new individual node in HPC3

- 2 X Intel Xeon Gold 6336Y Processor @ 2.4GHz, 24 Cores/Processor, 48 Cores/node
- 256GB Memory
- 480 GB SSD – System Drive
- 1.92TB NVMe – fast local scratch
- 10 Gigabit/s Ethernet
- 100 Gigabit/s Infiniband

There are various ages of nodes in HPC3 with both Intel and AMD Processors. Some nodes have 40 cores, some 64.

Most cores in HPC3 are performance similar and operate at about 2.4GHz
When you log in to hpc3, you will land on one of three login nodes.

% ssh hpc3.rcic.uci.edu

No HPC3 node is directly-connected to a public network

All of your computation runs on one of these nodes. NOT a login node.
Example Task: Bowtie2

- **The application**: Bowtie2: fast aligner of sequence reads to reference genomes
- **Input data**: reference genome, read(s) to align
- **Output data**: where to store results
- **Resources required**: amount of memory and # cores

1. You need to understand everything your application requires (data, resources, parameters)
2. You need to describe to HPC3 your request == Job.
You have program to run ... what’s next?

• Policies (and real-time enforcement) prevents you from running significant computation on a login node
• If you cannot run on a login node, then
  • Somehow, you need to acquire appropriate resources that can run your application to completion.
• Job: application, data needed for the application, how to call your application, and a resource request
• Scheduler (Slurm) – takes job requests, allocates resources, and runs the job.
• ➔ You submit your job to the scheduler.
• Sometime later, your job should run somewhere on the cluster and then you can process the results.

Unlike running on your desktop, most jobs in HPC3 are not interactive == batch job.
(you can run interactive jobs on HPC3, but do so only when really required)
Things that you can request in a resource allocation

- How many **computing cores** your application needs (and can effectively use)
- How much **memory** (RAM) your application needs to run
- How much **time** (by a wall clock) your application need to complete its work
- If it requires a **GPU**
- **Special features** – particular CPU instructions (e.g. AVX-512), large local scratch space (NVMe)
The function of a job scheduler (Slurm)

- Considers all resource requests, attempts to place each one (run the job) somewhere in the cluster
  1. Slurm won’t oversubscribe the physical resources of a node
     - Example: Suppose a 40-core, 180GB memory node
       - Already running 30 jobs each requesting 5GB and a single core (150GB/180GB allocated, 30/40 compute cores allocated)
       - Supposes a new job is requesting 8 cores and 32GB of memory
         - New job doesn’t “fit”, there are enough cores, but not enough available memory
  2. Slurm takes into account priority.
     - Example: jobs that have been waiting a while to run generally are of higher priority than just-submitted jobs
- On average – the HPC3 scheduler must consider a new job every 4 seconds.
A few statistics for HPC3

- Over the past year HPC3 delivered
  - 8,441,268 jobs (16 jobs per minute)
  - 47,551,935 core hours
    - ~5428 core-years
    - ~135 servers running full out for a year.
- 807 Distinct Users
- Largest single job request: 1280 cores = 32 servers
- Average job size ~ 5 cores/job

Usage of HPC3 varies month-to-month
Queueing in HPC vs. HPC3

HPC
Private queues
- abio
- adriana
- air
- braincircuits
- braincircuits2
- free queues
- pub

"Which hardware is mine?"

HPC3
Common queues (partitions)
- debug
- standard
- gpu
- free

"What kind of resource does my job need?"

80+ queues owner preemption

-5 queues free queue preemption

free queues owner preemption

high memory

* retired cluster – had private nodes and queues, no accounting
HPC3 principles: No oversubscription + Fair queueing

No oversubscription
If you own X% of the total cluster, your starting account balance is ~ X%
of the total number of hours that can be delivered in a year by the entire cluster.

Fair Queueing
Non-FIFO. Jobs arriving earlier in the queue are not guaranteed to schedule first.
Want to prevent large number of jobs from user A blocking a small number of jobs from user B

Bias towards interactive turnaround for small debugging jobs
Optimize people time for the debug process.
Small core count + short time duration jobs should schedule as quickly as possible

Fair running
If you are running an accounted job, once your job is started, it will not be pre-empted/killed

Free cycles
Users who pick up spare cycles == run free jobs can have their jobs killed
so that accounted jobs can run as soon as possible

UCI Research Cyberinfrastructure Center
Accounting and limit keep chaos at bay …

- There are limits on just about everything
- The most obvious limit is “how big is your slurm bank account?”
  - For allocated jobs, the account is charged:
    - 1 unit per hour for each core in use. (e.g. a 4-core job, consumes 4 units every hour of running)
    - 32 units per hour for each GPU in use
- Per-core memory limits **default is 3GB/core**
- Job time limit (up to 14 days, default is two days)
- Core limits – if you ask for one core, you get one core. Even if multi-threaded (enforced by the kernel using cgroups).
HPC3 accounting: jobs draw from an accounting bank

**Accounted jobs vs. free jobs**
- Accounted – once a job is started, it cannot be killed or pre-empted
- Free – a free (non-accounted) *can be killed at anytime*

**Three ways of filling your account**
- **Granted cycles**: UCI core funds purchase hardware to provide enough resource to support granted cycles
- **Converted**: condo-style, researchers purchase hardware, RCIC manages nodes capability
  is converted to core-hours. Formula:
  Physical hardware can deliver $N$-core-hours/year. $0.95N$ are deposited into an owner's account each year the owner has a node (or nodes) in the cluster.
- **Purchased**: Hours are pre-purchased (~$0.01/core-hour) in chunks (e.g., $100$ increments buys ~ $10000$ core-hours)

Umm... I need to Run a specific code
See the software map on [https://rcic.uci.edu/hpc3/software-tutorial.html](https://rcic.uci.edu/hpc3/software-tutorial.html)

**Software Map**

- **Use modules:**
  - biotools-admix
    - features:
      - bowtie2/2.5.4.1
      - java/1.8.0
      - ncbi-blast/2.10.0
      - ncbi-nmp/2.10.2
      - perl/5.30.0
      - pilon/1.23
      - protoblast/2.11.4
      - python/2.7.17
      - R/3.6.2
      - ração/1.4.13
      - samtools/1.10
      - SPAdes/3.14.0
      - trf/4.0.9

- **Provides:**
  - biotools-admix
    - bandage/0.8.1
    - bcftools/1.10.2
    - bedtools/2.29.2
    - bowtie2/2.2.3
    - bowtie2/2.4.1
    - bgztools/0.6.0
    - bwa/0.7.17
    - fastqc/0.11.9
    - cufflinks/2.2.1
    - edirct/1.0
    - fastp/0.20.0
    - fastqc/0.11.9
How to request software

- Some software is not possible to move when performing major OS upgrades*
  - Too old for the new OS
  - Versions will be different because of dependencies, new OS.
- Some packages you will need to install yourself
  - R (we have ~400 R modules pre-installed from CRAN)
  - Perl (we have ~200 perl modules pre-installed from CPAN)
  - Python (we have ~60 python modules pre-installed from PyPi)
  - Conda (we provide bioconda for PacBio tools and anaconda for python3)
- See https://rcic.uci.edu/hpc3/getting-help.html#askforsoftware guide
  - Request what you really need – RCIC doesn’t have the manpower to install software that might be of use
  - Experiment on installing your own with R/Python/Perl/Conda
  - Required elements of software request

*HPC3 is currently CentOS7 – August 2022 → Enterprise Linux 8
Basics of being a good citizen on a cluster

1. **Cluster is a shared resource, it is NOT your personal machine**
2. **What you do affects all the other users, so think before you hit that Enter key**
   - Do not run interactive jobs on login nodes
   - Do not transfer data on login nodes
3. **Secured from mischief and disasters.**
   - We restrict users’ ability (permissions) to install and run unwanted software applications
   - Be careful when bringing applications from unknown sources. **DO NOT ask for sudo (elevated privileges) access**
4. **For your jobs: use the resources you need, don’t ask for more**
5. **Learn how to submit a trouble ticket to speed up problem resolution**
So, you need to run your application

1. There are many hundreds of domain-specific software applications already installed on HPC3 – See if the one you want is there.

2. YOU have to be the expert on how to run YOUR application
   a) What input data is needed
   b) What parameters (flags, arguments) are given to call the program
   c) What resources does it need (memory, cores, disk)

Step 1 – See if the software you need is already available on HPC3
What are environment modules

- **Environment module**: a file (modulefile), understood by the Modules package, to dynamically modify user's environment
- Each **modulefile** contains all the info needed to configure the shell to use a specific application.
- Command **module load** interprets the modulefiles and
  - Sets aliases
  - Sets environment variables
  - Loads depended modules
- Command **module avail** lists all installed software and their versions

## Module Command

```
pspadm@login-115-1$ module avail
```

```
------------------------------------------- /usr/share/Modules/modulefiles -------------------------------------------
dot module-git module-info modules null use.own
```

```
------------------------------------------- /etc/modulefiles -------------------------------------------
mpi/openmpi-8.6.64
```

```
------------------------------------------- /opt/rcic/Modules/modulefiles/AL LEARNING -------------------------------------------
pytorch/1.5.1 tensorflow/2.0.0 tensorRT/4.0.1.5
```

```
------------------------------------------- /opt/rcic/Modules/modulefiles/BIOTOOLS -------------------------------------------
```

```
------------------------------------------- /opt/rcic/Modules/modulefiles/Chemistry -------------------------------------------
amber/19.11/gcc.8.4.0
chimera/1.18
gromacs/2020.4/gcc.8.4.0
```
How to search for modules/programs on HPC3?

To access a program, you need to have its directory in your $PATH

```
$hpc3-14-00 2046% echo $PATH
/data/utils/system-files/system-wide-env-setup:/opt/rocks/yaml2rpm:
/opt/apps/python/3.8.0/bin:/data/homezvol0/n/perl5/bin:/usr/local/bin:
/usr/bin/usr/local/sbin/usr/sbin/data/homezvol0/n/bin
```

What is my $PATH?

```
$hpc3-14-00 2047% which pip
/opt/apps/python/3.8.0/bin/pip
```

What pip am I using?

**Language modules** are collections of related variables, functions and subroutines that perform a set of specific programming tasks. Simply put—files consisting Perl/Python/R code. **Search using its language methods.**

To access a language module, you need to use its language: **Perl / Python / R**

To access a language **Perl / Python / R** you need ... **environment modules**
Environment module commands summary

- **$ module avail**
  - shows all installed software environment modules

- **$ module avail R**
  - show R modules

- **$ module keyword -i salmon**
  - check all modules for a keyword (-i is case insensitive)

  ```
  salmon/1.1.0 : Name salmon
  salmon_1.1.0
  ```

- **$ module display R**
  - shows environment modification + description

- **$ module help R**
  - show module specific help (description)

- **$ module load R**
  - loads R at whatever latest version **not ideal**

- **$ module load R/4.0.2**
  - loads R at specified version **preferred method**

- **$ module list**
  - lists currently loaded modules

- **$ module unload R/4.0.2**
  - unloads specified module **(in reverse order if many)**

- **$ module purge**
  - removes all loaded modules
What Perl modules are installed?

Method 1: instmodsh
hpc3-14-00 2001% module load perl/5.30.0
hpc3-14-00 2002% instmodsh
Available commands are:
    i  - List all installed modules
    m <module> - Select a module
    q  - Quit the program

    cmd? i
Installed modules are:
    Algorithm::Diff
    Alien::Build

Method 2: cpan
hpc3-14-00 2006% cpan
Terminal does not support AddHistory.
To fix enter> install Term::ReadLine::Perl
    cpan shell -- CPAN exploration ...(v2.22)
Enter 'h' for help.

    cpan[1]> r
Fetching with LWP:
    ...
DONE
Writing /data/homezvol0/npw/local/share/.cpan/Metadata

Package namespace installed latest in CPAN file
    Alien::Build  2.15  2.32  Alien-Build-2.32.tar.gz
    Alien::Libxml2  0.14  0.16  Alien-Libxml2-0.16.tar.gz

Method 3: perl test script
    use strict;
    use warnings;

    use Unicode::Map;
    use Bio::Perl;
    say STDERR "No errors";

    hpc3-14-00 2007% module load perl/5.30.0
    hpc3-14-00 2008% perl test.pl
What python modules are installed?

Method 1: pip

```
hpc3-14-00 2022% module load python/3.8.0
hpc3-14-00 2023% pip list
Package         Version
----------------- --------
absl-py         0.9.0    
appdirs         1.4.4    
astor           0.8.1    
backports.weakref 1.0.post1
...```

Method 2: quick test for one module

```
hpc3-14-00 2027% python -c "import mmtf"
hpc3-14-00 2028% python -c "import bla"
Traceback (most recent call last):
  File "<string>" , line 1, in <module>
ModuleNotFoundError: No module named 'bla'```
What R modules are installed?

hpc3-14-00 2022% module load R/4.0.2
hpc3-14-00 2022% R

> installed.packages()
... 
xlsxjars        "4.0.2"
XML             "4.0.2"
xml2            "4.0.2"
xopen           "4.0.2"
...

> find.package("XML")
[1] "/opt/apps/R/4.0.2/lib64/R/library/XML"

> find.package("XML2")
Error in find.package("XML2") : there is no package called ‘XML2’
How to install language modules

Many language modules can be installed in user space

There is no single repository for download for all, use main ones to start with

Perl  https://www.cpan.org
Python  https://pypi.org
R  https://cran.r-project.org

How to install your desired language module, see section 3.1 Install it yourself in Getting Help user guide  https://rcic.uci.edu/hpc3/getting-help.html

<table>
<thead>
<tr>
<th>R packages</th>
<th>Python packages with pip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Conda local environments</td>
<td>Perl CPAN modules</td>
</tr>
</tbody>
</table>
So, You want to run a program - Batch Jobs

1. How to run jobs on HPC3?
2. Batch Script
3. Batch Submission
4. What are the common problems?
# The Batch Script – A Unix Shell Script with COMMENTS that Slurm understands

#!/bin/bash
# Account to charge
#SBATCH -A ppapadop_lab
# Queue
#SBATCH -p standard # partition
(queue)
# Job Parameters
#SBATCH -N 1 # number of nodes
#SBATCH -n 1 # number of cores
#SBATCH -t 00:00 # time (D-HH:MM)
#SBATCH -o simple.%j.out # stdout
#SBATCH -e simple.%j.err # stderr
hostname
set
module avail

#SBATCH == parameter to slurm. Try man sbatch to see all available parameters
Submit the Batch Script as a Job

[ppapadop@login-i15:-]$: sbatch simple_job.sh
Submitted batch job 13538398
[ppapadop@login-i15:-]$: squeue -u ppapadop

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ACCOUNT_ST</th>
<th>TIME</th>
<th>CPUS</th>
<th>NODE</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13538398</td>
<td>standard</td>
<td>simple_j</td>
<td>ppapadop</td>
<td>ppapadop_1</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1 (None)</td>
</tr>
</tbody>
</table>

[ppapadop@login-i15:-]$: qstat -u ppapadop

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ACCOUNT_ST</th>
<th>TIME</th>
<th>CPUS</th>
<th>NODE</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[ppapadop@login-i15:-]$: head simple.13538398.out

bash-5.0$
Get Some info about a completed job

[ppapadop@login-il5:~]$ **seff** 13538398
Job ID: 13538398
Cluster: hpc3
User/Group: ppapadop/ppapadop
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:00:00
CPU Efficiency: 0.00% of 00:00:00 core-walltime
Job Wall-clock time: 00:00:00
Memory Utilized: 0.00 MB (estimated maximum)
Memory Efficiency: 0.00% of 2.93 GB (2.93 GB/core)

**seff** – after a job has completed, it can tell you how much resource was utilized
Let's look at a real job

[ppapadop@login-il15:~]$ seff 13201239
Job ID: 13201239
Cluster: hpc3
User/Group: xxxxxxxx/xxxxxxxx
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 20
CPU Utilized: 00:20:37
CPU Efficiency: 3.17% of 10:50:20 core-walltime
Job Wall-clock time: 00:32:31
Memory Utilized: 27.88 GB
Memory Efficiency: 7.74% of 360.00 GB

- This is a very inefficient request.
- This information should be used to improve the request the next time this code is run

Requested 20 cores
Ran 20 cores for 32 minutes ~ 11 hrs of cpu time
Used ~28GB of RAM
Requested 18GB/core
Queues: Not all jobs require the same resources

- Memory/core is the big differentiator
- HPC3 is mostly "standard" memory nodes.
- There is only six hugenmem nodes and one maxmem node
- Access to higher memory nodes is by request
My Job isn’t running. Why?

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>USER</th>
<th>ACCOUNT</th>
<th>ST</th>
<th>TIME</th>
<th>CFUS</th>
<th>NODE</th>
<th>NODELIST (REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12899293</td>
<td>free</td>
<td>tgokey</td>
<td>tgokey</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>ReqNodeNotAvail,</td>
</tr>
<tr>
<td>12815566</td>
<td>free</td>
<td>tgokey</td>
<td>tgokey</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>Resources</td>
</tr>
<tr>
<td>13057774</td>
<td>free</td>
<td>tgokey</td>
<td>tgokey</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>ReqNodeNotAvail,</td>
</tr>
<tr>
<td>13162908</td>
<td>free</td>
<td>tgokey</td>
<td>tgokey</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>ReqNodeNotAvail,</td>
</tr>
<tr>
<td>13233401</td>
<td>free</td>
<td>tgokey</td>
<td>tgokey</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>ReqNodeNotAvail,</td>
</tr>
<tr>
<td>13525549 [1-3]</td>
<td>standard</td>
<td>mingonc</td>
<td>jranders_1</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>AssocGrpCPUMinutesLimit</td>
</tr>
<tr>
<td>13525555 [1-3]</td>
<td>standard</td>
<td>mingonc</td>
<td>jranders_1</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>AssocGrpCPUMinutesLimit</td>
</tr>
<tr>
<td>13525556 [1-3]</td>
<td>standard</td>
<td>mingonc</td>
<td>jranders_1</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>1</td>
<td>AssocGrpCPUMinutesLimit</td>
</tr>
</tbody>
</table>

- AssocGrpCPUMinutesLimit == Not enough units in Slurm Account
- ReqNodeNotAvail == job is asking for a specific node (Unusual)
- Resources – The configuration of resources requested is not currently available
- There are other reasons, too
I really need an interactive job

- Request immediate job (if resources are available you will be scheduled quickly)
- `srun --pty /bin/bash -i`

```
[ppapadop@login-i15:~]$ srun --pty -p free -c 4 /bin/bash -i
srun: job 13538485 queued and waiting for resources
srun: job 13538485 has been allocated resources
[ppapadop@hpc3-118-05:~]$`

- Run above asks for 4 cores and to run in the free partition
Common Situations

1. Submitting 1000s of nearly identical jobs
   Fix: convert to an array job  Crucial for many users!
2. Run heavy computational jobs on login nodes
   Fix: claim an interactive node or submit a batch job
3. Does not set job environment properly / use too much resources
   Fix: use modules
       ask resources you need and not more
4. Running out of space and not checking the disk quotas
   Fix: watch your usage and do periodic cleaning
5. Not testing your jobs submissions.
   Fix: test on a small input first
       check all names and variables are correct
       after submitting a job check the status
6. Not providing correct information when submitting a ticket for help
   Fix: read the User Guides and follow the directions

See User Guides
http://rcic.uci.edu
The dreaded “OOM killer”

- OOM = Out of Memory
- Why does it happen?
  1. Your application is requesting more RAM than has been allocated
  2. The operating system responds by KILLING your process
  3. Slurm Reports this as a non-zero exit status (code 125)
- How frequently does this happen?
  - Over the past 25 days:
    - 354395 jobs ran
    - 1974 were killed for out of memory (~0.6% of jobs)
Array Jobs

- What’s an array job? A job where you want to run nearly identical versions of the same application, but only one or two parameters change
  - Align a bunch of different reads to the same reference
  - Vary a single parameter searching for a particular result (parameter sweep)
- Why is it good?
  - Reduces workload on slurm – instead of 1000 nearly identical jobs, 1 job with 1000 elements
An array job in SLURM

```bash
#!/bin/bash
#SBATCH --job-name=MyArray
#SBATCH -p free
#SBATCH -t 1-2000%100         ## number of tasks to send and concurrency
#SBATCH -e %x.e%A_%a         ## %x - job name, %A - job id, %a - task id
#SBATCH -o %x.o%A_%a

module load myprog/1.2.3

INPUT=/pub/panteater/Test/Week1/Input.txt
OUTPUT=/pub/panteater/Test/Week1/Outputs

Args=$(awk "NR==$SLURM_ARRAY_TASK_ID" $INPUT)  # get arguments from file
myprog $Args -o $OUTPUT/out.$SLURM_JOB_ID-$SLURM_ARRAY_TASK_ID  # separate task output
```

- 2000 jobs (array indices 1-2000)
- Only run 100 at a time (usually to reduce file system load)
- Separate standard out and err for each array job.
- `SLURM_ARRAY_TASK_ID` is set by Slurm when an array element is running. Use this to determine what to do.
## Environment variables

<table>
<thead>
<tr>
<th></th>
<th>Slurm EnvVar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job ID</td>
<td>$$\text{SLURM_JOBID}$$</td>
</tr>
<tr>
<td>Job name</td>
<td>$$\text{SLURM_JOB_NAME}$$</td>
</tr>
<tr>
<td>Submit directory</td>
<td>$$\text{SLURM_SUBMIT_DIR}$$</td>
</tr>
<tr>
<td>Submit host</td>
<td>$$\text{SLURM_SUBMIT_HOST}$$</td>
</tr>
<tr>
<td>Node list</td>
<td>$$\text{SLURM_JOB_NODELIST}$$</td>
</tr>
<tr>
<td>Job array index</td>
<td>$$\text{SLURM_ARRAY_TASK_ID}$$</td>
</tr>
</tbody>
</table>
## Job specification

<table>
<thead>
<tr>
<th></th>
<th>Slurm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script directive</td>
<td><code>#SBATCH</code></td>
</tr>
<tr>
<td>Queue</td>
<td><code>-p partitionName</code></td>
</tr>
<tr>
<td>Node count</td>
<td><code>-N [min[-max]]</code></td>
</tr>
<tr>
<td>CPU count</td>
<td><code>-n count</code></td>
</tr>
<tr>
<td>Wall clock limit</td>
<td><code>-t [min] or -t [days-hh:mm:ss]</code></td>
</tr>
<tr>
<td>Standard output file</td>
<td><code>-o filename</code></td>
</tr>
<tr>
<td>Standard error file</td>
<td><code>-e filename</code></td>
</tr>
<tr>
<td>Combine stdout/error</td>
<td><code>-o filename (without -e)</code></td>
</tr>
<tr>
<td>Copy Environment</td>
<td>`--export=[ALL</td>
</tr>
<tr>
<td></td>
<td>Slurm</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Event Notification</td>
<td><code>--mail-type=[events] use sparingly!</code></td>
</tr>
<tr>
<td>Email Address</td>
<td><code>--mail-user=address</code></td>
</tr>
<tr>
<td>Job Name</td>
<td><code>--job-name=name</code></td>
</tr>
<tr>
<td>Job Restart</td>
<td><code>--requeue OR --no-requeue</code></td>
</tr>
<tr>
<td>Work Directory</td>
<td><code>--workdir=dirname</code></td>
</tr>
<tr>
<td>Resource Sharing</td>
<td><code>--exclusive OR --shared</code></td>
</tr>
<tr>
<td>Memory Size</td>
<td>`--mem=[mem][M</td>
</tr>
<tr>
<td></td>
<td>`--mem-per-cpu=[mem][M</td>
</tr>
<tr>
<td>Account to charge</td>
<td><code>--account=account</code></td>
</tr>
</tbody>
</table>
Where to get more help and information

http://rcic.uci.edu
RCIC receives several support requests per day

- Many are routine (account creation)
- Sometimes things ARE really wrong
- Please look at our web site to see if your problem might be addressed there.
Making request to RCIC – we’re not mind readers!

1. I am submitting my job with my job script and I think there is something missing in my script and I am unable to find it. Can you look at it?

2. When I am running the jobs from the model there is an issue of 'libnetcdf' and I am unable to fix this as well. The path where I am running the job is '/dfs3/pub/userX/PROGY/test2'. More details are there in the screenshot below.
   1. What is the submit script?  2. Screenshot has no info on the cause of error

3. I am unable to access HPC3. My connection gets closed on login. Please refer to the image below.
   1. Where were logging in?  2. VPN?  3. Do you even have an account on HPC3?

4. I need to run a program ThisGreatProgram. Can you install it please. It’s commonly used in bioinformatics field so maybe it’s better to install it as a public module. The instructions are sudo apt-get ...
   1. Missing URL & version  2. How big is group who will be using it?

5. I’m having trouble to write any file in the directory I usually work in /wt/panteater/. Do you know why?
   1. On what node?  2. What was the command?  3. What was the error?
https://rcic.uci.edu

- Covers many, but not all things.