Albert Einstein College of Medicine
OF YESHIVA UNIVERSITY

Science at the heart of medicine

High Performance Computing Facility
Albert Einstein College of Medicine
Jack and Pearl Resnick Campus
1300 Morris Park Ave
Bronx, NY 10461
hpc-sysadmin@einstein.yu.edu

Using the Einstein High Performance Computing Cluster

April 2014

Contents

This document assumes understanding of *How to Access the High Performance Computing Cluster*.

1 Login/submit nodes

Using the Einstein HPC Portal (`https://hpcportal.einstein.yu.edu`), you can provision a login/submit node for your own use. This node will be a 1-CPU virtual machine with 512MB memory designed for execution of basic `bash` commands and submission of jobs to the cluster using the `qsub` command. The status of your login node should be visible through the HPC Portal and will look like Figure 1.

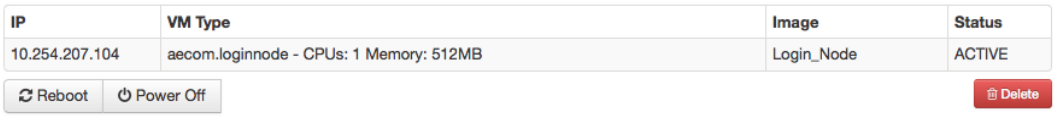

Figure 1: Example active login node. Note that the IP address is unique to your node.

1.1 Accessing your login node

In order to access your login node, you will need to use a Secure Shell (SSH) connection. If you are connecting from off-campus, you will need to establish a connection using the Einstein SSH gateway or VPN (Please see *How to Access the High Performance Computing Cluster* for more information).

Linux and Mac OS X users

Linux and Mac OS X users should simply open the terminal application and enter the command: `ssh username@ipaddress` where `username` is your YUAD username and `ipaddress` is the IP address is the IP address of your login node reported in the HPC portal login node console (see Figure 1). The shell connection will prompt you for your YUAD password – enter your password and hit the return key.

Windows users

Windows users can use an SSH client application like PuTTY (see: `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`). Alternatively an emulated environment (Cygwin) or a virtual machine (e.g. VirtualBox can be used to connect in the same manner as in a Linux environment. The PuTTY suite of tools allows you to perform SSH tasks like secure shell, secure file transfer and generating SSH keys. Cygwin and Virtual Linux machines provide more functionality as they provide Windows users with the same tools as are available to the Linux platform.

PuTTY is the fastest way to get started. Figure 2 shows the interface that PuTTY presents. Enter the IP address (see your HPC portal page, example in Figure 1) into the "Host Name (or IP address)" field, then click connect. The terminal window will prompt you for your username. If you click on the "Data" category, you can enter your username, then return to the session category, give the session a name (Saved Sessions). Then click "Save". This will preserve your information for your next session.

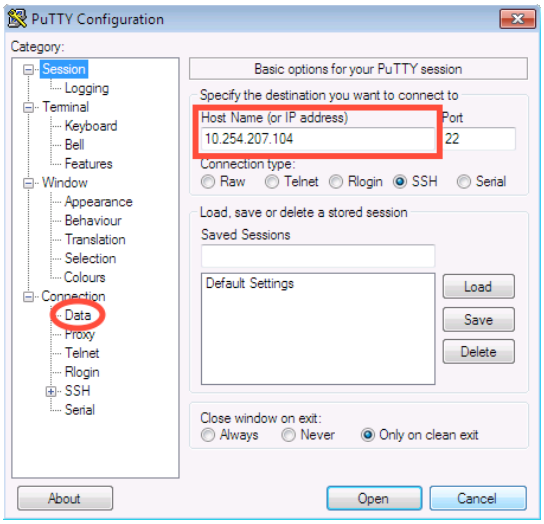

Figure 2: The PuTTY connection dialog, indicating the IP address field and "Data" category.

2 Modules

The Einstein HPC cluster utilizes the Environment Modules package `http://modules.sf.net` for software management. "Modules" provides for dynamic modification of the user's environment by module files. This ensures that the specific version of the software you will be using and all of its dependencies are correctly configured with minimal setup by the user.

2.1 Introduction to module commands

In order to see the full list of available modules, use the `module avail` command.

```
hpcuser@aecom-2e9498b$ module avail

----------------------------- /public/modulefiles ------------------------------
annovar/2014.nov.12
argtable/2-13/gcc.4.4.7
atlas/3.11.30/gcc.4.4.7
bam-readcount/0.7.4/cmake.3.1.3
bamtools/2.3.0/cmake.3.1.3
bamUtil/1.0.13/gcc.4.4.7
bcftools/1.2/gcc.4.4.7
...
```

You may see the list of all versions of a particular module by adding the module's name to the `module avail` command.

```
hpcuser@aecom-2e9498b$ module avail java

----------------------------- /public/modulefiles ------------------------------
java/1.7.0_67 java/1.8.0_20
```

Inspect individual modules using the `module show` command.

```
hpcuser@aecom-2e9498b$ module show GenomeAnalysisTK/3.3-0/java.1.7.0_67
-------------------------------------------------------------------
/public/modulefiles/GenomeAnalysisTK/3.3-0/java.1.7.0_67:

module-whatis  Sets up your environment to use GenomeAnalysisTK-3.3-0 with java-1.7.0_67
module  load java/1.7.0_67
conflict  GenomeAnalysisTK
prepend-path  CLASSPATH /public/apps/GenomeAnalysisTK/3.3-0/java.1.7.0_67
prepend-path  PATH /public/apps/GenomeAnalysisTK/3.3-0/java.1.7.0_67
-------------------------------------------------------------------
```

Note that loading the `GenomeAnalysisTK/3.3-0/java.1.7.0_67` module will automatically load the `java` dependency (version `1.7.0_67`), will conflict with any other version of GenomeAnalysisTK, and will prepend your `CLASSPATH` and `PATH` environment variables with sensible values.

Here is an example of loading a module, using the `module load` command. First we inspect the module (the BWA reference-based aligner) using `module show`.

```
hpcuser@aecom-2e9498b$ module show bwa/0.7.10/gcc.4.4.7
-------------------------------------------------------------------
/public/modulefiles/bwa/0.7.10/gcc.4.4.7:

module-whatis  Sets up your environment to use bwa-0.7.10
conflict  bwa
prepend-path  PATH /public/apps/bwa/0.7.10/gcc.4.4.7/bin
-------------------------------------------------------------------

hpcuser@aecom-2e9498b$ module load bwa/0.7.10/gcc.4.4.7
hpcuser@aecom-2e9498b$ which bwa
/public/apps/bwa/0.7.10/gcc.4.4.7/bin/bwa
hpcuser@aecom-2e9498b$ bwa

Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.10-r789
Contact: Heng Li <lh3@sanger.ac.uk>

Usage:   bwa <command> [options]
...
```

Notice that the BWA module file prepends our `PATH` with the path to the `bwa` binary. After loading the module, the correct version of `bwa` is simply accessed by typing the command.

To get a list of currently loaded modules, use `module list`, to unload a particular module, use `module unload modulename`, to unload all modules, use `module purge`, and to get more help type `module help`.

It is good practice to include the entire version string in the `module load` command, especially when including this directive in an analysis script. Documenting the precise software version is a first step to keeping your research reproducible.

3 Einstein High Performance Computing Cluster

The Einstein HPC Cluster system consists of virtual machines acting as personal login/submit nodes which may submit jobs to a centralized scheduler node. The scheduler node chooses the most optimal arrangement of job executions on the cluster, watches the execution of your job and can notify of the job's status and upon completion.

Currently, the Einstein HPC cluster consists of (at least) 16 20-core compute nodes with 128 GB RAM each. Each node has access to your home and shared directories.

3.1 Network access and the Data Mover node

As a protocol for HPC security, login nodes do not have network access outside the Einstein network and the compute nodes do not have network access. In order to facilitate installation of new software or to download external data sets, there is a "data mover" node which you will need to log into in order to access the external network.

If you have specialized requests for network access or require assistance bringing large data sets to the cluster, please contact hpc-sysadmin@einstein.yu.edu.

3.2 Univa Grid Engine

The Einstein cluster utilizes the Univa Grid Engine (UGE) as a scheduler. UGE is a fork of the Sun Grid Engine.

As a scheduler, UGE is designed to take requests for job scripts and to schedule them for execution on the remote compute nodes. These requests are made using the UGE `qsub` command. For complete documentation on using the `qsub` command, please see the manual page (`man qsub`).

Requesting resources for your job

Before describing the mechanism for submitting jobs to the cluster, it is important to have some understanding of the resources that you will be consuming on the remote compute nodes. There are three primary resources that you are requesting access to when you submit a job to the cluster: processors, memory and time. If you are able to estimate the amount of each of these resources properly – before you submit the job – you will ensure that your job runs efficiently and without interruption, and the scheduler will be able to determine how to interleave your request with other user's requests in the most efficient way.

Here is some discussion about "requestable" resources:

Processors The number of processors that your job will be using needs to be conveyed to the scheduler. In the Einstein cluster, this is by default 1 CPU, unless you specify a particular parallel environment in which to execute. For normal (non-MPI) multi-threaded (SMP; Symmetric Multi-Processing), this is the `smp` parallel environment, which is set to allow a maximum of 20 slots (equivalent to 20 processors) per job: equal to the number of cores on each standard compute node.

Memory A hard limit for the amount of accessible memory for the job is enforced by the scheduler. By default, jobs are granted 6GB; use the `h_vmem` attribute if you will require more. When using the `smp` or `mpi` parallel environments, the total `h_vmem` is calculated by multiplying the number of slots times the `h_vmem` request (e.g. `-pe smp 20 -l h_vmem=5g` requests 20 SMP cores and 100 GB of RAM). Note that the first time you are running an analysis, you may need to perform some testing to determine how much memory to request.

Time The length of time that you expect you job to run for should be conveyed to the scheduler for efficient scheduling. Jobs that have not completed by the end of this interval will be terminated. Use the `h_rt` attribute to specify run time (e.g. `-l h_rt=00:30:00` for 30 minutes). Currently the default run time is infinite; keep in mind that this may change in the future, and it is good practice to enter a sensible value to help the scheduler plan efficiently.

Processors and memory are set to be "consumable" on our cluster; that is, a single node only has a finite amount of these resources to offer and they will be allocated for the jobs that request them.

3.3 `qsub`

The `qsub` command is the primary UGE command for submitting jobs to the cluster. It can be used to submit single- and multiple-node jobs. Single-node jobs include single-threaded binaries or scripts, multi-threaded jobs (SMP), and task arrays. Multiple-node jobs are generally MPI (Message Passing Interface) jobs where the same processing activity is managed over several nodes simultaneously. MPI jobs are beyond the scope of this document.

There are a number of options to the `qsub` command that are relevant to its use. What follows is a list of some of the most useful arguments for submitting typical UGE grid jobs.

`-b {y|n}` explicitly specify that the following command is a binary (`yes`) or a UGE submission script (`yo`).

`-cwd` execute the command or script in the current working directory.

`-hold_jid <job list>` wait to execute this job until all of the jobs in `<job list>` are complete. `<job list>` is a comma-separated list of UGE job ID numbers or names of grid engine jobs. In the case of a name, it will wait for all jobs with the specified name that are running at the time of submission.

-b {y|n} Join the STDOUT and STDERR of the job into a single output file. Yes or no; default no.

-l resource=value,... set the specified resource request to value. For example, `-l h_vmem=20g` requests that the job be allocated 20 GB of memory per slot (CPU).

-m b|e|a|s|n,... set the conditions under which emails will be sent. For example, `-m ea` will email at the **e**nd or if the job is **a**borted.

-M user@host the email where notifications are sent.

-N jobname assign a name to your job.

-o /path/to/out | -e /path/to/err explicitly define a path to the STDOUT (`-o`)

-pe <parallel environment> <n> For SMP (multi-threaded) processes, use the `smp` parallel environment. `smp` offers 20 slots (CPU equivalents) maximum (see `qconf -sp smp`). For example, `-pe smp 20` would request 20 CPUs for the job. and the STDERR (`-e`) output files.

-S /bin/bash explicitly define the shell that UGE uses to interpret your commands/script. This should be `bash` by default.

-V export your environment variables into the compute host's shell. This is useful in combination with `-b y`, however; you should use modules to load your environment in your analysis scripts.

Anatomy of a `qsub` script

```
1  #!/bin/bash
2  #
3  # example.sh
4  #
5  #$ -cwd
6  #$ -j y
7  #$ -N example
8  #$ -S /bin/bash
9  #$ -l h_vmem=1g
10 #$ -l h_rt=00:05:00
11
12 hostname
13 sleep 60
14 echo 'finsihed sleeping'
```

Listing 1: Example `qsub` script, `example.sh`

The example qsub script in Listing 1 is a simple submission script that simply sleeps and uses `echo` to print to the STDOUT. Line 1 contains a special notation called the *shebang* (`#!`). The shebang is a character sequence that tells the interpreter reading this file which language to use to interpret it. It should always be the first line in an interpreted executable (bash, Perl, Python, etc.) In this case, we are using the `bash` shell (by default, located at `/bin/bash`). Lines 2-4 begin with a hash/pound symbol (`#`), which indicates to the interpreter that these lines contain a comment. Lines 5-10 contain `qsub` directives, which are indicated by the #$ notation at the beginning of the line. `qsub` interprets these values as if the parameters were entered on the command line. Beginning at line 12, the lines without # symbols represent commands to be executed. The first command prints to STDOUT the name of the compute host that the job is sent to, line 13 "sleeps" for 60 seconds, and line 14 prints the string 'finished sleeping' to the STDOUT. If the contents of the file are saved in a file called `example.sh`, the job may be submitted to the cluster using the command `qsub example.sh`. Note that in lines 10 & 11, this script explicitly requests a memory limit (1 GB) and a max runtime (5 minutes) but not a number of cores, which will be set to 1 by default. Here is an example terminal session where the job is submitted to the cluster using `qsub` and the status of the job is monitored using `qstat`.

```
hpcuser@aecom-2e9498b$ qsub example.sh
```

```
Your job 1003 ("example") has been submitted
hpcuser@aecom-2e9498b$ qstat
job-ID prior   name    user    state submit/start at     queue    jclass slots ja-task-ID
-----------------------------------------------------------------------------------------
  1003 0.55500 example hpcuser     r 04/03/2015 15:28:04 all.q@n0            1
```

In this example, the job is submitted to the cluster with the name "example" and is assigned the job ID: 1003. The "state" column indicates that the job is running (`r`). The state column might indicate that there is some other status issue, for example queued and waiting (`qw`) or that there was an error (`E`). The "slots" column indicates the number of cores the job is occupying on the node. The output of the job will appear in the STDOUT file (recall that STDOUT and STDERR were joined by `-j y`.

```
hpcuser@aecom-2e9498b$ cat example.o1003
n0
finished sleeping
```

The contents of the file show the node's name and our completion text. It is possible to inspect the status of a job while it is running using the `qstat` command.

```
hpcuser@aecom-2e9498b$ qsub example.sh
Your job 1004 ("example") has been submitted
hpcuser@aecom-2e9498b$ qstat -j example
==============================================================
job_number:                 1004
jclass:                     NONE
exec_file:                  job_scripts/1004
submission_time:            04/03/2015 15:39:00.066
owner:                      hpcuser
uid:                        109283
group:                      Domain
gid:                        100002
sge_o_home:                 /home/hpcuser
sge_o_log_name:             hpcuser
sge_o_path:                 /uge/8.2.0/bin/lx-amd64:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/
sge_o_shell:                /bin/bash
sge_o_workdir:              /home/hpcuser
sge_o_host:                 aecom-2e9498b
account:                    sge
cwd:                        /home/hpcuser
merge:                      y
hard resource_list:         h_vmem=1g,h_rt=300
mail_list:                  hpcuser@aecom-2e9498b.scm
notify:                     FALSE
job_name:                   example
jobshare:                   0
shell_list:                 NONE:/bin/bash
env_list:
script_file:                example.sh
department:                 defaultdepartment
binding:                    NONE
```

```
mbind:                        NONE
submit_cmd:                   qsub example.sh
granted_license       1:
usage                 1:      cpu=00:00:00, mem=0.00000 GBs, io=0.00000, vmem=N/A, maxvmem=N/A
scheduling info:              (Collecting of scheduler job information is turned off)
```

SMP jobs

For SMP processing, where the software that you are using specifically allows for the concurrent use of multiple threads on the same node, you must add the parallel environment directive to the submission request. For example, if you run a software package using 10 threads, add the directive `-pe smp 10` to your submission request. Remember, you must tell the software to use 10 threads as well!

Task array jobs

UGE allows for a particular use case called a task array (or parameter sweep). This allows you to program a single script that can can be submitted once but then operate on a range of input data.

```
1  #!/bin/bash
2  #
3  # task.sh
4  #
5  #$ -cwd
6  #$ -j y
7  #$ -N task-example
8  #$ -S /bin/bash
9  #$ -l h_vmem=1g
10 #$ -l h_rt=00:05:00
11 #$ -t 1-5
12
13 echo This is task $SGE_TASK_ID
14 echo ${JOB_ID}-${JOB_NAME}-${SGE_TASK_ID} >> task_report.txt
```

Listing 2: Example task array, `task.sh`

Listing 2 shows an example task array. Line 11 shows the request to run a task array, in this case using 5 tasks (1-5). The script uses `echo` to print the string "`This is task $SGE_TASK_ID`". `$SGE_TASK_ID` is a special variable that gets set by the scheduler that indicates to the script which task is currently executing. In this case, we simply output our task ID to the STDOUT, but this variable can be used to access input files using a numbered naming convention using the same script. For example the files `input.1.txt` and `input.2.txt` could be accessed using `input.${SGE_TASK_ID}.txt` and a single `qsub` submission using `-t 1-2`.

Line 14 prints the string `${JOB_ID}-${JOB_NAME}-${SGE_TASK_ID}` to the end of the file `task_report.txt` as the jobs complete. Note that it is not generally advisable for tasks to write to the same file as you can not predict the order or whether or not the previous write is finished on some file system types. The output of running this job should look like this:

```
hpcuser@aecom-2e9498b$ ls -l task*
---------- 1 hpcuser Domain Users  15 Apr  3 15:54 task-example.o1007.1
---------- 1 hpcuser Domain Users  15 Apr  3 15:54 task-example.o1007.2
---------- 1 hpcuser Domain Users  15 Apr  3 15:54 task-example.o1007.3
---------- 1 hpcuser Domain Users  15 Apr  3 15:54 task-example.o1007.4
---------- 1 hpcuser Domain Users  15 Apr  3 15:54 task-example.o1007.5
```

```
---------- 1 hpcuser Domain Users 100 Apr  3 15:54 task_report.txt
-rw-r--r-- 1 hpcuser Domain Users 231 Apr  3 15:54 task.sh
hpcuser@aecom-2e9498b$ cat task-example.o1007.1
This is task 1
hpcuser@aecom-2e9498b$ cat task_report.txt
1007-task-example-2
1007-task-example-1
1007-task-example-3
1007-task-example-4
1007-task-example-5
```

Note that task-array job output file names get a suffix of the task array id.

4 Storage

4.1 Shared folders

A symbolic link to your shared folders is available in /<username>-shared. You home directory and shared folders are automatically mounted on the cluster and will be available to your compute jobs. For more information about mounting this storage on your personal computer, please see *How to Access the High Performance Computing Cluster*.

4.2 High speed scratch space

When submitting a job to a compute node, you have access to high-speed (non-networked) local disk storage (2 TB) for temporary use. The folder /scratch/<username> can be used as scratch space by simply writing files there. Your script must clean up and copy any results files to your home directory or shared folders prior to completion, as the folder is deleted upon completion of the UGE job.

5 Getting help

E-mail hpc-sysadmin@einstein.yu.edu for further assistance.