A Brief Introduction to The Center for Advanced Computing

May 1, 2006

Hardware

- 324 Opteron nodes, over 700 cores
- 105 Athlon nodes, 210 cores
- 64 Apple nodes, 128 cores
- Gigabit networking, Myrinet networking, Infiniband networking soon

Hardware: nyx

- nyx is the Opteron cluster; `nyx-login.engin.umich.edu` is the login host for this cluster
- Currently has 2TB NFS file system, soon to have 6TB, much improved NFS file system
- Running RedHat Enterprise Linux 4

Software

- fftw – Fast Fourier Transform Library (parallel and serial)
- fluent – fluid dynamics application
- gaussian – electro-chemical analysis application
- hdf5 – data format library
- java – Sun's Java Language
- mathematica – symbolic math application
- matlab – matrix math application
- nag – Numerical Algorithm Group's Fortran Compilers
- netcdf – data format library
- openmpi – MPI libraries
- pgi – Portland Group Compilers
- R – matrix math application
- simpson – solid-state NMR simulation software

Current List of Software

To get a current list of software on the cluster you are using, type `module avail`, you'll see something like:

```
---------------------- /home/software/rhel4/Modules/3.2.1/modulefiles ----------------------
R/2.2.1-gcc       gaussian/03-64bit module-cvs        null              radmind/1.5.1
dot               hdf5/1.6.5-gcc    module-info       openmpi/1.0.1-gcc simpson/1.1.1-gcc
fftw/2.1.5-gcc    hdf5/1.6.5-pgi    modules           openmpi/1.0.2-gcc simpson/1.1.1-pgi
fftw/2.1.5-pgi    java/1.5.0_06     nag/7             openmpi/1.0.2-pgi torque
fluent/6.2        mathematica/5.2   netcdf/3.6.1-gcc  pdsh              use.own
gaussian/03-32bit matlab/7.1        netcdf/3.6.1-pgi  pgi/6.1(default)
```

- To select a software package, type: `module load` *`package/version`*.
- To see what you have loaded, type: `module list`
- For help with the `module` command type: `module help`

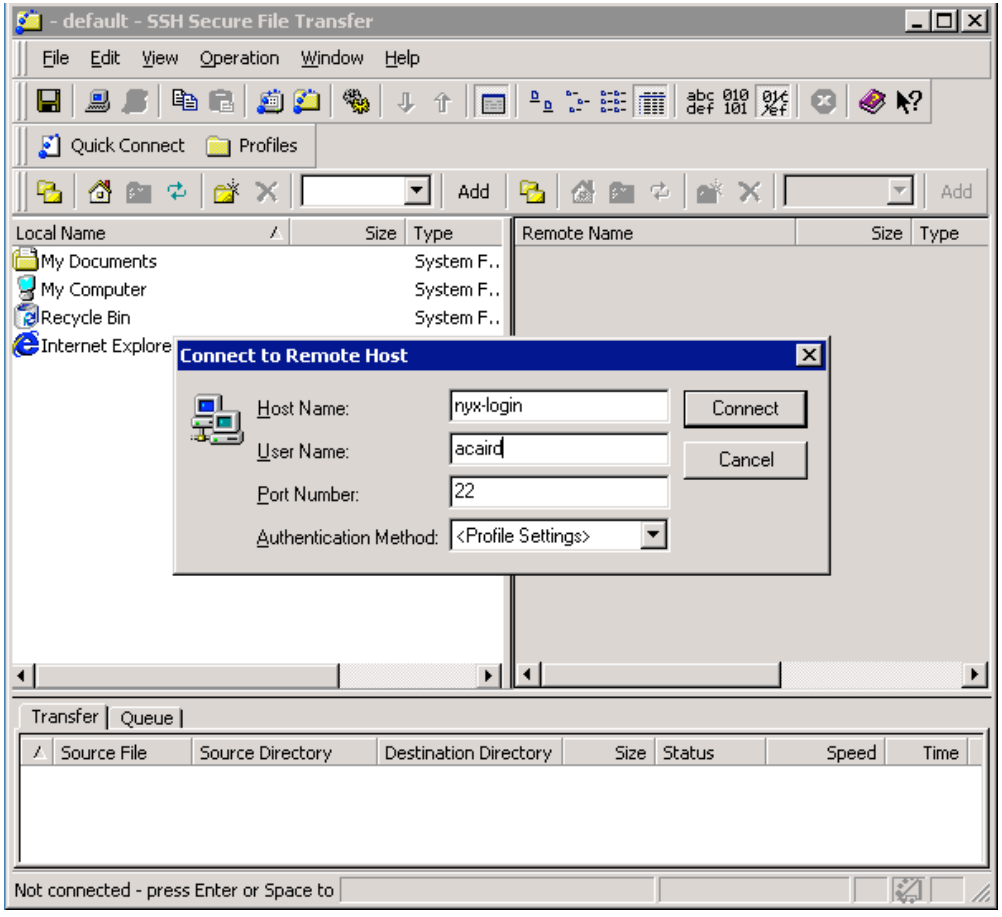
We load some basic utilities be default, so when you first log in you will see the Torque/PBS commands and the PGI compilers in your list of loaded modules.

Transferring Files

- Files are transferred to the file space on the clusters using either Secure Copy (`scp`) or Secure FTP (`sftp`).
- Your password for file transfers and logins is your UM Kerberos (Level-1) password and your login is your Uniqname.

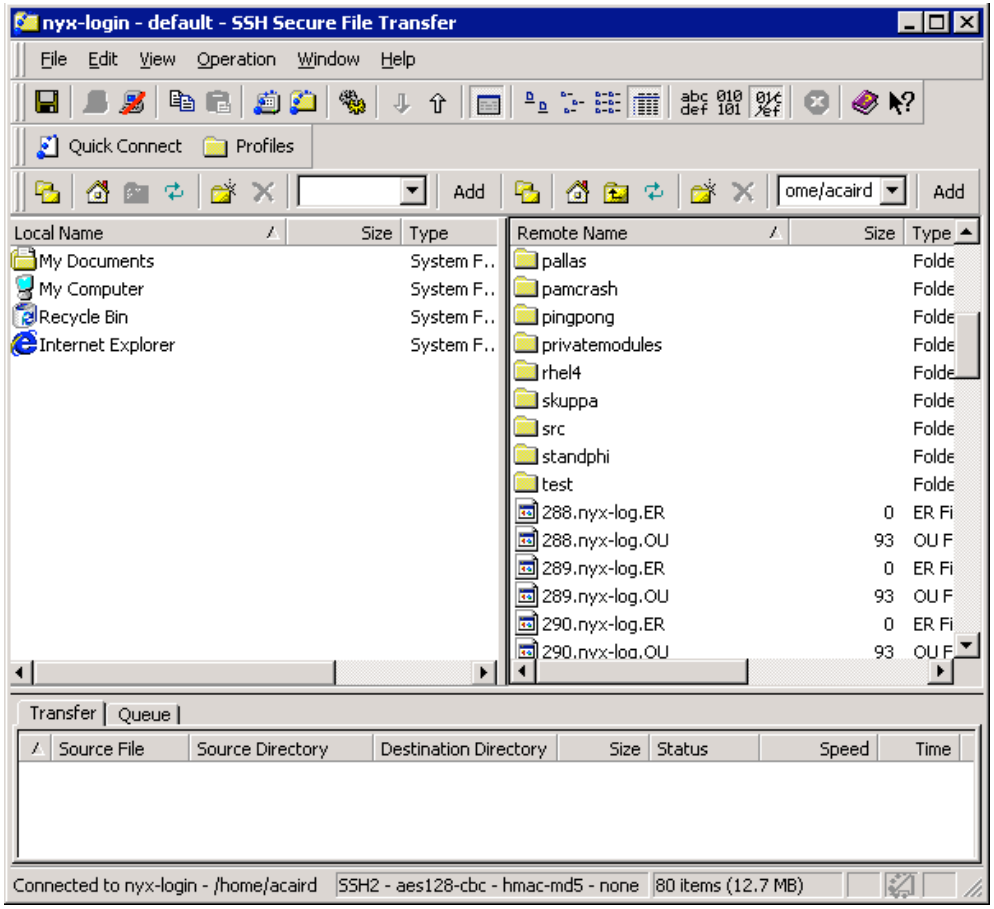
File Transfers: Windows

- SSH Secure Communications' Secure File Transfer
- click "Quick Connect":



File Transfers: Windows

- agree to add key to local database (only happens once), click "OK" on "SSH Authentication Response"
- You will see:



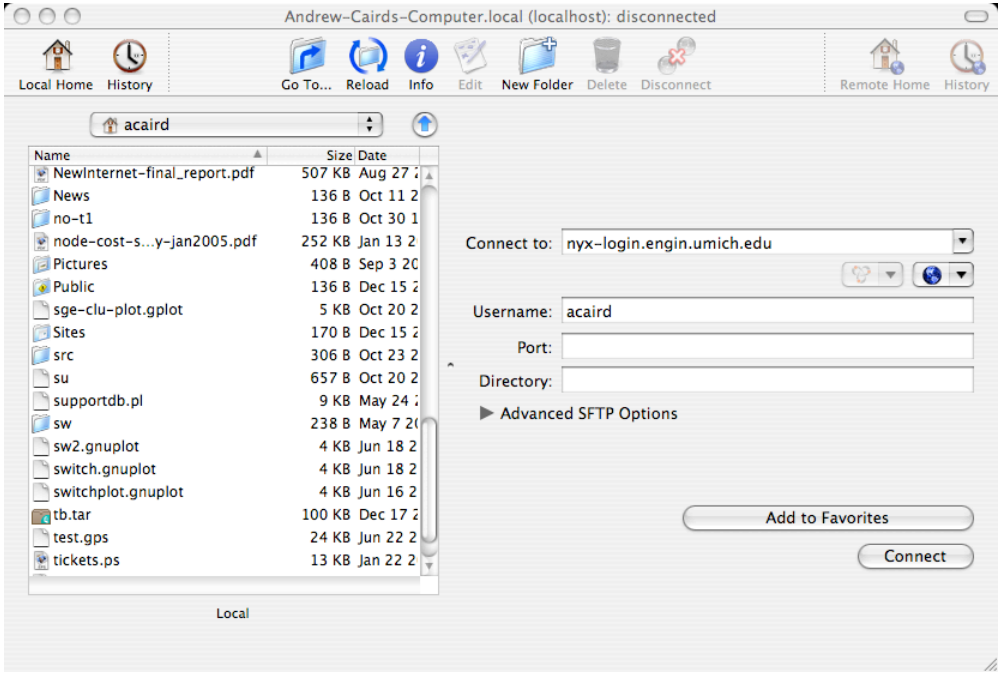
- You can drag and drop files back and forth

File Transfers: Windows

- There are other programs for Windows besides SSH's SCP program, any modern SCP/SFTP program will work
- SSH Secure Communications: `http://www.ssh.com`
- WinSCP: `http://winscp.net/eng/index.php`
- Putty: `http://www.chiark.greenend.org.uk/~sgtatham/putty/`
- Cygwin: `http://www.cygwin.com/`
- lots of others, see Google

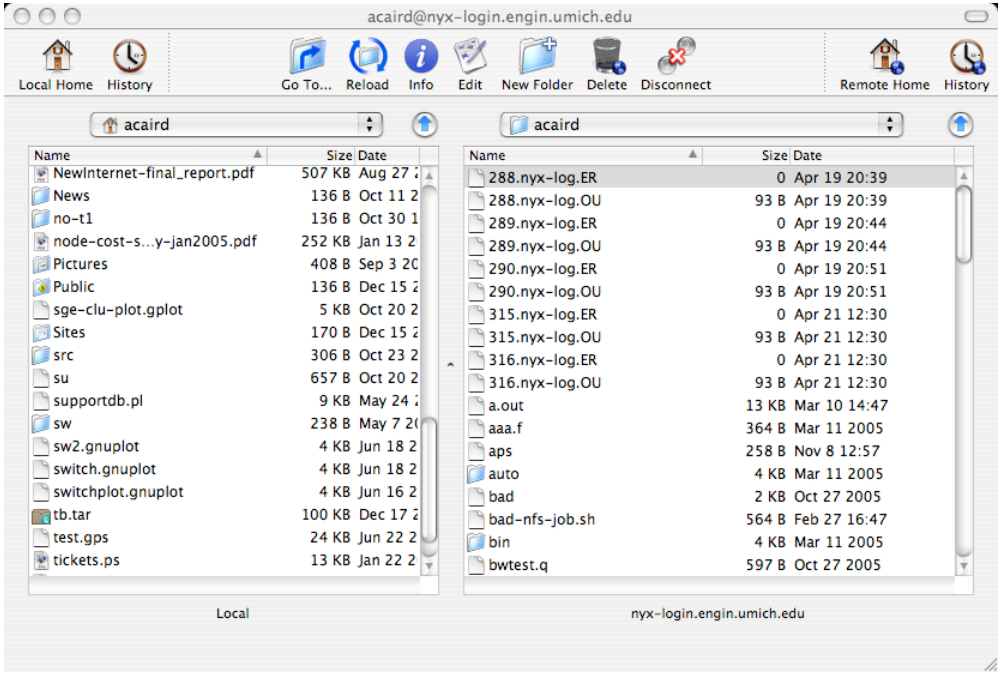
File Transfers: Mac

- UM/RSG's Fugu
- Fill the in information as shown:



File Transfers: Mac

- Enter password when prompted
- You will see:



- You can drag and drop files back and forth

File Transfers: Mac

- There are other programs besides Fugu
- Fugu: `http://rsug.itd.umich.edu/software/fugu/`
- Built-in `scp`/`sftp` from Terminal
- Rbrowser: `http://www.rbrowser.com/`
- Fetch: `http://fetchsoftworks.com/`
- lots of others, see Google

File Transfers: Linux

- Using `scp`:

```
% scp -r src nyx-login:
Password:
[...]
MP_memcpy.c          100% |*****************************************|  6784       00:00
armci.c              100% |*****************************************|  7590       00:00
gm.c                 100% |*****************************************|  6432       00:00
gpshmem.c            100% |*****************************************|  2611       00:00
ib.c                 100% |*****************************************| 31924       00:00
[...]
```

- Using `sftp`:

```
% sftp nyx-login
Connecting to nyx-login...
Password:
sftp>
```

- This works from the Mac Terminal, too.

Logging in

- Your login is your Uniquname
- Your password is your ITD/ITCS Kerberos password (Level 1 password)
- Use `ssh` to connect to the clusters
- All access is command line — there is no graphical access to the clusters; any graphical pre- or post-processing should be done on your own computer
- For tutorials on using Linux, see:
  - Introduction to Linux

    http://www.engin.umich.edu/caen/technotes/introunix/
  - Advanced Linux

    http://www.engin.umich.edu/caen/technotes/advancedunix/
  - Linux Commands

    http://www.engin.umich.edu/caen/technotes/unixcommands/

Logging in

- Your login is your Uniquname
- Your password is your ITD/ITCS Kerberos password (Level 1 password)
- Use `ssh` to connect to the clusters
- All access is command line — there is no graphical access to the clusters; any graphical pre- or post-processing should be done on your own computer
- For tutorials on using Linux, see:
  - Introduction to Linux

    http://www.engin.umich.edu/caen/technotes/introunix/
  - Advanced Linux

    http://www.engin.umich.edu/caen/technotes/advancedunix/
  - Linux Commands

    http://www.engin.umich.edu/caen/technotes/unixcommands/

Logging in

- Your login is your Uniquname
- Your password is your ITD/ITCS Kerberos password (Level 1 password)
- Use `ssh` to connect to the clusters
- All access is command line — there is no graphical access to the clusters; any graphical pre- or post-processing should be done on your own computer
- For tutorials on using Linux, see:
  - Introduction to Linux

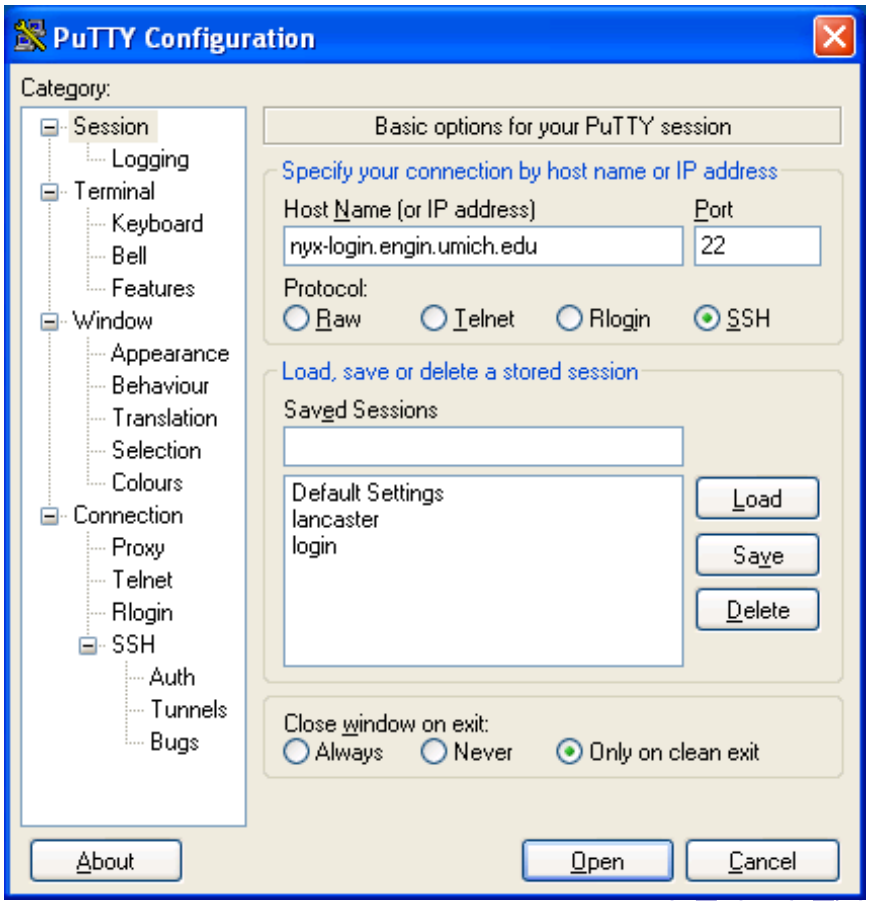
    `http://www.engin.umich.edu/caen/technotes/introunix/`
  - Advanced Linux

    `http://www.engin.umich.edu/caen/technotes/advancedunix/`
  - Linux Commands

    `http://www.engin.umich.edu/caen/technotes/unixcommands/`

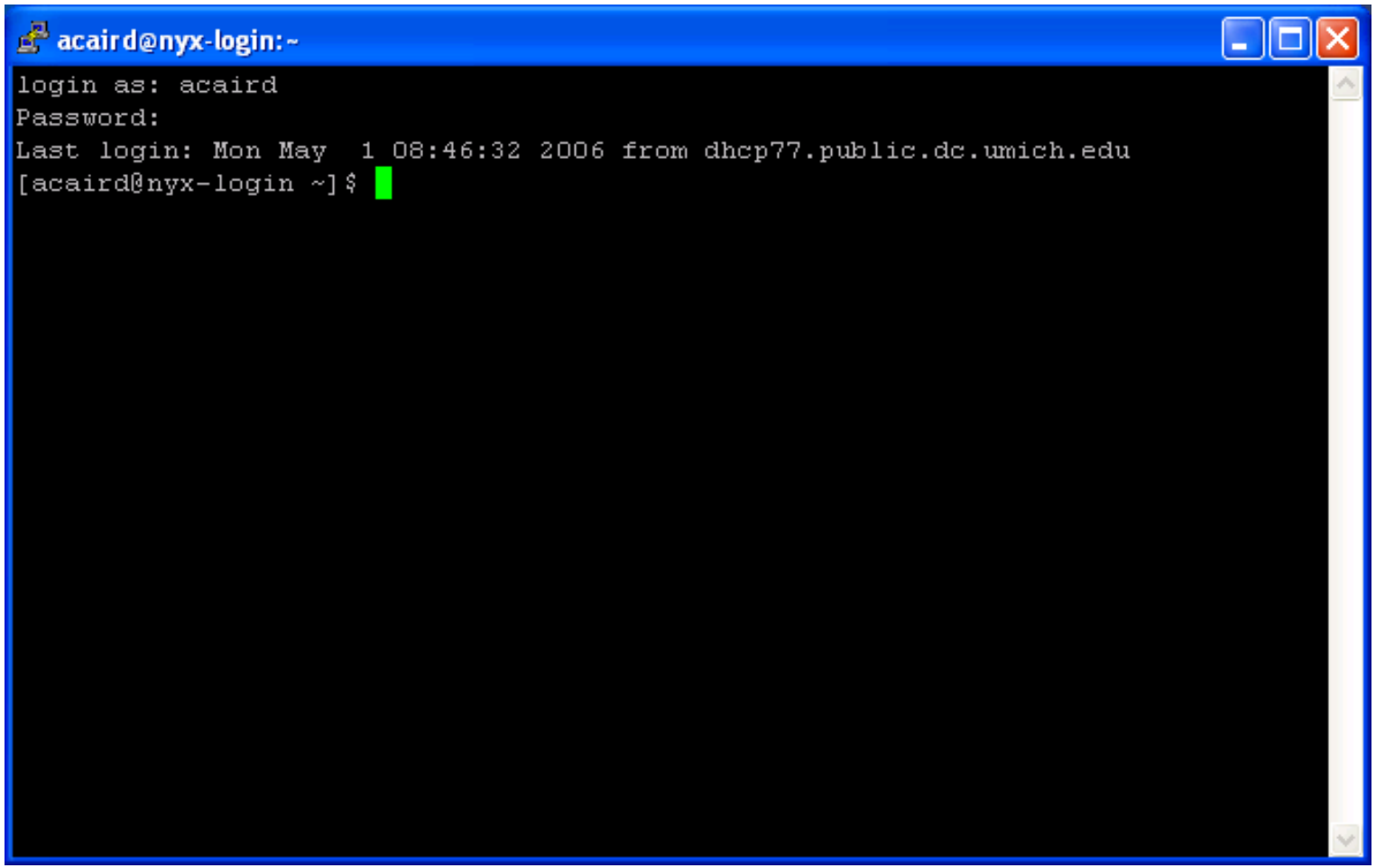
Logging in: Windows

- Putty is a freely available SSH client for windows `http://www.chiark.greenend.org.uk/~sgtatham/putty/`
- To log in, enter the host as shown:



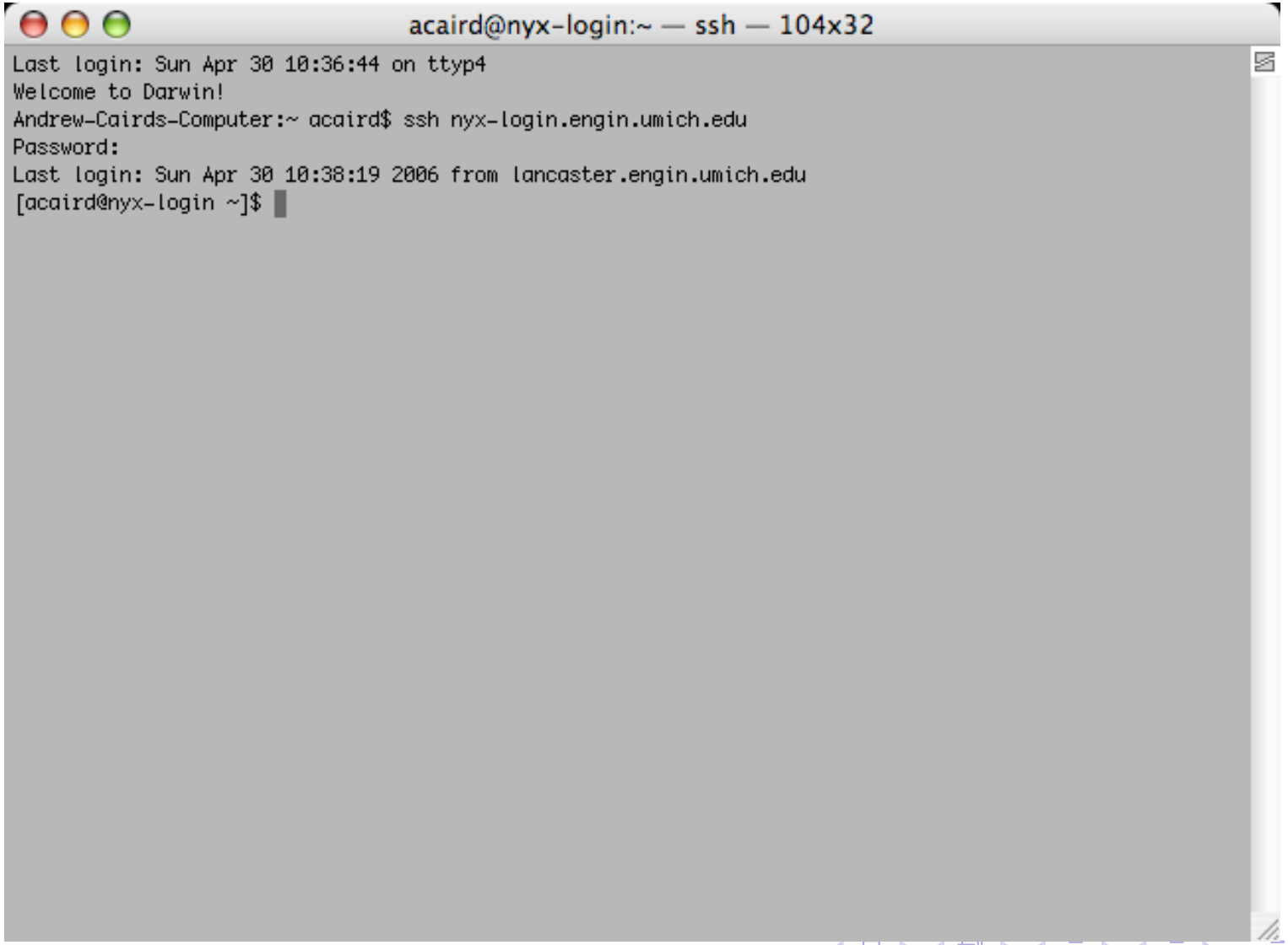
Logging in: Windows

- Then enter your Uniqname and password and you'll get the shell prompt:



Logging in: Mac

- Use the included SSH client from the Terminal program:



Logging in: Linux

- Use the included SSH client from and shell:

Compilers and Tools

- We support the Portland Group compilers (C, C++, Fortran77, Fortran90) and the Gnu compilers (gcc, g++) on Linux
- Like all software, this is selected by loading the proper module, by default we load a recent version of the Portland Group compilers; if you want a different version, you can unload the default and load a different version
- We also load a recent version of MPI for compiling (mpicc, mpiCC, mpif77, mpif90) and running parallel programs; you can change versions of this as well using the `module` command
- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.

Compilers and Tools

- We support the Portland Group compilers (C, C++, Fortran77, Fortran90) and the Gnu compilers (gcc, g++) on Linux
- Like all software, this is selected by loading the proper module, by default we load a recent version of the Portland Group compilers; if you want a different version, you can unload the default and load a different version
- We also load a recent version of MPI for compiling (mpicc, mpiCC, mpif77, mpif90) and running parallel programs; you can change versions of this as well using the `module` command
- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.

Compilers and Tools

- We support the Portland Group compilers (C, C++, Fortran77, Fortran90) and the Gnu compilers (gcc, g++) on Linux
- Like all software, this is selected by loading the proper module, by default we load a recent version of the Portland Group compilers; if you want a different version, you can unload the default and load a different version
- We also load a recent version of MPI for compiling (mpicc, mpiCC, mpif77, mpif90) and running parallel programs; you can change versions of this as well using the `module` command
- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.

Compilers and Tools

- We support the Portland Group compilers (C, C++, Fortran77, Fortran90) and the Gnu compilers (gcc, g++) on Linux
- Like all software, this is selected by loading the proper module, by default we load a recent version of the Portland Group compilers; if you want a different version, you can unload the default and load a different version
- We also load a recent version of MPI for compiling (mpicc, mpiCC, mpif77, mpif90) and running parallel programs; you can change versions of this as well using the `module` command
- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.

Introduction to the PBS Batch System

- All access to the compute nodes (everything other than the login node) is via the batch system
- We use a system called Torque, it is derived from PBS
- The batch system controls access to queues
- The scheduling system decides if and where jobs can run
- There are two general queues: `long` and `short`
- There are many private queues for people who own or rent nodes
- You have access to the private queue called `bio1`

Introduction to the PBS Batch System

- All access to the compute nodes (everything other than the login node) is via the batch system
- We use a system called Torque, it is derived from PBS
- The batch system controls access to queues
- The scheduling system decides if and where jobs can run
- There are two general queues: `long` and `short`
- There are many private queues for people who own or rent nodes
- You have access to the private queue called `bio1`

Introduction to the PBS Batch System

The steps to using the batch system are:

1. Create a batch file: this is a short (5-15 lines) text file with some batch commands and the commands to run your program
2. Submit the file to the batch system
3. Check on the status of your job
4. Delete your job if you want to cancel it

Creating a PBS Batch File

A simple example using **R**:

```
#!/bin/sh
#PBS -N r-job
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M acaird@umich.edu
#PBS -q bio1
#PBS -joe
R --save < input.R
```

Creating a PBS Batch File

A simple example using **R**:

```
#!/bin/sh
#PBS -N r-job
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M acaird@umich.edu
#PBS -q bio1
#PBS -joe
R --save < input.R
```

Creating a PBS Batch File

A simple example using **R**:

```
#!/bin/sh
#PBS -N r-job
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M acaird@umich.edu
#PBS -q bio1
#PBS -joe
R --save < input.R
```

Creating a PBS Batch File

A simple example using **R**:

```
#!/bin/sh
#PBS -N r-job
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M acaird@umich.edu
#PBS -q bio1
#PBS -joe
R --save < input.R
```

Creating a PBS Batch File

A simple example using **R**:

```
#!/bin/sh
#PBS -N r-job
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M acaird@umich.edu
#PBS -q bio1
#PBS -joe
R --save < input.R
```

Creating a PBS Batch File

A simple example using **R**:

```
#!/bin/sh
#PBS -N r-job
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M acaird@umich.edu
#PBS -q bio1
#PBS -joe
R --save < input.R
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N hpl-8x2
#PBS -l nodes=8:ppn=2:opt240,walltime=8:00:00
#PBS -q route
#PBS -M acaird@umich.edu
#PBS -m ae
#PBS -j oe
cd ${HOME}/HPL/
echo "I ran on: "
cat $PBS_NODEFILE
mpiexec -np 16 ${HOME}/HPL/hpl-oop/bin/OOMP/xhpl
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N hpl-8x2
#PBS -l nodes=8:ppn=2:opt240,walltime=8:00:00
#PBS -q route
#PBS -M acaird@umich.edu
#PBS -m ae
#PBS -j oe
cd ${HOME}/HPL/
echo "I ran on: "
cat $PBS_NODEFILE
mpiexec -np 16 ${HOME}/HPL/hpl-oop/bin/OOMP/xhpl
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N hpl-8x2
#PBS -l nodes=8:ppn=2:opt240,walltime=8:00:00
#PBS -q route
#PBS -M acaird@umich.edu
#PBS -m ae
#PBS -j oe
cd ${HOME}/HPL/
echo "I ran on: "
cat $PBS_NODEFILE
mpiexec -np 16 ${HOME}/HPL/hpl-oop/bin/OOMP/xhpl
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N hpl-8x2
#PBS -l nodes=8:ppn=2:opt240,walltime=8:00:00
#PBS -q route
#PBS -M acaird@umich.edu
#PBS -m ae
#PBS -j oe
cd ${HOME}/HPL/
echo "I ran on: "
cat $PBS_NODEFILE
mpiexec -np 16 ${HOME}/HPL/hpl-oop/bin/OOMP/xhpl
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N hpl-8x2
#PBS -l nodes=8:ppn=2:opt240,walltime=8:00:00
#PBS -q route
#PBS -M acaird@umich.edu
#PBS -m ae
#PBS -j oe
cd ${HOME}/HPL/
echo "I ran on: "
cat $PBS_NODEFILE
mpiexec -np 16 ${HOME}/HPL/hpl-oop/bin/OOMP/xhpl
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N hpl-8x2
#PBS -l nodes=8:ppn=2:opt240,walltime=8:00:00
#PBS -q route
#PBS -M acaird@umich.edu
#PBS -m ae
#PBS -j oe
cd ${HOME}/HPL/
echo "I ran on: "
cat $PBS_NODEFILE
mpiexec -np 16 ${HOME}/HPL/hpl-oop/bin/OOMP/xhpl
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N hpl-8x2
#PBS -l nodes=8:ppn=2:opt240,walltime=8:00:00
#PBS -q route
#PBS -M acaird@umich.edu
#PBS -m ae
#PBS -j oe
cd ${HOME}/HPL/
echo "I ran on: "
cat $PBS_NODEFILE
mpiexec -np 16 ${HOME}/HPL/hpl-oop/bin/OOMP/xhpl
```

Submitting, Checking, and Deleting Batch Jobs

- After you create your PBS script, you need to submit it:

```
$ qsub -V hpl.q
542.nyx-login.engin.umich.edu
```

- After you submit your script, you can check on the status of your job:

```
$ qstat -au acaird
nyx-login.engin.umich.edu:
Job ID                  Username Queue    Jobname    SessID NDS   TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ ----- --- ------ ----- - -----
542.nyx-login.engin.    acaird   short    hpl-8x2     18922     8  --    --  08:00 R 00:00
$ checkjob 542
[... lots of output ...]
```

- If you want to delete your job:

```
$ qdel 542
```

Submitting, Checking, and Deleting Batch Jobs

- After you create your PBS script, you need to submit it:

```
$ qsub -V hpl.q
542.nyx-login.engin.umich.edu
```

- After you submit your script, you can check on the status of your job:

```
$ qstat -au acaird
nyx-login.engin.umich.edu:
Job ID                  Username Queue    Jobname    SessID NDS   TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ ----- --- ------ ----- - -----
542.nyx-login.engin.    acaird   short    hpl-8x2     18922     8  --    --  08:00 R 00:00
$ checkjob 542
[... lots of output ...]
```

- If you want to delete your job:

```
$ qdel 542
```

Submitting, Checking, and Deleting Batch Jobs

- After you create your PBS script, you need to submit it:

```
$ qsub -V hpl.q
542.nyx-login.engin.umich.edu
```

- After you submit your script, you can check on the status of your job:

```
$ qstat -au acaird
nyx-login.engin.umich.edu:
Job ID                  Username Queue    Jobname    SessID NDS   TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ ----- --- ------ ----- - -----
542.nyx-login.engin.    acaird   short    hpl-8x2     18922     8  --    --  08:00 R 00:00
$ checkjob 542
[... lots of output ...]
```

- If you want to delete your job:

```
$ qdel 542
```

PBS Email

PBS will send an email at the start and end of your job if you use the `-m` and `-M` options in your PBS script. The email after a job completes successfully looks like:

```
Date: Sun, 30 Apr 2006 12:50:17 -0400
From: adm <adm@nyx-login.engin.umich.edu>
To: "Caird, Andrew J" <acaird@umich.edu>
Subject: PBS JOB 542.nyx-login.engin.umich.edu
---------------------------------------

PBS Job Id: 542.nyx-login.engin.umich.edu
Job Name:   hpl-8x2
Execution terminated
Exit_status=0
resources_used.cput=13:17:26
resources_used.mem=1220672kb
resources_used.vmem=11146704kb
resources_used.walltime=00:49:57
```

Understanding the Scheduler

The scheduler determines what jobs can run, when the can run, and where. There are many factors that go into the scheduler's decision.

- Limits
  - Maximum number jobs eligible for scheduling: 2
  - Maximum number of CPUs in use by one person: depends on queue
  - Maximum number of jobs in the queue at one time: no limit
- Priority
  - Who you are: user and group level priorities
  - How long you've waited: the longer you wait, the higher your priority
  - Your recent usage (fairshare): People with less usage over the past month will have a higher priority than those with a lot of usage

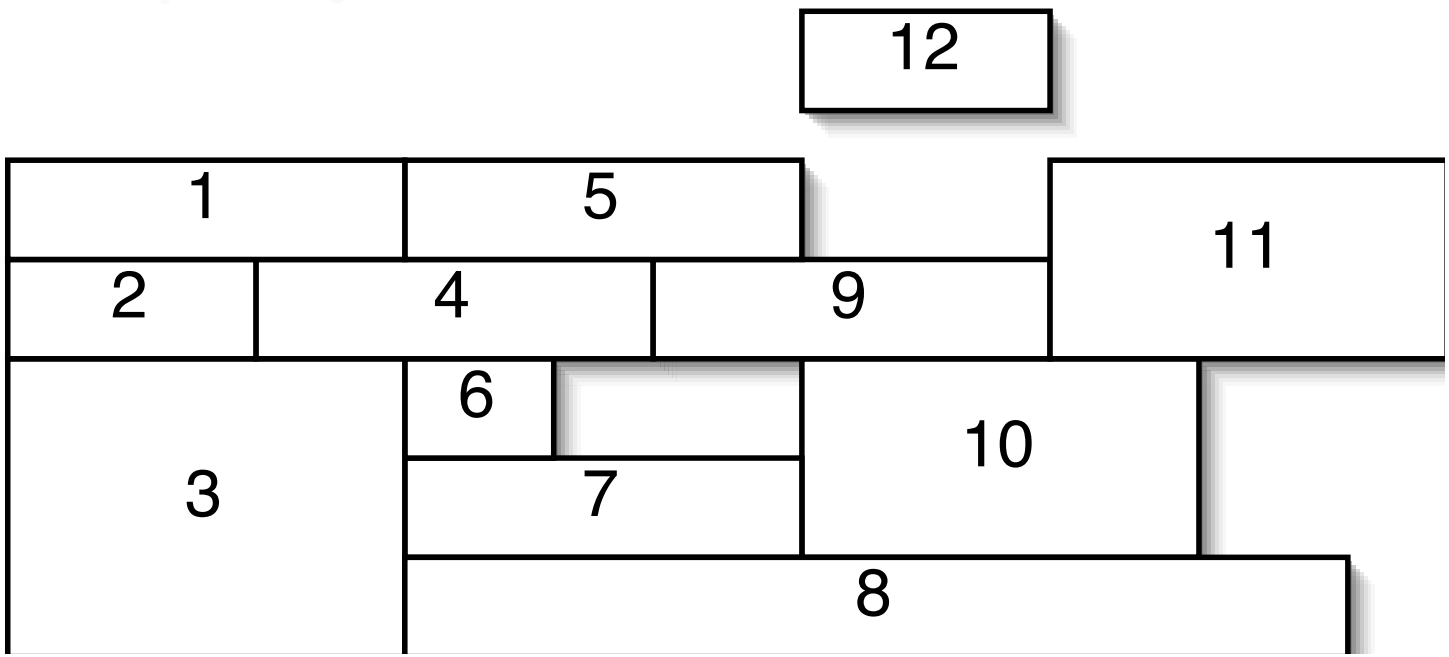
Understanding the Scheduler

The scheduler determines what jobs can run, when the can run, and where. There are many factors that go into the scheduler's decision.

- Limits
  - Maximum number jobs eligible for scheduling: 2
  - Maximum number of CPUs in use by one person: depends on queue
  - Maximum number of jobs in the queue at one time: no limit
- Priority
  - Who you are: user and group level priorities
  - How long you've waited: the longer you wait, the higher your priority
  - Your recent usage (fairshare): People with less usage over the past month will have a higher priority than those with a lot of usage

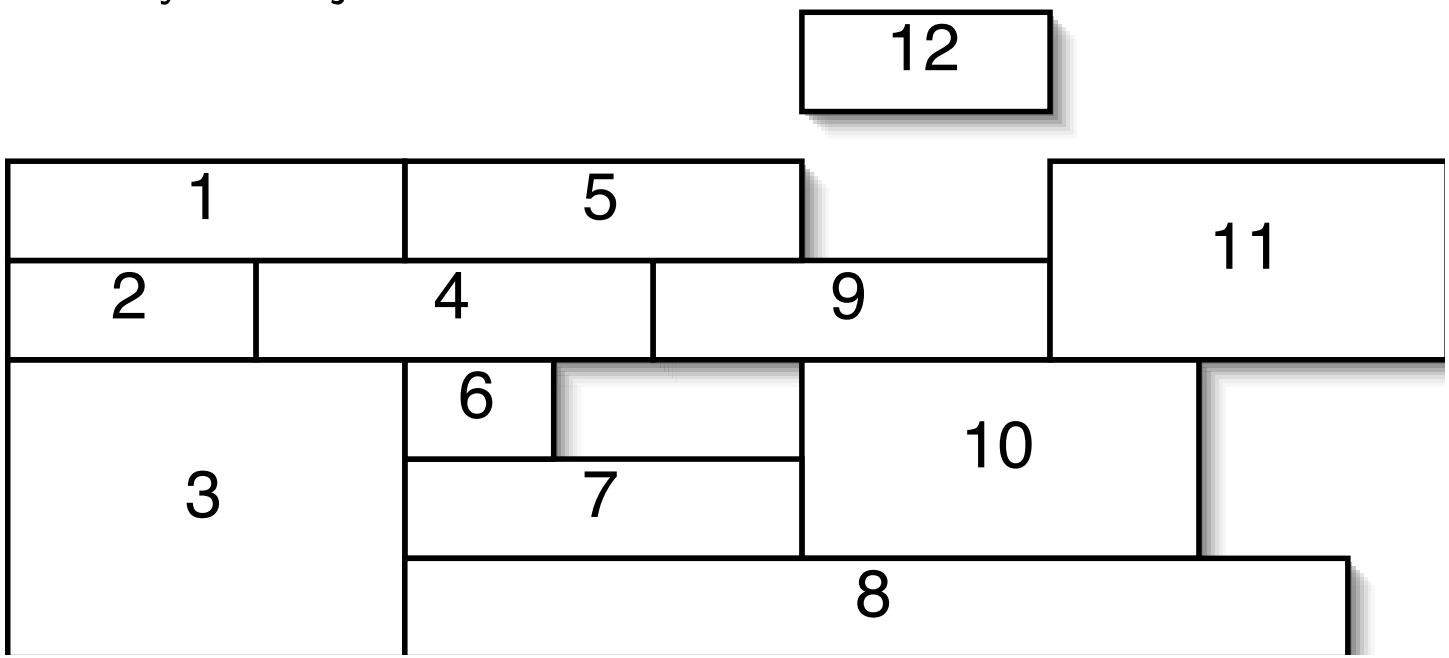
Understanding the Scheduler

- Reservations
  - Advance reservations: holds nodes for users or groups
  - Job reservations: scheduler will reserve nodes for the next several jobs in each queue
- Backfill
  - If the reservations leave holes in the schedule, they may be filled by short jobs that otherwise would have waited.



Understanding the Scheduler

- Reservations
  - Advance reservations: holds nodes for users or groups
  - Job reservations: scheduler will reserve nodes for the next several jobs in each queue
- Backfill
  - If the reservations leave holes in the schedule, they may be filled by short jobs that otherwise would have waited.



Understanding the Scheduler

There are several commands that can give you insight into the scheduler's decisions.

- `showq` — shows the state of the queue at that moment in time, showing the running jobs in order of soonest to finish to longest to finish; the idle jobs in order of priority; and the blocked jobs in the order they were submitted
- `diagnose -p` — shows the factors that go into computing the priority for all of the idle jobs
- `checkjob` *`jobnumber`* — for idle jobs this will show why the job can't start
- `showstart` *`jobnumber`* — this makes a (poor) estimate of when the job will start

Summary

- Resources
  - Lots of CPUs
  - A reasonable amount of software
  - Watch or subscribe to `http://cac.engin.umich.edu` for updates
- Access
  - All access is via the SSH family of commands: `ssh`, `sftp`, `scp`
  - There are lots of clients for these commands for the different platforms
  - There is no graphical access, everything is via the command line

Summary

- Job Submission
  - Every job needs a PBS script file
  - Two most important commands: `qsub -V` and `qstat -au` *uniqname*
- Job Scheduling
  - Scheduling depends on a lot of factors, it is best to submit jobs and let the scheduler optimize for their start.

Summary

- News: `http://cac.engin.umich.edu`
  - RSS feed
  - New of changes, outages, other pertinent piece of information
- Contact: `cac-support@umich.edu`
  - Questions or concerns should be sent here (not to an individual) since this is read by six people. The odds of a quick reply are best this way.
  - We aren't parallel programmers, but we'll do what we can to help.