Intermediate Linux and High Performance Computing on Orchestra

HMS Research Computing

Spring 2016

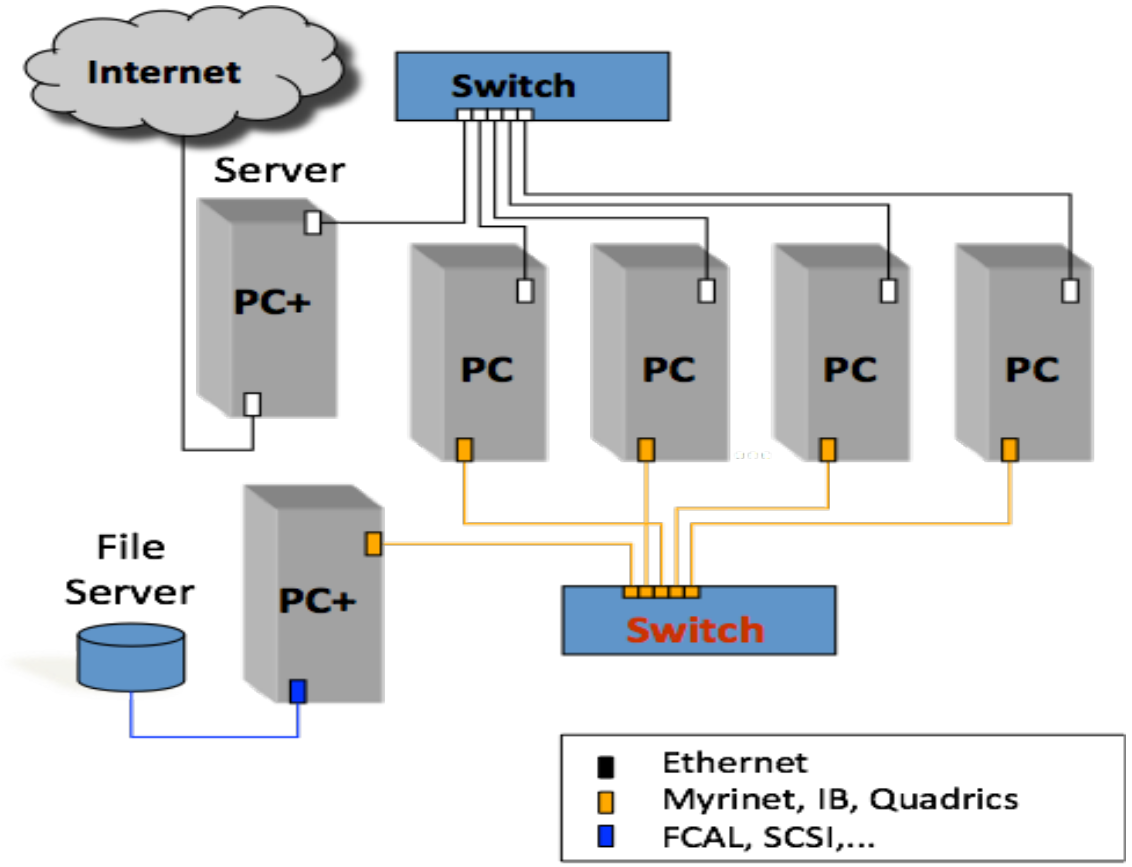
Class Objectives

- Quick review of the Orchestra HPC environment
- Learn new Linux and Orchestra tools
- Work with command line output and error
- More advanced ways to run and manage Orchestra jobs
- Introduction to loops and shell scripting

What is Orchestra?

- Wiki page: https://wiki.med.harvard.edu/Orchestra
- Tech spec:
  - Over 500 compute nodes
  - Over 6500 cores
  - 10GigE interconnection
  - Over 40TB RAM
- CentOS Linux
- LSF scheduler
- Total 25PB storage

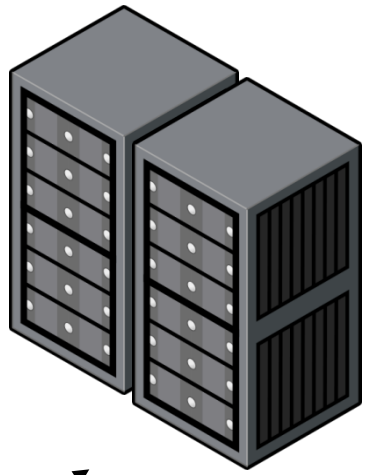
Generic Cluster Architecture



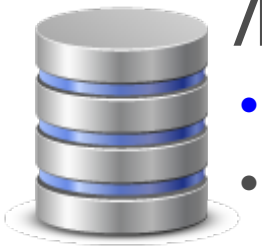
- Login nodes are for data management, job submission, code development, etc.
- Compute nodes are for production runs
- Your storage space is centralized network storage system
- Hundreds users are sharing the resources, so please be nice each other.
- Job scheduler works based on complex algorithms of fair-share, priority management, load balancing, etc.

Orchestra Storage

Orchestra Cluster
- 6500+ cores
- LSF batch system



Your computer



/home
- /home/user_id
- quota: 100GB per user
- Backup: extra copy & snapshots:
- daily to 14 days, weekly up to 60 days

/n/data1, /n/data2, /groups
- /n/data1/institution/dept/lab/your_dir
- quota: expandable
- Backup: extra copy & snapshots:
- daily to 14 days, weekly up to 60 days

“Scratch” storage: Coming Soon

- **/n/scratch2**
- **For storing data only needed temporarily for analyses.**

  - **Lustre** --> high-performance parallel file system
  - More than 1 PB of shared disk space. Each account can use up to 5 TB.
  - No backups! Unused files will automatically deleted on a regular schedule.
  - We are currently doing performance testing. More details to be announced!

Logging Into Orchestra: Mac

- Open a terminal (search “terminal”)

  ssh username@orchestra.med.harvard.edu

- To display graphics back to your desktop (X11 forwarding)

  Install XQuartz (google it) and have it running

  ssh **–X** username@orchestra.med.harvard.edu

Logging Into Orchestra: Windows

- Install PuTTY (google it)
- In box under “Host Name (or IP address)”

  orchestra.med.harvard.edu

- To display graphics back to your desktop (X11 Forwarding):

1. Install Xming (google it) and have it running
2. In PuTTY, under Connection -> SSH -> X11:
3. Check “Enable X11 Forwarding” first and THEN log in

Logging Into Orchestra: Linux

- Open a terminal (search: “terminal”)

  ssh username@orchestra.med.harvard.edu

For graphics (X11 Forwarding)

  ssh **–X** username@orchestra.med.harvard.edu

Checking Storage Usage

- To check your available storage, run “quota”:

  mfk8@loge:~$ quota

  Reports are updated hourly.

Home directory: everyone gets 100 GB.

Group directories, as applies to your account:

/groups
/n/data1
/n/data2

rsync: better/faster data transfer

- To copy or synchronize data between Orchestra directories
  - mfk8@loge:~$rsync -av /home/mfk8/data/. /n/data1/dir/groupdata/.
- To copy from a remote system/desktop to Orchestra For GB+ transfers, use Orchestra's file transfer server: transfer.orchestra.med.harvard.edu
- rsync can resume interrupted transfers!

rsync: from desktop -> Orchestra

1. bash

```
andys-mbp:~ alb15$ rsync -av -e ssh testfile alb15@transfer.orchestra.med.harvard.edu:~/
building file list ... done
testfile

sent 151 bytes  received 42 bytes  386.00 bytes/sec
total size is 26  speedup is 0.13
andys-mbp:~ alb15$
```

Using Software: Environment Modules

- Most “software” on Orchestra is installed as an environment module. Allows for clean, easy loading, including most dependencies, and switching versions.

  $ module avail

  $ module avail seq/

  $ module avail stats/

  $ module avail seq/bowtie/

Loading/Unloading Modules

- Loading modules

  $ module load seq/bowtie/2.0.6

- Which module version is loaded (if at all)?

  $ which bowtie

- See all modules loaded

  $ module list

- Unloading modules

  $ module unload seq/bowtie/2.0.6

- Dump all modules

  $ module purge

Python, R, Perl

- Users manage their own package libraries: get the version you want, when you want it!
- Python: virtual environment allows pip installs

https://wiki.med.harvard.edu/Orchestra/PersonalPythonPackages

- R: set up personal R library for cran, bioconductor

https://wiki.med.harvard.edu/Orchestra/PersonalRPackages

- Perl: local::lib allows cpan, cpanm

https://wiki.med.harvard.edu/Orchestra/PersonalPerlPackages

Retrieving backups fom daily snapshots

- Snapshots kept for up to 60 days under /home and group directories.
- mfk8@loge:~$ cd .snapshot
- mfk8@loge:~$ ls

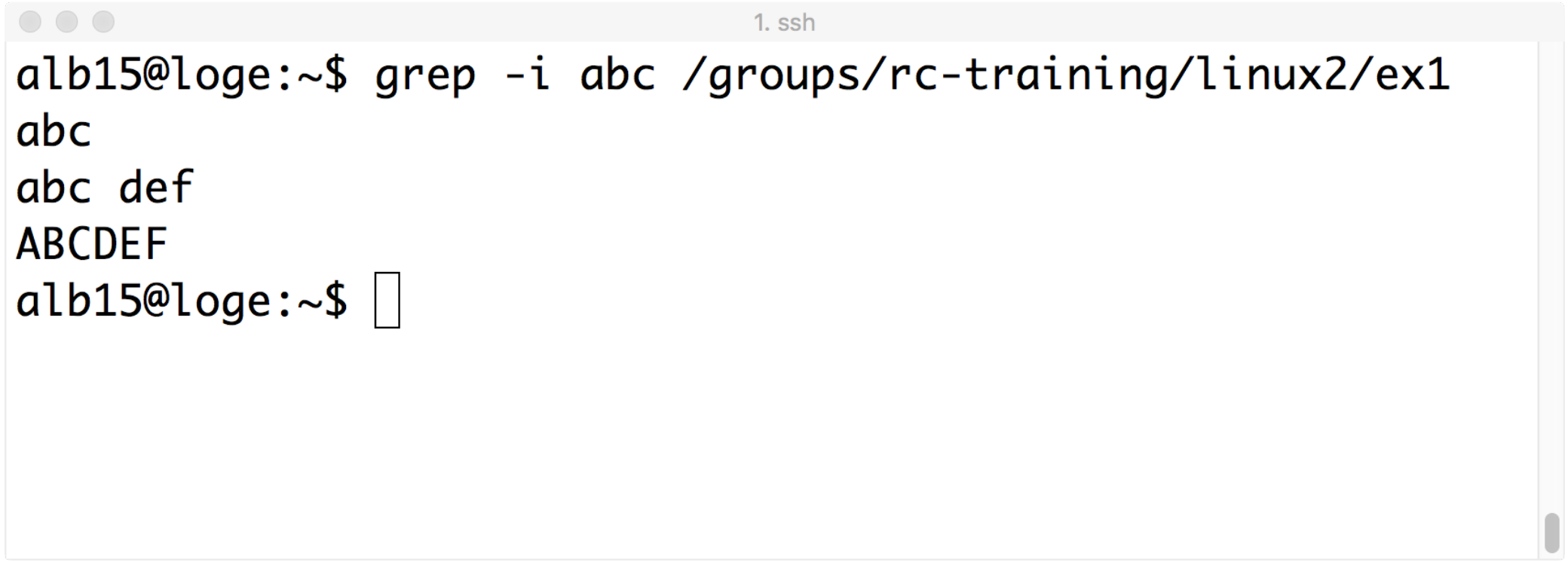
  Orchestra_home_daily_2015-10-02-02-00

  Orchestra_home_daily_2015-10-01-02-00
- mfk8@loge:~$ cd Orchestra_home_daily_2015-10-02-02-00
- mfk8@loge:~$ cp MyRetreivedFile ~

Symbolic Links (aka symlinks)

- A special file type similar to a Windows or Mac “shortcut”.
- Symbolic links can resolve across different filesystems.
  - Make a file or directory and then a symlink to it called myshortcut:
  - mfk8@loge:~$ mkdir work
  - mfk8@loge:~$ ln -s work myshortcut
  - mfk8@loge:~$ ls -l (note the “L” file type for symlink)
- Note: a “hard” link (without the “-s” option) is different.
  - Always use “ln -s” unless you really know what you’re doing!

Text searching with “grep”



The syntax of grep

- "grep" can search files or any text input for patterns line by line
  - Etymology: *g/re/p* (***g****lobally search a* ***r****egular* ***e****xpression* and ***p****rint*)

- Common options:
  - -i ignore case
  - -r search all files in subdirectories recursively
  - -v show everything *except* the specified pattern

- mfk8@loge:~$ grep -v 'pattern' myfile.txt
  - Quotes are needed when patterns include spaces and !@#$% type characters.

Example with grep: try this!

- cd /groups/rc-training/linux2
- cat ex1
- grep abc ex1
- grep -i abc ex1
- grep -v abc ex1
- grep -iv abc ex1
- grep abcd ex1
- grep “abc def” ex1

Wildcard * Pattern Matching

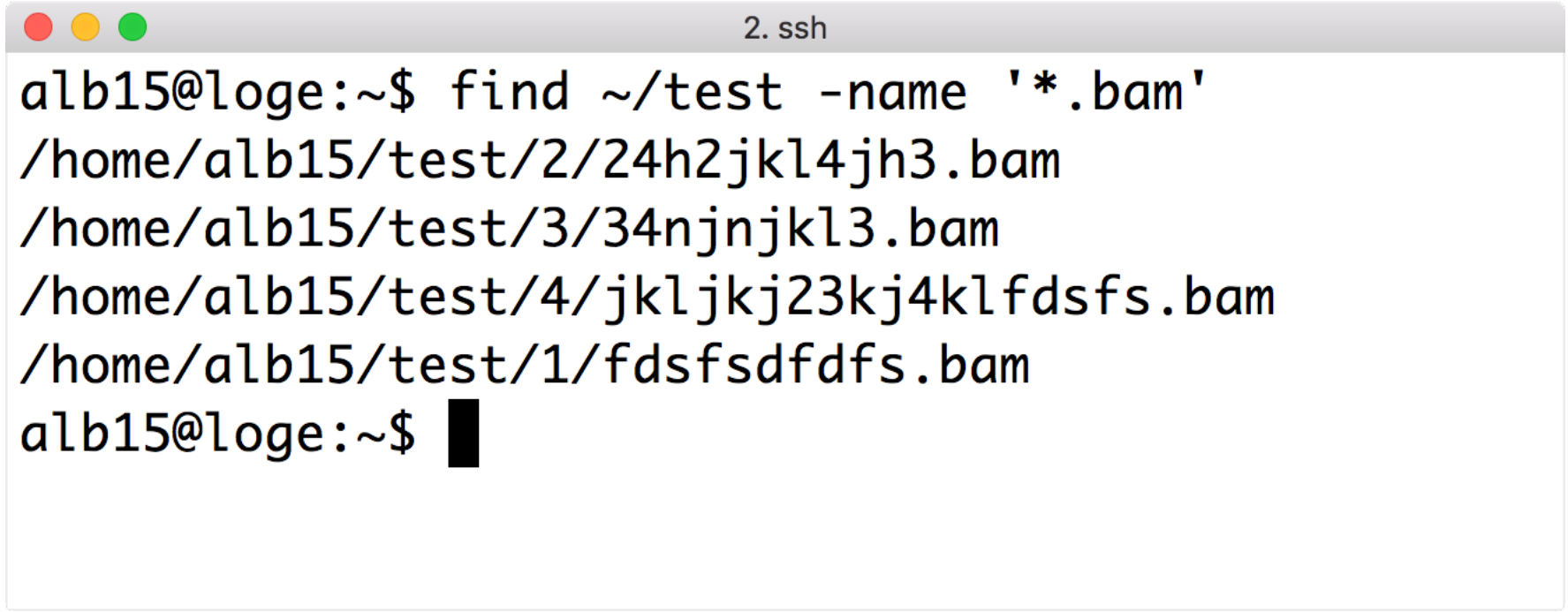
- Useful for copying/removing/etc all files matching a certain pattern
- Example Case:

  To copy “all” files ending in “.fastq”:

  ```
  mfk8@loge:~$ cp  *.fastq  ./NewFastqFolder
  ```

Pattern-matching an get much more granular than this, but we don’t want to overwhelm you yet.

Command line search with “find”



“find” Command Syntax

- Use “find” to search directories based on file attributes:
  - Name: find /tmp -name ‘myfile*.txt’ (match the name of pattern myfile*.txt)
  - Owner: find /tmp -user mfk8 (owned by user mfk8)
  - Group: find /tmp -group mylab (owned by mylab group)
  - Date: find /tmp -ctime 30 (modified in the past 30 days)

  - *(see manual page for other options)*

Using find: -exec option

- Sometimes it's useful to run a command against the results of "find".
- Examples:

  - find /tmp -atime 30 -exec rm {} \; (remove all files older than 30 days)
  - find ~ -atime 30 -exec mv {} ~/archive \; (move old files to an archive directory)
  - find . -name '*.bam' -exec ln -s {} \; (create a symlink for each BAM file)

  - *The {} symbol is a placeholder for the output of find.*
  - *The escaped semi-colon \; is required in this case to finish the command.*

File Permissions

- By default, Orchestra home directories are world-readable!
  - That means all other Orchestra users have read access.
  - Why? This is a standard configuration in Linux/UNIX to allow for file sharing.
  - You can restrict your own file permissions with the “chmod” command:
  - mfk8@loge:~$ chmod o-rwx /home/mfk8

- By default, Orchestra shared directories are restricted to the group.
  - Group directories are owned by a designated group, e.g. “mylab”
  - File permissions are set as 2770 (no access outside of the owning group)
  - If desired, you can request changes from Research Computing.

Fixing permissions in group directories

- If files are not owned or writable to the group in your lab's shared directory, but should be, you can fix **files you are the owner of**.
  - example group = mylab
  - example directory = /n/data1/hms/sysbio/mylab/mydir

- mfk8@loge:~$ cd /n/data1/hms/sysbio/mylab
- mfk8@loge:~$ chgrp -R mylab mydir
- mfk8@loge:~$ find mydir -type d -exec chmod g+rwxs {} \;
- mfk8@loge:~$ find mydir -type f -exec chmod g+rw {} \;

What is Standard Output / Error ?

- Standard Output:
  - The “normal” output of Linux commands and applications.
  - **The bsub “–o” option is for creating an output log file**

- Standard error comes from:
  - bad command syntax
  - incorrect file permissions (can’t read / write a file)
  - missing library files or other application errors
  - **The bsub “–e” option is for creating an error log file**

- By default, both are displayed in the terminal screen.

Redirecting Standard Output

- Redirect (>)
  - Writes output to a new file instead of writing to the terminal.
  - Will overwrite an existing file completely.
- Append (>>)
  - Appends output to the end of a file without overwriting.
- Pipe (|)
  - Redirects standard output to a new command.

Examples: redirecting standard output

- Redirect (>)
  - mfk8@loge:~$ grep abc /groups/rc-training/linux2/ex1 > out.txt
  - mfk8@loge:~$ cat out.txt
- Append (>>)
  - mfk8@loge:~$ find /groups/rc-training/linux2 -name 'ex*' >> out.txt
  - mfk8@loge:~$ cat out.txt
- Pipe (|)
  - mfk8@loge:~$ samtools sort in.bam | view –h > out.bam

“bsub” – submitting a job to Orchestra

```
mfk8@loge:~$ bsub  -q  queue  -W  hr:min   your_job
```

- Required Options (jobs will not dispatch without these):

  -q (queue)

  -W (runtime in hr:min)

Primary shared Job Queues

- ***priority*** if you have just one or two jobs to run
- ***mcore*** if you have multi-core jobs to run.
- ***short*** if your jobs will take less than 12 hours to run.
- ***interactive*** to get an interactive login for up to 12 hours.
- ***mpi*** if you have an MPI parallel job
- ***long*** everything else

- bqueues displays which queues you have access to
- https://wiki.med.harvard.edu/Orchestra/ChoosingAQueue

Setting a job's Runtime Limit

- **-W in hours:minutes (this option is required!)**
- Runtimes are subject to the maximum time permitted per queue
- If your job exceeds your runtime, your job will be killed ☹
- Running many jobs that finish quickly (less than a few minutes) is suboptimal and may result in job suspension. **Contact RC to learn how to batch jobs!**

Example: setting a 2 hour runtime limit to the short queue:

```
mfk8@loge:~$ bsub  -q  short  -W  02:00  ./myseq.sh
```

Multithreading

- A single CPU can execute multiple processes (threads) concurrently
- -n indicates how many cores are requested
- Jobs that are over-efficient (use more cores than reserved) jeopardize the health of a node
- Reserve the same amount of cores in your job and your bsub!
  - The default is 1 core if –n is not specified.

CPU Limit: matching # of cores to your job

- CPU limit = number of seconds the cluster works on your job
  - which is calculated by LSF using: Ncores * Runlimit (-n * -W)
  - The default is 1 core per job unless you specify more in bsub with: -n

- Here's a common error, using "tophat" as an example:

  bsub –q short –W 8:00 tophat –p 8

  tophat requests 8 cores but gets 1 (no –n so the default is 1 core).

  Result: Job killed in 1 hour.

  Should have been: bsub –q short –W 8:00 –n 8 tophat –p 8

Monitoring Jobs

- List info about jobs/their status:

  mfk8$loge:~$ bjobs

  -r (running jobs)

  -p (pending jobs)

  -l (command entered, long form)

- List historical job information

  mfk8$loge:~$ bhist jobid

Job Suspensions

- Jobs in the “long” queue can be suspended by LSF temporarily to allow jobs from the “short” queue (less than 12h) to run.
- If a job is suspended for >50% of the –W runlimit cumulatively, it will be automatically moved to the “no_suspend” queue where it can’t be suspended anymore.
- bstop jobid - manually pause a running job
- bresume jobid - manually resume a paused job

Terminating Jobs

- Terminate a job (jobid given at submission)

  mfk8$loge:~$ bkill jobid

- Terminate multiple jobs

  mfk8$loge:~$ bkill jobid1 jobid2 jobid3

- Terminate all of your jobs (be careful!)

  mfk8$loge:~$ bkill 0

bsub exercise: submit a job and change its state

1. ssh

```
alb15@loge:~$ bsub -q short -W 1:00 sleep 3600
Job <6108102> is submitted to queue <short>.
alb15@loge:~$ bjobs
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
6108102    alb15   PEND  short      loge.orches             sleep 3600 Mar  8 13:01
alb15@loge:~$ bjobs
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
6108102    alb15   RUN   short      loge.orches clarinet002 sleep 3600 Mar  8 13:01
alb15@loge:~$ bstop 6108102
Job <6108102> is being stopped
alb15@loge:~$ bjobs
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
6108102    alb15   USUSP short      loge.orches clarinet002 sleep 3600 Mar  8 13:01
alb15@loge:~$ bresume 6108102
Job <6108102> is being resumed
alb15@loge:~$ bjobs
JOBID      USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
6108102    alb15   RUN   short      loge.orches clarinet002 sleep 3600 Mar  8 13:01
alb15@loge:~$ bkill 6108102
Job <6108102> is being terminated
alb15@loge:~$ bjobs
No unfinished job found
alb15@loge:~$
```

bsub: specify output and error logs

1. ssh

```
alb15@loge:~$ bsub -q short -W 1:00 -o job.output -e job.error "touch /test"
Job <6113404> is submitted to queue <short>.
alb15@loge:~$ cat job.error
touch: cannot touch `/test': Permission denied
alb15@loge:~$
```

Other bsub options

-n 4	*(number of cores. default =1)*
-R “rusage[mem=8000]”	(*memory requested in MB*)
-Is	*(interactive login shell)*
-N	*(notify when job completes)*
-J jobname	*(define a Job Name)*
-a openmpi	*(type of mpi run)*

Reserving Memory

- Most nodes have 90GB memory. Some have more.
- Make a resource request with

  -R “rusage[mem=8000]” (must request memory in MB units)
- Memory multiplies by cores requested, so

  -n 4 -R “rusage[mem=16000]” reserves 64GB memory
- Asking for more memory may cause jobs to pend longer
- TERM_MEMLIMIT errors indicate not enough memory was reserved

Reserving GPUs

Orchestra has < 10 nodes equipped with GPUs. To access them, you must specify the "ngpus" resource with bsub:

-R "rusage[ngpus=1]" (select # of GPUs for the job)

- Orchestra currently has some older "clarinet" nodes with 1 GPU installed, and 1 newer "contrabassoon" node with 8 GPUs installed.
- If you need help verifying your application can work with Orchestra's CUDA or GPU hardware, please contact RC.

MPI

- Can be an efficient way to run batches of jobs which are not "embarrassingly parallel"
- There is a dedicated queue with same-type compute nodes for optimal performance of MPI (800 cores)
- Matlab, Python, Java, R, C++, Fortran have MPI options
- Orchestra implementation: openMPI-1.8.
- Contact RC if you are considering MPI and want advice!

Interactive Jobs on compute nodes – clarify job submission and compute policies for nodes

- The login servers are not designed to handle intensive processes. Using 40% of a CPU results in a process being terminated. It is sometimes better to use an "interactive session." These last up to 12 hours. Start by entering your first job! This will (usually) log you into a "clarinet!"

  mfk8@loge:~$ bsub -Is -q interactive -n 1 bash

  "bsub" is how jobs are submitted to Orchestra LSF

  -Is (capital eye, s) tells Orchestra you want an interactive shell

  -q is the "queue" to enter (interactive)

  -n is the number of cores requested (1) (20 maximum)

  "bash" opens a session using the bash UNIX shell

  mfk8@clarinet001:~/$

Automate commands with a “for” loop

- Repeat one or more commands against a designated list.
- This syntax is for the bash shell. Other shells (tcsh) are different.

- Examples:
  - mfk8@clarinet001:~$ for x in 1 2 3 ; do mkdir $x ; done
  - mfk8@clarinet001:~$ for x in `cat list` ; do cp $x ~/work ; done

Using bsub with a “for” loop

- When you need to submit a bunch of separate jobs systematically

  - mfk8@loge:~$ for i in [input] ; do [your bsub command] ; done”

- For example:
- mfk8@loge:~$ for i in *.bam; do bsub -q short -W 30 "samtools sort -n $i $i.sorted; done"

Shell Scripts: The Basics

```
#! /bin/sh          #always at the top. Defines type of shell to run.
#program with options
tophat -p 4 -o ./mytophatdir1 hg19 file1_1.fastq file1_2.fastq
tophat -p 4 -o ./mytophatdir2 hg19 file2_1.fastq file2_2.fastq

#save as a file such as myshellscript.sh
#To make the script “executable” by default, update permissions:
# chmod u+x myshellscript.sh
```

Submitting a Shell Script as a job

1). Traditional Shell script

```
mfk8@loge:~$ bsub –q mcore –W 12:00 –n 4 –e errfile.txt –o outfile.txt
  –N ./myshellscript.sh
```

2). A script with LSF options already embedded (next slide):

```
mfk8@loge:~$ bsub < myshellscript.sh
```

Creating a Shell Script for job submission

Create a txt file (job.txt) with contents defining the job array as below.

bsub < job.txt

```
#! /bin/sh
#BSUB -q mcore
#BSUB -W 12:00
#BSUB -o %J.out
#BSUB -e %J.err
#BSUB -N
#BSUB -n 4
#BSUB -R “rusage[mem=12000]”
module load seq/tophat/2.1.0 seq/bowtie/2.1.0 seq/samtools/1.2
tophat -p 4 -o ./mytophatdir1 hg19 file1_1.fastq file1_2.fastq
#save as myshellscript2.sh
```

Using Job Arrays in a script

- #BSUB -L /bin/bash
- #BSUB -n 1
- #BSUB -q short
- #BSUB -W 00:10
- #BSUB -J copytest[1-10]
- #BSUB -o copytest.%I.out
- job_index=$LSB_JOBINDEX
- time cp file$job_index.zip fileout$job_index.zip

Job Dependencies

- When you need behavior like "run job B only if job A finishes"
- bsub –w dependency_expression

- See bsub manual page for full details. Examples:
- bsub –w done (jobid)
- bsub –w exited (jobid)
- bsub –w started (jobid)

More Exercises

- How many jobs do I have running on Orchestra?
  - mfk8@loge:~$ bjobs | wc -l
  - ("wc -l" command counts # of lines)
- How many jobs in the "RUN" state do I have running on Orchestra?
  - mfk8@loge:~$ bjobs | grep RUN | wc -l
- How many total jobs are running on Orchestra?
  - mfk8@loge:~$ bjobs -u all | wc -l

Installing Software: Binary Example

- mfk8@loge:~$ bsub -Is -q interactive bash
- mfk8@clarinet001:~$ wget http://path/to/binary/mysoftware.tar.gz
- mfk8@clarinet001:~$ tar -zxvf mysoftware.tar.gz
- mfk8@clarinet001:~$ ls mysoftware/bin

Installing Software: Source

- mfk8@loge:~$ bsub -Is -q interactive bash
- mfk8@clarinet001:~$ module load dev/compiler/gcc-4.8.5
- mfk8@clarinet001:~$ wget http://path/to/source/mysoftware.tar.gz
- mfk8@clarinet001:~$ tar -zxvf mysoftware.tar.gz
- mfk8@clarinet001:~$ cd mysoftware
- mfk8@clarinet001:~$ less README
- mfk8@clarinet001:~$ ./configure --prefix=/home/mfk8/software
- mfk8@clarinet001:~$ make
- mfk8@clarinet001:~$ make install

Customizing your account's environment

- Command Aliases: create your own commands! Examples:
  - alias ll='ls -la'
  - alias h=history
- Change your default umask:
  - umask 0002
- Alter your default command path. Add a custom "bin" directory:
  - export PATH=$PATH:/home/user/bin

- Add any of these commands to your account's profile (for bash):
  - ~/.bash_profile (this file is read once upon login)
  - ~/.bashrc (this file is read at every invokation of your shell)

For more direction:

- https://wiki.med.harvard.edu/Orchestra/NewUserGuide
- http://rc.hms.harvard.edu
- rchelp@hms.harvard.edu
- Office Hours: Wednesdays 1-3p Gordon Hall 500
- Class survey:

  http://hmsrc.me/interhpc2016-survey1