Contents

1 Getting Started 3
  1.1 Getting Started ................................................................. 3

2 Servers 9
  2.1 Compute Servers ............................................................... 9
  2.2 Linux Clusters ................................................................. 10
  2.3 Other Servers ................................................................. 12

3 Software 13
  3.1 Unix Software ................................................................. 13
  3.2 Research Software ........................................................... 33

4 arc-docs 41
  4.1 How to Edit ................................................................. 41
PSURC Docs is a collection of documents, notes, instructions and tutorials that help explain resources available to students and faculty provided by PSU Research Computing through PSU’s OIT.

**Warning:** First time here? Please read the *Getting Started* guide!
1.1 Getting Started

1.1.1 Server Resources

There are two kinds of computing services for running jobs that ARC provides, Compute Servers and Clusters.

Compute Servers

When you make a research account you will have access to our compute servers Circe and Hecate by default. These servers are large computers running linux that allow multiple users to run jobs simultaneously. Once you are connected to a compute server, you can run jobs in the command line like you would on a normal linux computer, just be mindful of other users and their jobs.

Compute Clusters

A compute cluster is a set of networked computers that act as one. Compute clusters provide much more power for parallel jobs than the compute servers Circe and Hecate. We have two clusters here at ARC, Hydra and Gravel. If you want access to the compute clusters you will have to request access in addition to your regular research account. To request access to ARC’s compute clusters please contact: consultants@pdx.edu

Additional Information About Our Servers

Knowing your software

The best thing you can do to learn how to use new and complicated Unix software packages is get it running on your local machine. Once you have that experience, getting it running in a shared computing environment will make a lot more sense.

All of ARC’s Servers run CentOS Linux, a flavor of Unix.

Some software specific support can be provided by ARC students and employees if time is available.

Since all software is different the best we can offer on this are tool suggestions to help with local testing.

Here is a good strategy to learn unfamiliar software:

1. Install it locally on your desktop or in a virtual machine
2. Install it to your home directory, or use the system wide install of the program on the computation servers and see if it works the way you would expect.

3. Install your program to your home directory on cluster and try launching it through the scheduler on a compute node.

**Setting up a testing environment locally** There isn’t one way to set up a local test environment, but here are a few tools to help get you started.

**Testing on Linux** If you run Linux locally, usually you can simply install the dev tools and start building and testing software from source.

**Testing on OS X** OS X has the benefit of running a flavor of Unix and offers a package manager that has a large catalog of research software. Install Xcode using the App Store and set up homebrew and the homebrew-science tap.

**Testing on Windows** Windows is not a Unix operating system and therefore will present a challenge to setting up Unix software locally. You are better off using a Virtual Machine and testing with that. You can try to use something like Cygwin if you wish.

**Testing Using a VM** This option is available to all operating systems. Using a tool like vagrant along side virtualbox, you can quickly install a Centos 6.5 image to test in:

- CentOS 6.5 Vagrant Manifest

**Installing to your home directory** Once your software is installed and working locally, the next step is to install it to your home directory on the computational servers (if its not already installed system wide). You can use stow to manage these manual builds in your home directory.

**Picking A Server**

Once you have an idea of the software you wish to run, and its hardware requirements, you can pick a server to run your compute jobs on.

**Circe circe.rc.pdx.edu**

*Circe* is a general compute server for smaller jobs. All minor computing tasks should take place here. It also offers a GPGPU for hardware accelerated computations.

- Good for jobs that require less than 10GB of RAM.
- Can utilize GPGPU(CUDA) parallelization.
- No job scheduler.

**See also:**

Read more about how the *Nvidia Tesla K20 GPGPU* card can help speed up your computations.
**Hecate**  hecate.rc.pdx.edu

*Hecate* is the other general purpose compute server for RAM hungry jobs. Jobs that require over 10 GB of ram should have priority over jobs that are not ram intensive. If you are running a CPU or IO bound job that does not require lots of RAM, you should move your job to Circe or a node in the clusters.

Hecate is good for jobs that:
- Requires large (over 10 GB) of RAM.
- No Scheduler

**Hydra Cluster**  hydra.rc.pdx.edu

The *Hydra Cluster* is the primary compute cluster at ARC. It is available for massively distributed compute jobs, or just running CPU/IO intensive tasks on single nodes if they are disruptive to other tasks on the general compute servers. Programs are required to use the slurm scheduler which requires a bit of extra training to get started on.

Hydra is good for jobs that:
- Use message passing (MPI, MPIC etc)
- Tend to have a high CPU load and disrupt other processes on the general compute servers and require their own dedicated node.
- Can be broke into many little parts and executed in parallel across multiple nodes.
- Are easily disrupted by other processes on the general compute servers and require a dedicated node.
- Have users wishing to run their programs using a Job scheduler.

**Gravel Cluster**  gravel.rc.pdx.edu

The gravel_cluster is the secondary compute cluster at ARC. It runs slightly older hardware, and generally is less busy than hydra. Students are free to use gravel if they wish but changes to hydra are tested on gravel first.

Gravel is good for jobs that:
- Might be disruptive to jobs running on Hydra
- Won’t fit on hydra if hydra is busy
- Want to try something new without affecting performance of jobs on hydra

### 1.1.2 Connecting to Our Servers

`ssh` is used to connect to our servers. `ssh` is included on Mac OSX and Linux and can used via the command line. If you are on Windows you will need additional software, most commonly PuTTY, to access our servers over `ssh`.

**Connecting From Off Campus**

If you are off the campus network you will have to use PSU's `vpn` client OR `ssh` onto hera.rc.pdx.edu and then `ssh` onto the server you want to access from there.
Additional Information About Connecting

Connecting to a Server

The research systems are accessed primarily through use of a command-line interface utilizing a secure shell (SSH). For information on how to use this go here:

- Using SSH

OSX/Linux/Unix  Use your system provided terminal program to connect to ARC research servers using SSH.

Windows  Use putty to connect to ARC research servers.

Off Campus?  The research servers can only be connected to on campus. If you wish to connect to the research servers off campus, you have two options:

- PSUs VPN
- Connecting to Hera (rc.pdx.edu) then connecting to the research server of your choice from that connection.

1.1.3 Storage

There are several options for storage on our systems.

Home Directories

Your home directory (~) is your own personal default storage space. When you connect to a server via ssh you are in your home directory. Home directory space is very limited. Do not store large files here–only configuration files, documents, and locally built programs. The compute servers share home directories.

Scratch Drives

Each server has its own Scratch Drive or scratch space. Scratch Drives are fast storage devices that you can use to store the data you want to run your jobs on. Scratch Drives are only intended to be used as short-term storage so please move your data off of scratch when you are done with your analysis.

If you are going to use a scratch drive please make a directory named after yourself to keep all of your files in.

Scratch drive locations

Compute Servers (Circe/Hecate):

1. /disk/scratch

Hydra (compute cluster):

1. /scratch
2. /scratch2

Note: Scratch Space is not shared between the servers. e.g: scratch on Hecate is separate from scratch on Circe
Research Shares (I-Drive)

Research Shares are shared folders that you need to request access to. To request access to a share OR request a separate research share for your project please go to: http://intranet.pdx.edu/oit/i-drive-request

Research shares can be found in: /vol/share on all systems (excluding gravel)

1.1.4 Other Tutorials on Getting Started

Tuning your Software

The optimal number of threads to use is entirely dependent on the program being run and may take a bit of trial and error. More threads does not usually result in faster runs and can actually be detrimental.

**Warning:** Please limit the number of threads to half of the currently available on the compute servers unless you have demonstrable evidence of increased performance with more threads.

Start small and increase the concurrency of your program. Benchmarking will help you discover the optimal settings for your program.

**Warning:** More threads does NOT equal faster performance. Play with the concurrency and thread settings and benchmark to find the sweet spot.

Moving Data

Getting data on on and off the servers can be done using SFTP or rsync.

Rsync Between Servers

If you need to switch which server you are running on, the fastest safe way to move your files is with rsync which can synchronize folders reliably.

**Moving a folder from hecate to circe**  Log into circe via ssh:

```
$ ssh user@circe.rc.pdx.edu
```

Then tell rsync which files you want to move:

```
$ rsync -v -a -e ssh user@hecate.rc.pdx.edu:/disk/scratch/FOLDER.TO.MOVE /disk/scratch/
```

This will clone the folder from hecate onto circe. NOTE: The files remain on the original server so please clean up those to preserve disk space.

Using SFTP and a GUI

Sometimes its easiest to just use a GUI to move files onto the research servers. Here are some decent options.

Windows

- WinSCP
OS X

- Cyber Duck
- Transmit (Paid) - Local portland product!

Nice your processes

It is important to nice intensive processes so that they don’t interfere with using the most basic functions of the research server such as changing directories or moving files.

The nice and renice commands adjust the priority of a process. ARC recommends that all users nice all of their processes with a value of at least 5. To nice a process when it is started type:

```
$ nice +5 [other_commands]
```

where [other_commands] is the command you actually wan to run. If you have already started a job and want to nice it after the fact, you can use the renice command to do this. First identify the process ID that you wish to nice:

```
$ ps -u
```

Then run renice on it:

```
$ renice +5 [PID]
```

where [PID] is the process ID.

More Resources

Here are some links to more resources on learning technical computing and unix/linux skills:

- Software Carpentry
- Cyberwizard Institute
- Cyberwizard Lectures
2.1 Compute Servers

Here at ARC we provide two general purpose compute servers, Circe and Hecate.

2.1.1 Usage Rules and Guidelines

1. Programs that require large amounts of memory (100’s of GB or more) should be run on hecate.

2. Before you run your jobs on a server run htop to check the server load. It will be pretty obvious if the server you are on is under a heavy load (lots of green, red, yellow and blue). If so, checkout the other server to see if it is more available.

3. Please limit the number of threads a multi-threaded job uses. Be mindful of the other user’s jobs on the server you are using. If there is a heavy load on the server you are using, please nice your jobs so they do not clog up the system too much. If someone complains about your job consuming too many resources we will nice it for you.

4. If you want software installed, try building it locally before contacting us.

Circe

<table>
<thead>
<tr>
<th>Server Domain</th>
<th>circe.rc.pdx.edu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td></td>
</tr>
<tr>
<td>Ram</td>
<td>189GB</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 6.5</td>
</tr>
<tr>
<td>CPU</td>
<td>2 x Intel Xeon E5-2665</td>
</tr>
<tr>
<td>Clock</td>
<td>2.40 GHz</td>
</tr>
<tr>
<td>Cores</td>
<td>16 (2 x 8)</td>
</tr>
<tr>
<td>Special Hardware</td>
<td>NVIDIA GK110GL [Tesla K20m] (rev a1)</td>
</tr>
<tr>
<td>HyperThreading</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

Nvidia Tesla K20 GPGPU

Circe is equipped with a Tesla K20 GPGPU. A GPGPU, when utilized properly, can drastically improve the performance of certain types of computing jobs by increasing computational concurrency.

Nvidia has a list of software that can take advantage of their GPGPUs:
Nvidia Software List

Software that can use the GPGPU:

- WRF
- Gaussian G09
- G-BLASTN
- GPU-BLAST

Hecate

<table>
<thead>
<tr>
<th>Server Domain</th>
<th>hecate.rc.pdx.edu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Dell R720</td>
</tr>
<tr>
<td>Ram</td>
<td>757GB</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 6.5</td>
</tr>
<tr>
<td>CPU</td>
<td>2 x Intel Xeon E5-2690</td>
</tr>
<tr>
<td>Clock</td>
<td>2.90 Ghz</td>
</tr>
<tr>
<td>Cores</td>
<td>16 (2 x 8)</td>
</tr>
<tr>
<td>Special Hardware</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

2.2 Linux Clusters

2.2.1 Hydra Cluster

Hydra is a research cluster available to PSU students and faculty.

Hydra Specifications

<table>
<thead>
<tr>
<th>Server Domain</th>
<th>hydra.rc.pdx.edu</th>
<th>compute-0-[0-13]</th>
<th>compute-1-[0-11]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Dell PowerEdge R720</td>
<td>Dell PowerEdge R620</td>
<td>Dell PowerEdge R900</td>
</tr>
<tr>
<td>RAM</td>
<td>64GB</td>
<td>64GB</td>
<td>128GB</td>
</tr>
<tr>
<td>OS</td>
<td>Rocks 6.1.1</td>
<td>Rocks 6.1.1</td>
<td>Rocks 6.1.1</td>
</tr>
<tr>
<td>CPU</td>
<td>2 x Intel Xeon E5-2650</td>
<td>2 x Intel Xeon E5-2650</td>
<td>4 x Intel Xeon E7330</td>
</tr>
<tr>
<td>Clock</td>
<td>2.00GHz</td>
<td>2.00GHz</td>
<td>2.40GHz</td>
</tr>
<tr>
<td>Cores</td>
<td>16 (2 x 8)</td>
<td>16 (2 x 8)</td>
<td>16 (4 x 4)</td>
</tr>
<tr>
<td>Special Hardware</td>
<td>SSE4_2</td>
<td>SSE4_2</td>
<td></td>
</tr>
<tr>
<td>Hyper Threading</td>
<td>Enabled</td>
<td>Disabled</td>
<td>N/A</td>
</tr>
<tr>
<td>Network</td>
<td>N/A</td>
<td>10 Gigabit</td>
<td>1 Gigabit</td>
</tr>
</tbody>
</table>

Applications

Applications that have been compiled/configured locally are stored in:

/share/apps
Fig. 2.1: A photo of the Hydra Cluster servers (highlighted in red).
Storage

Hydra has a large scratch space that can be used for computations:

/disk/scratch/[user folder]
/disk/scratch2/[user folder]

We have two JBODs attached:

- Dell PowerVault MD1220 (scratch)
- Dell PowerVault MD1200 (scratch2)

2.3 Other Servers

2.3.1 Hera

Hera(.rc.pdx.edu) is a server that you can use to connect to research servers (Circe/Hecate/Hydra) from off campus. Hera is accessible without PSU’s VPN service.

Connecting to another research server from Hera:

1. first, ssh into Hera from your local machine
2. then, ssh onto the server that you want to use

And there you have it. That is all you need to do to connect to our infrastructure from the outside world.
3.1 Unix Software

This section contains pages that give information and primers on how to use standard unix utilities and tools. Found a useful unix tool everyone should know about? Write it up and send it our way!

3.1.1 Bash

Bash is the default shell on ARC’s systems. You are free to use a different shell if you’d like, however, all of our documentation only supports Bash implementations.

Check which shell you are running

Log into any research server and enter the following command:

```
echo $0
```

If you see any kind of output other than something like this:

```
> echo $0
-bash
```

then you are likely running some other shell by default, most likely tcsh or some variation of this. This command tells you what your active shell is.

What to do if my default shell is NOT bash?

Send us a quick request from your PDX email address requesting that your default research account shell be set to bash.

What to do in the meantime

You can always switch over to bash by typing the following into the console no matter what your default shell is set to after logging in:

```
> bash
```
3.1.2 Boost 1.58.0

The Boost Library is a large set of libraries for C++.

**Using Boost 1.58.0 on your system:**

To compile code that uses boost libraries use the -I flag to set the path to boost. The path to boost 1.58.0 is:

```
/vol/apps/system/boost-1.58.0
```

Example:

```
$> g++ -I /vol/apps/system/boost-1.58.0 example.cpp -o executable_example
```

**Note:** When you are building software that uses boost 1.5 with make or other install scripts: Check for a configure flag that sets the path to the boost library. If there is one, set it to the path specified above.

3.1.3 .dotfiles

Dotfiles (files that live in your home directory that start with a .) are hidden preference files that allow you to change your shell environment and settings for different unix programs. You can see these files by passing the -a flag to the `ls` command.

When you want to make changes to your environment that take effect every time you log in just add them to your `.bash_profile`. Careful though! If you screw this file’s syntax up you may break your `$PATH` making it difficult to edit the file to fix the problem.

3.1.4 Ganglia

The clusters can be monitored using the Ganglia webview from on campus at the following URLs:

- http://hydra.rc.pdx.edu/
- http://gravel.rc.pdx.edu/

These can be viewed off campus by using PSUs VPN.

3.1.5 GCC

ARC aims to supply robust up-to-date versions of GCC.

Currently we support the lastest release of every major version of GCC. If there is a version of GCC that isn’t supported that you need please let us know and we’ll get to building it for you.

**GCC on compute servers (Circe/Hecate):**

All versions of GCC can be found under:

```
/vol/apps/gcc/
```
3.1. Unix Software
Versions Currently Supported:

- 4.4.7 (system)
- 4.6.4
- 4.7.4
- 4.8.5
- 4.9.3
- 5.1.0
- 5.2.0
- 5.3.0
- 6.1.0

Languages Currently Supported:

- C
- C++
- Objective C
- Objective C++
- Fortran

Note: GCC on the compute servers does not support 32 bit

GCC on Clusters (Hydra/Deino):

We are currently working on building GCC on the clusters.
GCC has been built up to 4.9.3 on Deino.

Versions Currently Supported:

- 4.4.7 (system)
- 4.6.4
- 4.7.4
- 4.8.5
- 4.9.3

Languages Currently Supported:

- C
- C++
Using non-system GCC

Using a non-system GCC is done serveral different ways depending on your use-case. This document will cover two common scenarios you might encounter.

Compiling a single file:

First, set your LD_LIBRARY_PATH to include the libraries of gcc build you want to use (This configuration will change when you log out or change shells). Depending on what you are doing, this step might not be necessary.

```
export LD_LIBRARY_PATH=/vol/apps/gcc/gcc-6.1.0/lib:/vol/apps/gcc/gcc-6.1.0/lib64
```

After your library path is set, compile your file using the full path to the non-system gcc compiler.

```
/vol/apps/gcc/gcc-6.1.0/bin/gcc myfile.c
```

Compiling a program with make:

The process of using a non-system gcc to compile software with make differs between programs. Generally make will look for environment vairables durring the configure step that tell make which compiler and libraries to build the program with. To get a list of significant environment vairables run the following command inside the programs base directory.

```
./configure --help
```

Some typical enviroment vairables for setting compiler paths include:

- CC -- C compiler (gcc)
- CXX -- C++ compiler (g++)
- FC -- Fortran compiler (gfortran)
- F77 -- Fortran 77 compiler (typically gfortran)

Before configuring your software set the nesseary compiler environment variables and your library path variable in your current shell.

```
export LD_LIBRARY_PATH=/vol/apps/gcc/gcc-6.1.0/lib:/vol/apps/gcc/gcc-6.1.0/lib64
export CC=/vol/apps/gcc/gcc-6.1.0/bin/gcc
```

```
./configure
make
make install
```

Note: The LD_LIBRARY_PATH variable doesn’t always need to be set. Consult ./configure –help for more information
3.1.6 Go

Go is an open source programming language initially developed by Google. Go version 1.4.2 is currently installed on the compute servers, Crice and Hecate.

Setting up Go on your system:

Go will not work right off the bat, you need to export the GOROOT directory variable to where Go is located. The system install of go is currently located in /vol/apps/system/stow/go-1.4.2/.

Add this line to either your ~/.bash_profile or ~/.bash_rc:

```bash
export GOROOT=/vol/apps/system/stow/go-1.4.2/
```

To check if the Go is working properly, run this command:

```bash
$> go version
```

If GOROOT is set properly your terminal should output:

```bash
go version go1.4.2 linux/amd64
```

Setting up your own Go install:

step 1: Download Go source code from golang.org/dl and unpack.

step 2: export GOROOT path variable to where you unpacked the package. Put these lines into your either your ~/.profile or your ~/.bash_rc:

```bash
export GOROOT=/path_to_go_install
export PATH=$PATH:$GOROOT/bin
```

Additional Info and Links:

- Go Hompage
- Go Source Code
- Go Download Documentation

3.1.7 htop

Using htop to keep an eye on the server

You can use a program called htop to keep an eye on the server.

3.1.8 nice

Information on modifying your process priority using nice.
Fig. 3.1: A screenshot of the htop program running on hecate.
How to nice your processes

It is important to nice intensive processes so that they don’t interfere with using the most basic functions of the research server such as changing directories or moving files.

The nice and renice commands adjust the priority of a process. ARC recommends that all users nice all of their processes with a value of at least 5. To nice a process when it is started type:

```
$ nice +5 [other_commands]
```

where [other_commands] is the command you actually wan to run. If you have already started a job and want to nice it after the fact, you can use the renice command to do this. First identify the process ID that you wish to nice:

```
$ ps -u
```

Then run renice on it:

```
$ renice +5 [PID]
```

where [PID] is the process ID.

3.1.9 Perl

To use an updated perl version we need to use perlbrew and point it at the perlbrew install that works system wide.

Make sure you are running bash as your default shell the add the following two lines to the end of your .bashrc file:

```
export PERLBREW_ROOT=/vol/apps/system/perl5/
source $PERLBREW_ROOT/etc/bashrc
```

Run this command to see which versions of perl are available:

```
perlbrew list
```

Run this command to activate the version you want:

```
perlbrew switch perl-5.22.0
```

**Note:** Make sure you use the the correct version that is listed as available above.

Run this command to switch back to system perl:

```
perlbrew switch-off
```

Installing Local Perl Modules:

You can use perlbrew to install modules to your own system locally.

To install local modules first create a local library with perlbrew:

```
$> perlbrew lib create perl-5.22.0@mylocallibrary
```

This command creates a local library for perl 5.22.0 called “mylocallibrary”

You can use the use parameter to use that library only for the current shell:
$> perlbrew use perl-5.22.0@mylocallibrary

or the switch parameter to switch that library to the default:

$> perlbrew switch perl-5.22.0@mylocallibrary

Now once you have switched to or are using a local library you can go ahead and install perl modules locally with cpanm:

$> cpanm Moo

To check if your installation worked check it with:

$> perldoc perllocal

To delete a library:

$> perlbrew lib delete perl-5.22.0@mylocallibrary

### 3.1.10 Python

It is difficult to update python beyond the version CentOS provides at the system level. For this reason, ARC offers an alternate python environments using pyenv.

If you are interested in installing your own python packages to your home folder, you must do so in a virtual environment. In short, switch to the version of python that you want in your virtualenv using pyenv, create the virtualenv, and proceed to do all your development in that virtualenv going forward.

**Activate pyenv in your environment**

You must add the following lines in your `~/.bashrc` file:

**Compute servers (Circe/Hecate):**

```bash
export PYENV_ROOT="/vol/apps/system/python/pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
```

**Compute Clusters (Hydra/Gravel):**

```bash
export PYENV_ROOT="/share/apps/user/stow/pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
```

Now re-source your settings file (IE `source ~/.bashrc`) or log out and back in.

**Check which versions of python are available**

Now that the `pyenv` command is available, check which versions are installed:
This shows three available versions of python. The currently active version denoted by the *. In this case, system python is the currently active python version.

**Switch your session to the version you want**

To change to a different version of python use the shell command:

```
> pyenv shell 2.7.6
> python --version
Python 2.7.6
```

Now python 2.7.6 is the active python version provided by pyenv

**Create a virtual environment in your home directory**

Virtual Environments are local python installations that allow you to install python packages locally with pip without having to request an admin to install them for you. Virtualenvs are easy to setup and quite disposable if you mess up. The virtualenv command will create a virtual environment with the python version that is currently selected in pyenv:

```
> mkdir my_environment
> virtualenv my_environment
> source my_environment/bin/activate
```

Upon first activation, its a good idea to update pip and setuptools:

```
> pip install --upgrade pip
> pip install --upgrade setuptools
```

After you have created, updated and activated your new virtual environment you can install packages with pip freely. To exit your python environment type:

```
> deactivate
```

To reactivate an environment you have already created type:

```
> source my_environment/bin/activate
```

Now you can freely install packages with pip once again!!

**Using Pyenv and Virtualenv on Hydra**

Because hydra uses a scheduler to run jobs using pyenv requires a few extra steps to get jobs running. This section is assuming that you have already created a virtualenv with pyenv and virtualenv using the steps above.

**Using Full Path to Python**

If you want to run jobs with a python virtualenv in slurm you will need to use the full path to the python installed in your virtualenv in the commands in your slurm script.
Example: You created a virtualenv with python 2.7.7 called env2 that is in your home directory (/home/me/). You want to run a python script you wrote called my_script.py on the slurm scheduler.

If you were running a python script in slurm with the default system python this line would be in your slurm script:

```bash
> srun python my_script.py
```

But because you are using a personal virtualenv you need to specify **full path** to the python in your virtualenv (this line will go into your slurm batch file):

```bash
> srun /home/me/env2/bin/python my_script.py
```

**Setting Python Environment Variables**

**WAIT!!!** You are not completely ready to run jobs through slurm quite yet. Before you can run your jobs you need to set two environment variables, PYTHONPATH and PYTHONHOME:

**Setting PYTHONPATH**

PYTHONPATH points to extra libraries you want to use with your script. In our case, we point it to the default libraries found in the pyenv installation.

**Example:** Type this into the command line on your head node before you run your slurm script

```bash
> export PYTHONPATH=/share/apps/pyenv/versions/<version_you_are_using>/lib/python<version>
```

Replace `<version_you_are_using>` with the version of python you chose to make your env with.

**Example:** If you chose to use python 2.7.7 your PYTHONPATH would be:

```bash
> export PYTHONPATH=/share/apps/pyenv/versions/2.7.7/lib/python2.7
```

If you used python 3.4.3 your PYTHONPATH would be:

```bash
> export PYTHONPATH=/share/apps/pyenv/versions/3.4.3/lib/python3.4
```

**Setting PYTHONHOME**

PYTHONHOME is an environment variable that points to the python executable you want to use. PYTHONHOME should be the path to your virtualenv directory.

**Example:** You created a virtualenv with python 2.7.7 called env2 that is in your home directory (/home/me/).

```bash
> export PYTHONHOME=/home/me/env2
```

**Running your job**

After you have set these environment variables in your shell you are set to schedule jobs through slurm. These environment variables will only live as long as your session is open. You will have to set them every time you login and want to run a job with your virtualenv in slurm.

You can add PYTHONPATH and PYTHONHOME to your ~/.bashrc to make the changes last after you logout. This will likely cause issues when creating new virtualenvs, using pyenv and using the python interpreter outside of your env and slurm.

---

3.1. Unix Software
3.1.11 Rocks Cluster Distribution

Hydra, Deino, and Gravel all run ROCKS 6.1.1 with the following rolls installed:

- area51
- base
- bio
- ganglia
- hpc
- java
- kernal
- os
- perl
- python
- slurm

3.1.12 screen

Set up GNU screen for long running processes

screen is a simple way to start a process on the research servers that you want to keep running even after you are logged out of the session.

Configuring Screen

Before we start, let's add a simple configuration file that vastly improves the screen program's behavior.

Create a file in your home directory called .screenrc:

```bash
touch ~/.screenrc
```

Paste in the following settings using your preferred $EDITOR

```bash
# Source: https://gist.github.com/bcomnes/6689991/download
# Sets the screen to use login shells
shell -$SHELL

# To reload .screenrc Press Ctrl - a : source ~/.screenrc

# Turn of startup message
startup_message off

# This helps prevent the screen from doing funny things.
defflow off
defnonblock on

# Enable 256-color mode when screen is started with TERM=xterm-256color
# Taken from: http://frexx.de/xterm-256-notes/
# I took it from: http://www.robmeerman.co.uk/unix/256colours
```

```bash
touch ~/.screenrc
```
# Note that TERM != "xterm-256color" within a screen window. Rather it is
# "screen" or "screen-bce"

# terminfo and termcap for nice 256 color terminal
# allow bold colors - necessary for some reason
attrcolor b ".I"

# tell screen how to set colors. AB = background, AF=foreground
termcapinfo xterm-256color 'Co#256:AB=\E[48;5;%dm:AF=\E[38;5;%dm'

# Sets the status bar
caption string "%?%F{%= Bk}%? %C%A %D %d-%m-%Y %{= kB} %t%= %?%F{%= kB}%?%{= wk}%?%n "
hardstatus alwayslastline

hardstatus string '%{= kG}[ %{G}%H %{g}][%{= kw}%?%-Lw%?%{r}(%{W}%n* %f%t%?(%u)%?%{r})%{w}%?%+Lw%?%?%= %{g}]

Download screenrc
This changes some subtle behavior and turns on a status bar making screen a little bit easier to wrap your head around. It also turns off the annoying startup message.

Using Screen

Here is a quick walkthrough of how to use screen. To start using screen, run:

```
> screen
```

This opens a new screen session. Type some commands into the window such as listing files (ls) or changing directory (cd ..).

Now we can disconnect from this screen and have it continue to run in the background. screen uses keyboard shortcuts where you press two keys at once, let go, then press the next key to actually issue the command. First press the two keys at the same time:

```
Ctl-a
```

Let go, then press:

```
d
```

This should disconnect you from your screen session and take you back to where you were before you launch screen. You can have multiple screen sessions running at the same time (and have even more screen windows per screen session if you really want!).

Note: In the future, this kind of keyboard shortcut will be referred to as Ctl-a d

Reconnecting to screen

To reconnect to screen, we can type screen -ls similar to list the running screen sessions. (Similar to how ls will list the files in the current directory).

```
> screen -ls
There is a screen on:
  19250.pts-8.rocks (Detached)
1 Socket in /var/run/screen/S-bcomnes.
```

This lists the running screen sessions. Each session has a number associated with it. To reconnect to a particular screen session type:

3.1. Unix Software
screen -r 19250

where 19250 is the number associated with the screen session you want to connect to.

To end a screen session, reconnect to it, and just exit out of all the processes running and then end the session by typing:

exit

There are lots of cool screen features. Here is a quick rundown of screen window management:

Ctl-a c Create a window in the current screen session

Ctl-a n Go to the next window in the current screen session

Ctl-a d Disconnect from current screen session.

3.1.13 Slurm Cluster Scheduler

This section contains information on general slurm use. If this is your first time running slurm, it is recommended that you read over some of the basics on the official Slurm Website and watch this introductory video: Introduction to slurm tools video

Example Job Submission

To submit a job to the scheduler, first figure out what kind of resource allocation you need. Once you have that set up a launching script similar to the following:

Download example.sh
#!/bin/sh

## Run this file with the command line "sbatch example.sh" for a working demo.

## See http://slurm.schedmd.com/sbatch.html for all options
## The SBATCH lines are commented out but are still read by the Slurm scheduler
## ***Leave them commented out with a single hash mark!***

## To disable SBATCH commands, start the line with anything other than "#SBATCH"
#SBATCH # this is disabled
###SBATCH # so is this
#SBATCH # disabled
SBATCH # disabled

## Slurm SBATCH configuration options

## The name of the job that will appear in the output of squeue, qstat, etc.
#SBATCH --job-name=this-is-the-job-name

## max run time HH:MM:SS
#SBATCH --time=10:00:00

## -N, --nodes=<minnodes[-maxnodes]>
## Request that a minimum of minnodes nodes (servers) be allocated to this job.
## A maximum node count may also be specified with maxnodes.
#SBATCH --nodes 1-3

## -n, --ntasks=<number>
## The ntasks option is used to allocate resources for parallel jobs (OpenMPI, FreeMPI, etc.).
## Regular shell commands can also be run in parallel by toggling the ntasks option and prefixing the command with 'srun'.
## ntasks default value is 1
## THIS OPTION IS NOT USED TO RESERVE CPUS FOR MULTITHREADED JOBS; See --cpus-per-task
## Mulithreaded jobs only use one task. Asking for more tasks will make it harder for your job to schedule.
#SBATCH -n 1

## --cpus-per-task=<number>
## The cpus-per-task option reserves a set number of CPUs (cores) for each task you request.
## The cpus-per-task option can be used to reserve CPUs for a multithreaded job
## The default value is 1
#SBATCH --cpus-per-task=1

## For hydra, main and main2 are the available partitions. This line is safe to omit
## on gravel as CLUSTER is the only partition. Check /etc/partitions.conf for currently defined partitions.
#SBATCH --partition main2,main

3.1. Unix Software
## Command(s) to run.

### You specify what commands to run in your batch below.
### All commands will be run sequentially unless prefixed with the `srun` command
### To run this example in sbatch enter the command: `sbatch ./example.sh`
### The output from this example would be written to a file called slurm-XXX.out where XXX is the jobid
### The slurm out file will be located in the directory where sbatch was executed.

```bash
MESSAGE='Hello, world!'
echo ${MESSAGE}
```

### Another example command
```
# my_computation_worker /home/user/computation_worker.conf
```

Once you write a launcher script with the correct resource allocations, you can launch your script using the following command:

```bash
> sbatch ./example.sh
Submitted batch job 440
```

This submits your job to the scheduler. You can check the status of the job queue by running:

```bash
> squeue
```

<table>
<thead>
<tr>
<th>JOBJID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REAISON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>433</td>
<td>main2</td>
<td>trinity-</td>
<td>aromo</td>
<td>R 3:00:47:07</td>
<td>1</td>
<td>compute-1-9</td>
</tr>
<tr>
<td>439</td>
<td>main2</td>
<td>tip_plan</td>
<td>jblac2</td>
<td>R 2:24:55</td>
<td>8</td>
<td>compute-1-[7-8,10,12-16]</td>
</tr>
<tr>
<td>438</td>
<td>main2</td>
<td>fdtd_job</td>
<td>jblac2</td>
<td>R 2:37:18</td>
<td>8</td>
<td>compute-1-[0-6,11]</td>
</tr>
</tbody>
</table>

### Useful Slurm Commands

Here are a list of useful slurm commands.

- **scancel** is the tool for canceling your jobs.

```bash
> scancel [jobid]
```

- **scontrol** shows information about running jobs.

```bash
> scontrol show job [jobid]
```

Userld=jblac2(169223) GroupId=jblac2(54419)
Priority=10220 Nice=0 Account=jblac2 QOS=normal WCKey=*default
JobState=RUNNING Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
RunTime=02:40:04 TimeLimit=4-04:00:00 TimeMin=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
Partition=main2 AllocNode:Sid=hydra:1704
ReqNodeList=(null) ExcNodeList=(null)
 NodeList=compute-1-[0-6,11]
BatchHost=compute-1-0
NumNodes=8 NumCPUS=128 CPUs/Task=1 ReqB:S:C:T=0:0:**
Socks/Node=** NtasksPerN:B:S:C=16:0:** CoreSpec=0
MinCPUnode=16 MinMemoryNode=0 MinTmpDiskNode=0
Features=(null) Cgres=(null) Reservation=(null)
Shared=OK Contiguous=0 licenses=(null) Network=(null)
Command=/home/jblac2/job.sh tip_3d_trial_2/geometry.fsp
WorkDir=/home/jblac2
StdErr=/home/jblac2/slurm-438.out
sinfo show information about the state of the cluster:

```
> sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE  NODELIST
DEBUG       up      infinite  0      n/a
main        up      infinite  14     idle compute-0-[0-13]
main2*      up      infinite  17     alloc compute-1-[0-16]
main2*      up      infinite  1      idle compute-1-17
```

smap shows a visual representation of the cluster:

---

Useful Slurm Links

- Slurm Documentation
- Slurm Tutorials
- Slurm Publications
- Slurm At Harvard
3.1.14 ssh

SSH Tutorials

Set up SSH keys and config files

You can enable secure, password-free authentication to the ARC servers using SSH keys. SSH keys are a public/private key system that is more secure than traditional passwords, and offers a more convenient login mechanism than typing in your password every time you connect.

SSH Keys work by generating two cryptographic key files. One of the files is private (keep it a secret!) and the other is public (it doesn’t matter if someone gets a copy of it, but don’t unnecessarily distribute it).

Generating Keypair

On your computer create a ssh RSA keypair by typing:

```bash
$ ssh-keygen -t rsa
```

This creates a pair of keys (public `id_rsa.pub` and private `id_rsa`).

**Should I set a Passphrase for my Private Key? Yes!**

Because it is really bad if someone gets a copy of your private key (`id_rsa`) you should set a passphrase for your private key. This passphrase is used to encrypt the private key so that it cannot simply be used if a copy is made by a bad actor. It also means that a password must be used every time your private key is needed. In order to avoid this annoyance, most modern operating systems will provide a keychain system that can keep track of and auto submit this private key passphrase. On OS X, the `Keychain.app` will be your passphrase manager and will prompt you save the passphrase. Most Linux distributions will automatically offer to save the password using it’s flavor of passphrase manager. You should accept the offer and let your system remember this.

On windows, you can use a program such as `pageant` or `keepass` with the `KeeAgent` plugin.

Set folder permissions on server

Because of some existing configuration errors, home folders are created with incorrect permissions. In order for ssh keys to work, you must set these correct permissions on the appropriate folders and files.

Open a ssh connection to the server and run the following:

```bash
$ touch ~/.ssh/authorized_keys
$ chmod 711 ~ && chmod 711 ~/.ssh && chmod 600 ~/.ssh/authorized_keys
```

Upload public key to server

On your computer run the following:

```bash
$ cat ~/.ssh/id_rsa.pub | ssh [user]@[server]:~/.ssh/authorized_keys
```

(change user and server as appropriate)

The password-free login should now work. Connect by `ssh [user]@[server]`
Note: You can create a single public/private key per device you connect from, or create a single key pair that is used on all your devices. Each method has its pros and cons relating to key invalidation. If you generate a key per device, you can simply append additional public keys on new lines in the `~/.ssh/authorized_keys` file.

Create aliases for servers

Creating an ssh config file will let you create shortcut names to servers you frequently ssh into.

Create a file in `~/.ssh/` called `config` and add servers with the following format:

```plaintext
host [hostname]
  hostname [server]
  user [username]
```

Example:

```plaintext
host example
  hostname example.com
  user odinusername
```

Now you can connect to the server running:

```plaintext
ssh example
```

Which will connect to `odinusername@example.com`

3.1.15 stow

Stow is a simple program that lets you automatically create symlinks between subfolders in a `stow` directory to the parent directory of the `stow` folder. It is useful for managing built-from-source software. It is used in `/vol/apps/` but it is also useful in your home directory.

Setting up programs in your home directory with stow

You can use a simple program called GNU stow to manage built-from-source applications in your home directory.

First, create a `stow` and `src` folder in your home directory:

```plaintext
> cd ~
> mkdir stow
> mkdir src
```

Next download the program you wish to build from source to the `src` folder:

```plaintext
> cd ~/src
> wget http://example.com/program-1.0.0.tar.gz
... 
> tar xvzf program-1.0.0.tar.gz
... 
> cd program-1.0.0
```

Create a folder in the `stow` folder where you want your program installed:

```plaintext
> mkdir ~/stow/program-1.0.0
```
Read about available build options and decide if you need to change any default options:

```bash
> cd ~/src/program-1.0.0
> ./configure --help
... 
# Lots of options
```

Configure the program to install to the `stow` prefix that we just created and set any other options or flags that you need. This step may vary from program to program. You may encounter errors or warnings at this step so watch out and research any that come up:

```bash
> cd ~/src/program-1.0.0
> ./configure --prefix=~/stow/program-1.0.0
... 
# Lots of output
```

Next make the program from source. Watch for errors on this step and research them as the come up:

```bash
# in ~/src/program-1.0.0
> make
... 
# lots of output
```

Once your program builds successfully, install it to the `prefix` directory that we set in the configure step. Watch for errors on this step:

```bash
# in ~/src/program-1.0.0
> make install
```

If the install is successful, navigate to the program folder in the `stow` folder and inspect what was installed:

```bash
> cd ~/stow/program-1.0.0
> ls -a
bin  lib
# etc...
```

If you see files/folders not conforming to the standard Unix folder structure:

```bash
/bin
/include
/lib
/lib64
/local
/sbin
/share
/tmp
```

you should consider cleaning up the install as the install did not follow standard convention and may make a huge mess. If the program installed cleanly, you can `stow` the program:

```bash
> cd ~/stow
> stow -S program-1.0.0
```

Running this `stow` command with the `-S` flag (for save/stow?) symlinks the contents of `~/stow/program-1.0.0` into the directory above the `~/stow` folder. In this case this is your home folder `~/`. `bash` is configured to use the default Unix directories in the home folder by default, so now binaries from `program-1.0.0` should be in your `$PATH`.

Let's say we want to `un-stow` a program to remove it from our environment or to stow a new version. We can simply run:
> cd ~/stow
> stow -D program-1.0.0

Running `stow` with the `-D` flag (for delete?) deletes the symlinks created by stow from the step above.

`stow` is intelligent enough to not overwrite files and keeps track of everything it installs. Its simple, but effective.

## 3.2 Research Software

This section contains pages that give information and tutorials on how to use various research utilities used on the ARC research servers and clusters.

### 3.2.1 Lumerical FDTD

### 3.2.2 Gaussian g09

#### Topics

- Gaussian g09
  - Testing Gaussian
  - Parallelization With Linda

#### See also:

Gaussian Supports GPGPUs! Read about our [Nvidia Tesla K20 GPGPU](#) card. Gaussian Supports Parallelization!

Read the example: [Parallelization With Linda](#)

The Gaussian software is installed on linux clusters and is available for use if you are authorized to use the available license. You must be added to the `gaussian` unix group in order to run `g09` which should be specifically requested when requesting a research account.

Setting up g09

g09 requires some simple modifications to your user environment. Add the following to to your `~/.bashrc` file:

```bash
g09root="/share/apps"
GAUSS_SCRDIR=/scratch/$USER/gaussian_scratch
export g09root GAUSS_SCRDIR
source $g09root/g09/bsd/g09.profile
```

The `$GAUSS_SCRDIR` env variable is used as the Gaussian scratch folder. For now, leave this in your home directory and keep an eye on its size and clean up old files.

#### Testing Gaussian

**Warning:** Guassian will not run on the `gravel.rc.pdx.edu` cluster due to the lack of the SSE4_2 CPU instruction set.

You can test to make sure g09 is working properly and your environment is set up correctly by setting up a simple g09 test and then writing a schelulings script to submit the job to `slurm`, the cluster scheduler. The following is a simple test:

Download `g09-test.gif`
%nprocshared=8
%mem=2GB
%chk=test2.chk
# opt hf/sto-3g nosymm

Title Card Required

0 1
O -0.37773358 -0.54671967 0.00000000
H 0.58226642 -0.54671967 0.00000000
H -0.69818817 0.35821616 0.00000000

This test file will run a single g09 job using 8 threads and 4Gb of memory.

Next set up a simple slurm script to schedule your g09 job. Set up a simple bash script with some special directives in the header to do this:

Download g09-slurm.sh

#!/bin/sh
#SBATCH --job-name=g09-test

# max run time
#SBATCH --time=10:00:00

# The number of compute nodes to request
# g09 only needs 1 node per job, but n tasks that you specify with %nprocshared
# Figure out the optimal number of nprocshared per task for your
# job through benchmarking. You can also request exclusive mode with the --exclusive
# flag.
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8

# Specify slurm partition
#SBATCH --partition main

# command to run
srun g09 g09-test.gjf

To enqueue the job run:

sbatch g09-slurm.sh

Now check the queue to see if your job has been accepted:

squeue

We can keep an eye on activity using:

sinfo

or by visiting the ganglia monitoring tool.

For a more extensive test try the following g09 file which will fail on servers without the correct CPU instutions required by gaussian:

Download l2-PtCl-td.gjf
%nprocshared=8
%mem=4GB
%chk=12-PtCl-td.chk
# td=(nstates=30) b3lyp/gen nosymm pseudo=read

Title Card Required

<table>
<thead>
<tr>
<th>0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

3.2. Research Software
<table>
<thead>
<tr>
<th>Symbol</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>-0.27926800</td>
<td>-6.12260300</td>
<td>6.38220100</td>
</tr>
<tr>
<td>N</td>
<td>0.76660100</td>
<td>5.27319100</td>
<td>-0.44208000</td>
</tr>
<tr>
<td>N</td>
<td>0.24485600</td>
<td>7.42269900</td>
<td>-1.62245700</td>
</tr>
<tr>
<td>S</td>
<td>1.51690500</td>
<td>6.42886500</td>
<td>-1.33477900</td>
</tr>
<tr>
<td>S</td>
<td>0.94299900</td>
<td>-5.63128500</td>
<td>5.40797100</td>
</tr>
<tr>
<td>C</td>
<td>-4.40664900</td>
<td>-9.12127000</td>
<td>11.89377000</td>
</tr>
<tr>
<td>C</td>
<td>-5.17687700</td>
<td>-7.96734800</td>
<td>12.14869900</td>
</tr>
<tr>
<td>C</td>
<td>-4.34283600</td>
<td>-6.01827700</td>
<td>14.18091500</td>
</tr>
<tr>
<td>C</td>
<td>-5.52467100</td>
<td>-7.65712900</td>
<td>13.45343900</td>
</tr>
<tr>
<td>H</td>
<td>-5.49773100</td>
<td>-7.35662900</td>
<td>11.32218300</td>
</tr>
<tr>
<td>C</td>
<td>-3.81232100</td>
<td>-13.07506900</td>
<td>10.35141900</td>
</tr>
<tr>
<td>C</td>
<td>-4.37690000</td>
<td>-13.36762500</td>
<td>9.11300500</td>
</tr>
<tr>
<td>H</td>
<td>-5.20418400</td>
<td>-12.49517500</td>
<td>7.32256000</td>
</tr>
<tr>
<td>H</td>
<td>-3.41484000</td>
<td>-13.83931000</td>
<td>11.00485000</td>
</tr>
<tr>
<td>H</td>
<td>-4.42850700</td>
<td>-14.39917000</td>
<td>8.78939000</td>
</tr>
<tr>
<td>C</td>
<td>-4.72718300</td>
<td>12.23479500</td>
<td>-4.60446800</td>
</tr>
<tr>
<td>C</td>
<td>-4.73955000</td>
<td>11.17103400</td>
<td>-5.51991100</td>
</tr>
<tr>
<td>C</td>
<td>-6.61586700</td>
<td>13.15471800</td>
<td>-5.63177100</td>
</tr>
<tr>
<td>C</td>
<td>-5.71757200</td>
<td>11.12029600</td>
<td>-6.50294900</td>
</tr>
<tr>
<td>H</td>
<td>-3.98134700</td>
<td>10.40268900</td>
<td>-5.45488900</td>
</tr>
<tr>
<td>C</td>
<td>-6.66488300</td>
<td>12.14225100</td>
<td>-6.57481300</td>
</tr>
<tr>
<td>H</td>
<td>-7.34871100</td>
<td>13.94933500</td>
<td>-5.59515200</td>
</tr>
<tr>
<td>H</td>
<td>-5.72584300</td>
<td>10.30085900</td>
<td>-7.21559900</td>
</tr>
<tr>
<td>C</td>
<td>-7.43919500</td>
<td>12.15381300</td>
<td>-7.33538000</td>
</tr>
<tr>
<td>C</td>
<td>-3.04183100</td>
<td>13.50639100</td>
<td>-3.28964600</td>
</tr>
<tr>
<td>C</td>
<td>-1.65019090</td>
<td>13.49879100</td>
<td>-3.10856300</td>
</tr>
<tr>
<td>C</td>
<td>-3.09417500</td>
<td>15.80897000</td>
<td>-2.88832600</td>
</tr>
<tr>
<td>C</td>
<td>-0.98894500</td>
<td>14.68226400</td>
<td>-2.81294000</td>
</tr>
<tr>
<td>H</td>
<td>-1.10503200</td>
<td>12.56981000</td>
<td>-3.20764500</td>
</tr>
<tr>
<td>C</td>
<td>-1.72118400</td>
<td>15.86649500</td>
<td>-2.71981700</td>
</tr>
<tr>
<td>H</td>
<td>-3.72850700</td>
<td>16.67813400</td>
<td>-2.77830800</td>
</tr>
<tr>
<td>H</td>
<td>0.08841800</td>
<td>14.68018200</td>
<td>-2.67644900</td>
</tr>
<tr>
<td>H</td>
<td>-1.24743500</td>
<td>16.81777900</td>
<td>-2.50536000</td>
</tr>
<tr>
<td>N</td>
<td>-3.74790900</td>
<td>14.64913000</td>
<td>-3.13885600</td>
</tr>
<tr>
<td>N</td>
<td>-3.67841900</td>
<td>-11.80553800</td>
<td>10.80506300</td>
</tr>
<tr>
<td>C</td>
<td>1.90408000</td>
<td>0.99546100</td>
<td>3.02272300</td>
</tr>
<tr>
<td>H</td>
<td>1.60934300</td>
<td>1.56239600</td>
<td>3.91115000</td>
</tr>
<tr>
<td>H</td>
<td>2.49848300</td>
<td>0.13329700</td>
<td>3.34569400</td>
</tr>
<tr>
<td>H</td>
<td>2.54291900</td>
<td>1.63447500</td>
<td>2.40267500</td>
</tr>
<tr>
<td>C</td>
<td>1.10500100</td>
<td>-0.26760300</td>
<td>0.97312800</td>
</tr>
<tr>
<td>H</td>
<td>1.68603700</td>
<td>-1.14965400</td>
<td>1.26503600</td>
</tr>
<tr>
<td>H</td>
<td>0.23864600</td>
<td>-0.60305700</td>
<td>0.39464700</td>
</tr>
<tr>
<td>H</td>
<td>1.73363000</td>
<td>0.35254100</td>
<td>0.32412300</td>
</tr>
<tr>
<td>N</td>
<td>-5.68518500</td>
<td>13.18764500</td>
<td>-4.64808200</td>
</tr>
<tr>
<td>N</td>
<td>-3.97023600</td>
<td>-9.89597400</td>
<td>12.91206600</td>
</tr>
<tr>
<td>S</td>
<td>-2.54644600</td>
<td>2.87372600</td>
<td>2.03405300</td>
</tr>
<tr>
<td>S</td>
<td>-2.64337900</td>
<td>-0.52444300</td>
<td>4.16929700</td>
</tr>
<tr>
<td>Cl</td>
<td>-1.13350700</td>
<td>-13.17743500</td>
<td>12.06869000</td>
</tr>
<tr>
<td>Cl</td>
<td>-1.47762200</td>
<td>-10.95005100</td>
<td>14.54071300</td>
</tr>
<tr>
<td>Cl</td>
<td>-5.89484700</td>
<td>16.20066100</td>
<td>-1.42115400</td>
</tr>
</tbody>
</table>
Try editing or copying the g09-slurm.sh to point to the l2-PtCl-td.gjf file and launch a second job on the scheduler.

**Parallelization With Linda**

Gaussian g09 jobs can be run in parallel across multiple nodes which may increase performance and decrease runtime of jobs if done correctly. Documentation is spotty, but here is a sample to help you started. More information can be found in the dynamic hostfile example.

A few more notes from the developers:

- “%NProcs=” (short for “%NProcShared”) in the input file requests the number of cores (processors) to use via shared-memory parallelization (number per Linda worker)
- It is also possible to pass the number as an environment variable e.g. “GAUSS_PDEF=16” for 16 shared-memory cores.
- “%NProcShared=” in the input takes precedence over “GAUSS_PDEF”, so one could override the latter by setting “%NProcShared” in the input file.

```
#!/bin/bash
#SBATCH --job-name=g09-test

# max run time
#SBATCH --time=10:00:00

# 4 servers, 1 linda worker per server, 16 CPUs per linda worker
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16

# Specify slurm partition
#SBATCH --partition main

# activate bash debugging from here
set -x
```
Name of your gjf file ie l2-PtCl-td.gjf
JobFile=l2-PtCl-td

This creates a list of nodes that you job received to run on
LindaList=./nodes_linda.$SLURM_JOBID
touch $LindaList

This creates a jobfile
JobName=./${JobFile}${SLURM_JOBID}.gjf
touch $JobName

Create a list of hostnames and save it to the LindaList machine file
srun hostname -s | sort -u > $LindaList

Tell linda to use ssh
export GAUSS_LFLAGS=" -opt "Tsnet.Node.lindarsharg: ssh"

Read the contents of the machine file and put it in the job file
workers="$LindaWorkers=$(cat $LindaList | tr \"\n\" \," | sed s/\/,/\"")

Write that out to the job file
cat <(echo -e "$workers\r") ./$JobFile.gjf > $JobName

Run gaussian using our job file and output to a matching results file
g09 /scratch/bcomnes/g09/linda/$JobFile/SLURM_JOBID.gjf /scratch/bcomnes/g09/linda/foo$SLURM_JOBID

### 3.2.3 MPI and MPICH

#### System MPI

Message passing has been installed on the research system, but has to be enabled using the module system. Here are examples on how to do that if you need those:

module load openmpi-x86_64
module load mpich2-x86_64

These commands can be added to your .bashrc if you need them routinely or dynamically loaded from shell scripts that launch your MPI jobs. You may be interested in running this on the Linux Clusters.

#### 3.2.4 PGI Cluster Development Kit

**Topics**

- **PGI Cluster Development Kit**
  - **PGI Cluster Development Kit: Research Servers**
  - **Portland Group Cluster Development Kit (PGICDK)**

**PGI Cluster Development Kit: Research Servers**

To use the PGI compiler you have to enable it. While it is worth reading through the PGI documentation on how to fully do this, here is a quick set of instructions to get you started.
PGI is installed in:
/vol/apps/system/pgicdk

You can gain access to its tools by adding the following to your .bashrc file:

```bash
export PGI=/vol/apps/system/pgi
export PATH=$PGI/linux86-64/2014/bin:$PATH
export MANPATH=$MANPATH:$PGI/linux86-64/2014/man
export LM_LICENSE_FILE=$LM_LICENSE_FILE:27005@pgi.license.pdx.edu:$PGI/license.dat
```

PGI comes with its own versions of openMPI and MPICH. You can enable MPICH for example, by these by adding this to your .bashrc file as well:

```bash
export PATH=$PGI/linux86-64/14.6/mpi/mpich/bin:$PATH
export MANPATH=$MANPATH:$PGI/linux86-64/14.6/mpi/mpich/man
```

ARC has a license for the cluster development kit.

**Portland Group Cluster Development Kit (PGICDK)**

ARC provides access to a Portland Group (PGI) Cluster Development Kit license. Some of the popular tools in this software package include:

- `PGFORTRAN`
- `PGCC`
- `PGC++`

It also includes tools that can take advantage of NVIDIA CUDA running on supported devices such as the Nvidia Tesla K20 GPU installed in Circe. Using PGI requires that you set up the tools in your user environment. This varies from system to system so please refer to the specific system you wish to use PGI on.

- `PGI Cluster Development Kit: Research Servers`
- `pgiCluster`
4.1 How to Edit

Arc-docs is a set of documentation that is built and hosted on ReadTheDocs. ReadTheDocs is a service that builds the documentation from a git repository whenever it receives a webhook event which is fired every time a new commit is made to that repo. The Arc-Docs repo is hosted on github: https://github.com/PSU-OIT-ARC/arc-docs

The easiest way to edit Arc-Docs is by navigating to the page you want to make a change to, then click the Edit on Github button in the top right corner of the page. This will take you to that file's page on the github repo which will allow you to edit it in the browser.

Most changes can be made directly to the shared repo in the PSU-OIT-ARC organization, but you are free to use the fork, modify pull request pattern if you wish for larger changes or changes you want feedback on.

4.1.1 ReadTheDocs and Sphinx

ReadTheDocs fully supports the Sphinx documentation tool, which has quickly become the standard tool for documenting python projects. Sphinx is a superset of reStructuredText which is largely a superset of Markdown. It tends to be a bit more strict syntactically, but allows for much greater control over complex references and included codeblocks.

Useful RST References

- reStructuredText Primer
- sublime-text-unofficial-documentation provides a nice model layout of RTD documents.

4.1.2 RST Tools

It benefits from a well configured technical text editor such as vim, emacs, or Sublime Text.

SublimeText RST Tools

- reStructuredText Improved
- Restructured Text (RST) Snippets
- Table Editor
- SublimeLinter-rst
Online Tools

• Online Sphinx Editor