Auburn University
High Performance and Parallel Computing

Hopper Cluster Training I: Fundamentals

FALL 2018

LAB EXERCISES

Prerequisites

- Valid Hopper HPC account
- Laptop connected to the AU wireless network
- Secure shell client, with connection settings for Hopper:
    - Linux and Mac OS users use your favorite terminal app
    - PuTTY (putty.org) is recommended for Window users

Reference

- Hopper User Guide: http://aub.ie/hug
- Lynda.com

Lab Overview

- Getting Started & Basic Linux Commands
- Research Software
- Job Submission
- Job Scheduling
- Cluster Best Practices \ Tips & Tricks

I. Getting Started & Basic Linux Commands

Reminder: command syntax is:

```
$ <command> <required> [optional]
```

The dollar sign ($) is a symbol for the command prompt, which is a way for Linux to tell you it is ready to accept a command. You do not need to type the first $ you see for each command. Wherever you see $, you should enter the command that immediately follows it.

Windows users can use a terminal emulation program like SecureCRT or PuTTY. To connect, you will create a new server profile for hopper.auburn.edu, using your normal AU username and password for the login credentials.

If you are using a terminal program, you can simply type:

```
$ ssh <auburn userid>@hopper.auburn.edu
```

Issue the following commands and take note of what you see on the screen, the output…

```
$ pwd
```

To find out more about any command, including the one that you just ran (pwd) you can refer to the Linux manual pages using the man command.

```
$ man pwd
```

As seen there, pwd stands for “present working directory.” When you run it, you should see something like /home/username on the screen, because that is the folder\directory in which you are currently working.

There are some tricks to navigating the man pages…

Linux Page Navigation (man, more, less)

[DOWN ARROW] To scroll down a line, hit the down arrow
[UP ARROW] To up down a line, hit the up arrow.
[SPACE] To scroll down a page, hit the space bar.
[CTRL-U] To scroll up a page, hit ctrl-u
[Q] To exit, type q
[SHIFT-G] Scroll to end
[0][G] Scroll to top

```
$ man ls
```

Notice there is more information than will fit on one screen. Use the arrow keys and shortcuts to move around in the manual page.

Now let's move around in the Linux file system…

```
$ cd
$ pwd
$ cd /tmp
$ pwd
$ cd ~
$ pwd
```

cd stands for change directory (folder). If you issue cd without anything else, it does nothing. If you give it a path (like /tmp), it will take you to a different directory in the file system.

The last command uses the special character "~" (tilde) which Linux translates to your home directory. You should see the path to your home directory on the screen after issuing the last command in this section.

So, now you have practiced moving around the file system and finding where you are.

Enter the following commands …

```
$ ls
$ mkdir training
$ ls -al
$ cd training
$ pwd
$ echo “hello world” > hello.txt
$ ls -al
$ cat hello.txt
```

In this section, you created and accessed a new directory, created a file with redirection (>) and viewed its contents.

```
$ echo “hello again” > hello2.txt
$ ls -al
$ cat hello2.txt
$ echo “hello world” >> hello.txt
$ cat hello.txt
$ echo “hello world” > hello.txt
$ cat hello.txt
$ ls -al > files.txt
$ cat files.txt
```

Here, we created yet another file with different content using output redirection (>). Then, we appended to the file using the append operation (>>).
Next, we overwrote our changes by going back to the single > operator which demonstrates that output redirection can be destructive!

Finally, we redirected the output of our “ls -al” command to a file. This demonstrates that we can create files from programs and commands we run.

Let’s take a look at who we are and what groups we are members of on the system.

```
$ id
```

Here you can see your username and groups along with the numeric IDs that Linux assigns to each.

Now let's look at our files…

```
$ ls -al
$ rm hello 2.txt
$ ls -al
$ groups
$ chgrp research hello.txt
$ ls -al
$ chmod 750 hello.txt
$ ls -al
$ ls -al hello.txt
```

This group of commands used the rm command to delete a file. Then we used chmod and chgrp to change the file ownerships and permissions.

Look back at the output of these commands and pay close attention to the changes when "ls -al" is run. What changes do you see? What do they mean?

```
$ cd ..
$ pwd
$ ls -al
$ cd training
$ pwd
$ ls -al .
$ ls -al ..
$ cd .
$ pwd
```

Here we perform a quick demonstration of the "." and ".." special operators.

Special File and Directory Characters

Double dots (..) tells Linux to look back into the directory "above" our current location.
A single dot (.) tells Linux we are talking about the current directory.

Files that begin with "." are hidden files in Linux.

As you can see, “cd ..” takes us back up to our home directory, while cd “.” doesn’t take us anywhere. You can also see . and .. in your directory listing when you run “ls -al”

Now you are ready to do some shell scripting…

```
$ nano myscript.sh
```

You should see a drastic change to your screen! This command has launched the “nano” file editor.

Here we can enter the contents of a file, much like a word processor or notepad application in a desktop environment like Windows.

Let’s create a small program that we can run using nano. Enter the following lines in your nano window

```
#!/bin/bash
echo "Hello World!”
```

```
^G Get Help ^O WriteOut ^R Read File^Y Prev Page ^K CutText ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Page ^U UnCutTx ^T ToSpell
```

Then enter CTRL-X. At the bottom of the screen nano will ask you if you want to save the file. Type Y.

Then, nano will ask you what you want to name the file. Nano will suggest the file name we provided when we ran the command. Just hit enter since we already told nano what we wanted to call the file when we entered the nano command above.

```
$ ls –al
$ chmod 750 myscript.sh
$ ls –al
$ ./myscript.sh
```

Now we have created a bash script! But we can’t actually run the script until we grant file “execute” privileges.

```
$ chmod 750 myscript.sh
```

Then, we run the script with "./myscript.sh". The "./" tells Linux that the file we want to run is in the current directory.

Does your script work? Is the output what you expected?

Extra Practice

Take a look at the following shell script. What do you think this script will do? Try creating the shell script using nano (be sure use a different file name this time) and see if you get the expected result.

```
#!/bin/bash
x=0
while [ $x -lt 10 ]
do
echo "$x: Hello World!"
((x+=1))
done
```

```
^G Get Help ^O WriteOut ^R Read File^Y Prev Page ^K CutText ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Page ^U UnCutTx ^T ToSpell
```

II. Research Software

Create a prime directory within your home directory and copy the prime files there:

```
$ pwd
$ mkdir prime
$ cp /tools/docs/tutorials/mpi/newprime/* ~/prime/
$ cd ~/prime
$ ls –l
$ chown <username>:<username> *
```

Take a look at the files you have just copied. For this lab, we are most interested in prime.c and prime.sh. You can see inside those files using more or nano. The prime.c file is program source code written in the C language. In order to turn source code into a set of instructions that the computer understands, it must me compiled. There are many different types of languages and compilers. In this example we will be using the OpenMPI C compiler[1] to build our source code.

Setting the Environment/Compiling Code

To compile a MPI program so that it will run in parallel, you must use an MPI-enabled compiler.

```
$ mpicc prime.c –o prime
```

To make sure you have Open MPI available to use:

```
$ module list
$ module avail openmpi
$ module load openmpi/gcc
$ module list
```

Now try to compile the source code again:

```
$ mpicc prime.c –o prime
```

If you don't see any errors or warnings, you have just compiled a parallelized program to calculate prime numbers. Now we just need to figure out how to run it on the cluster.

---

[1] The OpenMPI C compiler is actually a "compiler wrapper" that adds certain features to the GNU C compiler (gcc). MPI stands for the Message Passing Interface, which allows a program to scale across a disparate machines, i.e. the nodes in a supercomputer.

III. Job Submission

Testing Code

Remember, it's not a good practice to run any code extensively on the login node. However it is certainly acceptable to quickly test an executable just to see if it will run at all.

Once an application is running in the foreground, you can enter [CTRL-C] to stop it.

If you are unsure about what you might be running, you can use the "ps" or "top" commands to see which processes are associated with your account. These commands will return Process IDs (or PIDs). If necessary, you can use the "kill" command along with the PID to stop a process. See "man kill" "man ps" "man

Now *briefly* run the executable that you just created by running it directly on the login node:

```
$ mpirun –np 2 prime
```

You should see a three-column list of numbers. (If you see an error message, something has gone wrong, so let an instructor know.) Use CTRL-C to stop the program.

Now that we know our compiled program works as expected, we need to confirm that it will also run through the scheduler. The scheduler make sure that resources in a supercomputer are shared efficiently. In this lab, we talk to the scheduler using a command called qsub.

A Note on Queues

Note: AU HPC systems define classes of different types of computers as queues. The queue(s) specified in these exercises with the qsub -q option may not be the queue(s) you should use beyond this training. More information is provided in the "Job Scheduling" section below.

Let's use qsub to run your program in interactive mode. This will run your program as a job on one of the compute nodes, but will allow us to issue commands to the node interactively.

```
$ qsub –q gen28 –l nodes=1:ppn=2 –I
$ module load openmpi/gcc
$ mpirun –np 2 ~/prime/prime
$ exit
```

Finally, submit your job as a batch job to the cluster.

```
$ qsub -q gen28 -l nodes=1:ppn=2 prime.sh -d ~/prime
```

What happened? Did the job run successfully?

Here's the prime.sh script:

```
$ mpirun -np 2 prime > Prime.out
```

Anything missing?

Here are some other important sub parameters by example (man qsub for more details):

```
$ qsub -l nodes=1,walltime=4:00:00 -m abe -M abc@auburn.edu prime.sh
```

Advanced Users

Another way to submit a batch job is by using PBS directives:

$ qsub prime.pbs

See **Next Steps** section at the end of this document for more detail.

How to monitor your job:

```
$ showq -u <userid>
$ qstat -u <userid>
$ qstat -f <job#>
$ checkjob –v –v –v <job#>
```

How to cancel a job:

```
$ mjobctl -c <job#>
$ qdel <job#>
$ canceljob <job#>
```

IV. Job Scheduling

Reservations

The Hopper HPC cluster uses “reservations” to define priority access to a subset of nodes for each lab group. Reservations use preemption to guarantee this access within 1-2 hours. Jobs that are running outside of their reserved capacity are subject to preemption. For more information, visit: https://aub.ie/hpcres

When will my job run? What resources are available?

Before submitting a job, users should consider what's running on the nodes in their reservation:

The showres command will show the reservations to which you have access.

```
$ showres
```

If available nodes in your reservation, then use your reservation. If no available nodes in your reservation, but you do NOT want your job preempted, then use your reservation. However, your job must wait.

Use this script to determine what’s running on your reservation:

```
$ whats-running-on-my-reservation.sh
```

How to specify your reservation in your job submission:

1. Use ADVRES flag in qsub

```
$ qsub -l nodes=1:ppn=20 -W x=FLAGS:ADVRES:<reservation_id> job.sh
```

2. Use rsub instead of qsub as it will automatically include your reservation in the job sub:

```
$ rsub -l nodes=1:ppn=20 job.sh
```

Note: Use rsub only when running in the general queue and you are a member of a single group.
Running outside your reservation:

If available nodes and you do NOT care if job is preempted, then submit without reservation.
If no available nodes and you submit without reservation, your job must wait.

Use this script to determine what resources are available:

```
$ /tools/scripts/find-excess-capacity.sh
```

To see what is currently running on the entire cluster:

```
$ showq
```

To see what jobs you have currently running

```
$ showq -u <username>
```

Best Practices Summary

I. Running a new program for the first time:

- First, run *briefly* on login node just to make sure that your code will run. If the program runs without an immediate indication or an error or problem, use CTRL-C to exit.
- Then, run using qsub in interactive mode to make sure that it will run on a compute node.
- Finally, run in batch mode using qsub.

Do not run jobs on the login node except as a test

This means short jobs using small amounts of memory to ensure that your code will run. Processes that violate this will be killed.

II. Don't submit a job, assume it's running correctly and walk away or leave for weekend. Make sure the job is running and, if not, understand why not.

III. Specify walltimes in your job submission.

- Allows Scheduler to maximize utilization which means your jobs run sooner.
- Users should receive an email after a job completes that contains the actual walltime.

Submit short-running jobs with fewer resources in order to reduce likelihood of preemption when not using your group's reservation.

IV. Clean up when your jobs are finished.

- Hopper does not provide archival or long-term storage.
- If files no longer need to be available for work on the system, copy them off and delete them so that the space can be used for active projects.

V. Pay attention to your disk usage. Once the hard limit is reached in disk space or # of files, your program will stop executing.

VI. Do not share passwords or accounts. If you want others to access your files, then set them to read only.

How to Get Help

Because Hopper is regarded as a research (rather than production) system, HPC support is normally available only during regular business hours. When reporting problems, please provide as much relevant information as possible. This should include the following, as appropriate:

- date and time when the problem occurred
- job number
- text of the command(s) which you issued
- exact and complete text of any error messages
- any other information helpful in identifying or resolving the problem

If further assistance is needed, please email to schedule an appointment with one of the HPC admins at hpcadmin@auburn.edu.

Next Steps

You can add qsub options to your shell script. This saves time and lots of typing when you have to specify lots of options

For example, you can run prime using a PBS script: prime.pbs…

```
#!/bin/bash

email=`whoami`@auburn.edu
workdir=/home/`whoami`/prime
cd $workdir
#PBS -N Prime
#PBS -m abe
#PBS -M $email
#PBS -l nodes=1:ppn=2,pmem=1gb
#PBS -q core
#PBS -d $workdir

module load openmpi/gcc
mpirun prime
```

```
^G Get Help ^O WriteOut ^R Read File^Y Prev Page ^K CutText ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Page ^U UnCutTx ^T ToSpell
```

Notice the #PBS directives in this script are equivalent to some of the qsub options that you have specified on the command line.

Find software for your particular research domain and begin experimenting. We have many popular software packages available for use. You can see them with …

```
$ module avail
```

If a software package that you would like to use is not already available, visit https://aub.ie/hpcsw to request a new package.

The best way to learn your way around is with hands-on experience. If you run into any problems or have questions, email hpcadmin@auburn.edu.