

Effective Use of CCV Resources

Mark Howison

User Services & Support

This talk...

- ▶ Assumes you have some familiarity with a Unix shell
- ▶ Provides examples and best practices for typical usage of CCV systems
- ▶ Is a condensed form of the documentation available at:

<http://www.brown.edu/ccv/doc>

Overview

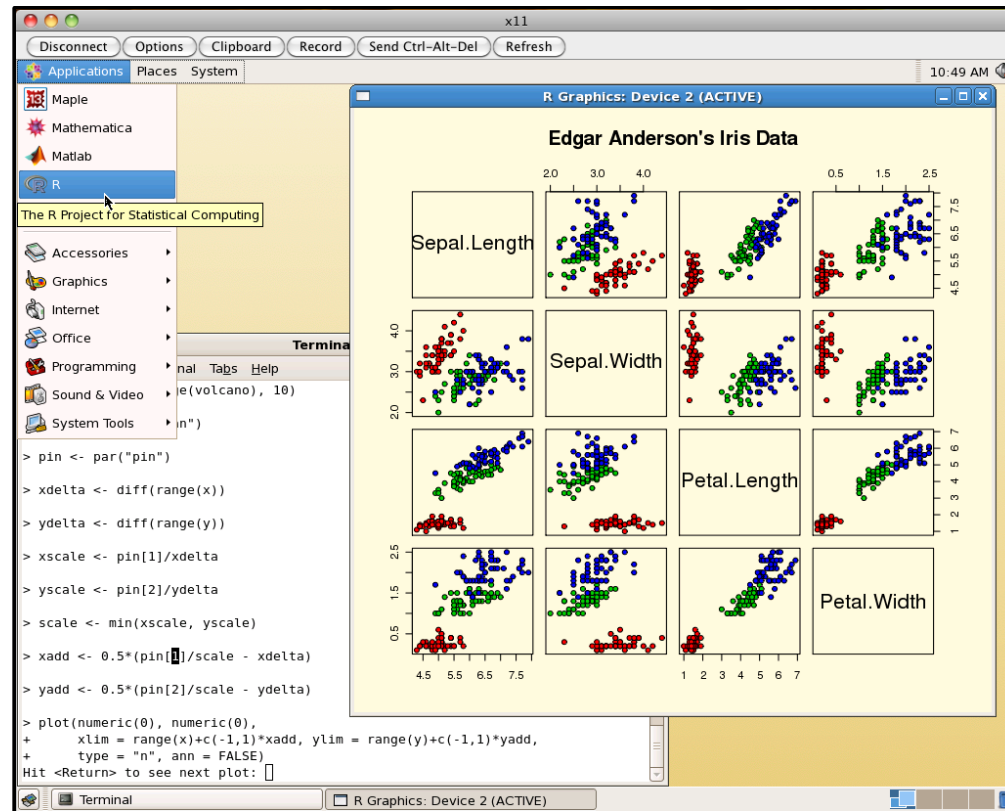
- ▶ Connecting to CCV
- ▶ Transferring files
- ▶ Available software
- ▶ Compiling and linking your own code
- ▶ Running and monitoring jobs

Logging in

- ▶ CCV uses the Secure Shell (SSH) protocol
- ▶ You will need an SSH client
 - Linux / OS X comes with a command-line client
 - Windows: try PuTTY or Cygwin
- ▶ Connect to the “ssh” server
 - Linux / OS X / Cygwin:
`ssh username@ssh.ccv.brown.edu`
 - PuTTY: enter `ssh.ccv.brown.edu` in “Host Name”

Virtual Network Computing

- ▶ OR connect through CCV's Virtual Network Computing client, available here:
 - <http://www.brown.edu/Departments/CCV/vnc>



Transferring files

- ▶ Mount your **RData** directory (`~/data` on Oscar) with Samba to use Windows Explorer or Mac Finder
- ▶ Or from a terminal, use the **scp** command:
 - Copy to Oscar:
`scp username@ssh.ccv.brown.edu:/remote/path /local/path`
 - Copy from Oscar:
`scp /local/path username@ssh.ccv.brown.edu:/remote/path`
- ▶ Or use a GUI “Secure Copy” program
 - e.g. WinSCP, Fugu (Mac)

Available software

- ▶ Software is organized using **Environment Modules**
- ▶ Load a software module with `module load <name>`
- ▶ Loading a module alters your environment, paths, etc:

```
[mhowison@login001 ~]$ module display intel
```

```
-----  
/gpfs/runtime/modulefiles/intel/12.0.4:  
setenv          MKL -L/gpfs/runtime/opt/intel/12.0.4/mkl/lib/intel64  
-lmkl_rt -liomp5 -lpthread  
prepend-path    PATH /gpfs/runtime/opt/intel/12.0.4/bin  
prepend-path    MANPATH /gpfs/runtime/opt/intel/12.0.4/man/en_US  
prepend-path    LD_LIBRARY_PATH /gpfs/runtime/opt/intel/12.0.4/lib/  
intel64:/gpfs/runtime/opt/intel/12.0.4/mkl/lib/intel64  
...
```

```
[mhowison@login001 ~]$ module load intel
```

```
[mhowison@login001 ~]$ which icc  
/gpfs/runtime/opt/intel/12.0.4/bin/icc
```

Available software (Cont'd)

- ▶ View available modules with `module avail`
 - Or search for a specific package with `module avail <package>`
- ▶ View your loaded modules with `module list`
- ▶ Module commands in your `~/.modules` file will automatically execute when you login
 - Add a module to your default list with `echo "module load <name>" >> ~/.modules`
- ▶ If you need software that is not installed, submit a request at <http://www.brown.edu/Departments/CCV/protected/software>

Compiling

- ▶ By default, the GNU compiler suite is loaded
 - `gcc` (C), `g++` (C++), and `gfortran` (Fortran)
- ▶ Portland Group (PGI) compilers are available with `module load pgi/11.4`
 - `pgcc` (C), `pgCC` (C++), and `pgf90` (Fortran)
- ▶ Intel compiler suite (with MKL) is available with `module load intel`
 - `icc` (C), `icpc` (C++), and `ifort` (Fortran)

Compiling (Cont'd)

- ▶ To use MPI with the PGI or Intel compilers, you must swap out the **openmpi** module!

```
module swap openmpi openmpi/1.4.3-pgi
```

or

```
module swap openmpi openmpi/1.4.3-intel
```

- ▶ Otherwise, **mpicc** will point to the wrong compiler

Linking

- ▶ Many modules include environment variable **shortcuts** that contain linking directions, e.g.

FFTW

```
gcc -o fftw-app fftw-app.c $FFTW
```

```
$FFTW = -I/gpfs/runtime/opt/fftw/3.2.2/include  
-L/gpfs/runtime/opt/fftw/3.2.2/lib -lfftw3
```

GotoBLAS

```
gcc -o blas-app blas-app.c $GOTO
```

```
$GOTO = -I/gpfs/runtime/opt/gotoblas2/1.13/include  
-L/gpfs/runtime/opt/gotoblas2/1.13/lib -lgoto2  
-lpthread -lgfortran
```

Running jobs

- ▶ An **interactive** job allows you to:
 - View the output of a program as it runs
 - Interact with a program by typing input, using a GUI, etc.
 - Quickly stop and restart a program, e.g. to test out different parameters or for debugging
 - Run on a system shared by other users
- ▶ A **batch** job allows you to:
 - Submit a script that will run without any intervention
 - Access dedicated resources for your job
 - Run for long periods of time without worrying about other users interfering with your job

Login nodes

- ▶ When you ssh to Oscar, you are placed on either `login001` or `login002`
 - The login nodes are intended for tasks like:
 - writing, compiling, and debugging code
 - transferring and managing files
 - submitting and managing batch or interactive jobs
 - They are a shared resource with many other users concurrently logged in
 - Please don't run large-memory or CPU-intensive programs on these nodes – it disrupts other users!

Batch jobs

- ▶ Specify resource requirements and the commands to run in a file called a **batch script**

```
#!/bin/bash
#PBS -N MyMPIJob
#PBS -l nodes=2:ppn=8
#PBS -l walltime=00:30:00

# execute an MPI program
mpirun -np $PBS_NP mpi_program <args>
```

- ▶ Submit the script to the **queue** with `qsub <script>`

Batch jobs (Cont'd)

- ▶ Some useful Torque options:

```
#PBS -j oe
```

Combine your **stdout** and **stderr** outputs into a single log file (by default called `<jobname>.o<jobid>` in the directory where you submitted the script)

```
#PBS -m abe
```

```
#PBS -M oscar_user@brown.edu
```

Send mail if your job **aborts**, and when it **begins** and **ends**

```
#PBS -V
```

Import your current environment when launching the script on the head node

Batch jobs (Cont'd)

- ▶ Packing 8 serial jobs onto an 8-core Oscar node:

```
#!/bin/bash
#PBS -N MySerialJobs
#PBS -l nodes=1 : ppn=8
#PBS -l walltime=00:30:00

# Run 8 copies of a serial program on 1 node
using the pbsdsh command.
# The PBS_VNODENUM corresponds to the CPU core
that each serial job will use.

pbsdsh serial_program input_file_ $PBS_VNODENUM
```

Available queues on Oscar

▶ Choosing a queue:

- Default is the **dq** queue
- At submit time with `qsub -q <name>`
- In your batch script with `#PBS -q <name>`

dq	220 x (8-core compute nodes)
gpu	42 x (8-core/2-GPU compute nodes)
timeshare	Packs jobs onto large memory nodes (multiple jobs can run on a single node); limit of one node per job
debug	4 Oscar compute nodes available for fast turn-around; time limit is 40 node-minutes

Interactive jobs

- ▶ You can start an interactive job in the **timeshare** queue using the `interact` command

```
interact [-n cores] [-t walltime] [-m memory] [-q queue] [hostname]
```

```
# Default is 1 core, 1:00:00 walltime, 4gb memory, timeshare queue:
```

```
interact
```

```
# Request more cores/time/memory like this:
```

```
interact -n 8 -m 64g -t 4:00:00
```

```
# Request a specific node (e.g. smp8) with:
```

```
interact -n 8 -m 64g -t 4:00:00 smp8
```

```
# Request a different queue with:
```

```
interact -q debug
```

Monitoring jobs

- ▶ See the status of all queues with `allq`
 - Hint: this list can be long, but you can scroll through it with `allq | more`
- ▶ Or, see only a specific queue with `allq <queue>`
- ▶ See a summary of your jobs with `myq`
- ▶ See another users jobs with `myq <username>`
- ▶ Alter a job with `qalter <options> <jobid>`
- ▶ Delete a job with `qdel <jobid>`

Job arrays

- ▶ Torque has a special feature for **parameter sweeps**
 - I.e., you want to run (“sweep”) the same program over a set of varying parameters
- ▶ Use `#PBS -t <range>`
 - Where range is of the form `0-N` or `0,2,4-8` etc.
 - A sub job with the name `<jobname>[i]` is created for each value in the range
 - In each sub job, the `$PBS_ARRAYID` environment variable is set to a different value of the range

Job arrays

- ▶ This job will use 4 nodes (32 cores) to sweep over 32 parameters:

```
#!/bin/bash
#PBS -N MySweep
#PBS -t 0-3
#PBS -l nodes=1:ppn=8
#PBS -l walltime=1:00:00

pbsdsh bash -c '''
    export ID=$((PBS_NUM_PPN*PBS_ARRAYID+PBS_VNODENUM));
    echo "Starting job $ID on $HOSTNAME (CPU $PBS_VNODENUM)";
    serial_program input_file_$ID
'''
```