The Palmetto Cluster User's Guide

Last updated by Galen Collier, CITI 2011.12.19
Found a typo? Let me know: galen@clemson.edu



Computational Science at CLEMSON UNIVERSITY

Contents

Need Help? Contact our Support Team

E-mail ithelp@clemson.edu with the word *Palmetto* in the subject line.

This e-mail will be seen by all Palmetto support team members, and the person who can best handle your request will respond. The response time can vary from about 15 minutes to more than 1 day, depending on the nature or complexity of the problem, and this also depends on how busy the support team is other issues.

Thanks for your patience!

It's very helpful if you can include this information with your message (if appropriate):

- PBS job ID number
- Paste a copy of your PBS job script into your message so we can read through it
- Paste a copy of the error message you see when the problem occurs

Quick Linux Review

Don't use Linux every day? A bit rusty? No problem, here's a brief review of some of the basics:

Linux is just another operating system (OS), like Windows or Mac OSX, but Linux is the most efficient and powerful OS, so it's particularly useful for high-performance computing (HPC).

Interacting with the system through a simple command-line interface is required because there's no room for the computational "overhead" of running a graphical user interface (GUI), especially not on a large HPC cluster with hundreds of users connecting to the system through remote network connections.

Palmetto consists of hundreds of compute nodes (the nodes that do the computational work), plus a few service nodes that handle other activities. The most important service node is the "head" node, also called the log-in node. This node (named user001) is where you begin when you connect to Palmetto.

When you log-in

When you log-in, you'll be presented with a system message called the "message of the day" (MOTD). It looks something like this, with your command prompt waiting for you below this message:

```
--------------------------------------------------------------------------
         Welcome to the PALMETTO CLUSTER at CLEMSON UNIVERSITY

   * Please email ithelp@clemson.edu with questions or to report problems.
   * HPC webpage is http://citi.clemson.edu/hpc

   * The FIRST TUESDAY OF EACH MONTH, from 9:00am to 12:00noon,
     is reserved for system wide cluster maintenance.

   DO NOT RUN JOBS ON THE USER LOGIN NODE.  THEY WILL BE
   TERMINATED WITHOUT NOTICE.  NO EXCEPTIONS.

   Useful commands:
     checkquota               - get your current disk quota
     module avail             - list software packages
     qstat -xf jobid          - check status of jobid
     qstat -Qf queuename      - check status of queuename
     pbstop -m 24             - check status of jobs across the cluster

   Palmetto User Guide:   http://desktop2petascale.org/resources/159

   NOTE: Workaround for PBS and -k option
   If you use "-k oe" or "-k e" or "-k o", you must set permissions
   on your /home dir to 711:    chmod 711 /home/userid
   otherwise your output will be blank at the job's end.

 ---------------------------- /etc/motd ----- Last Updated: 17-Nov-2011 ---

[galen@user001 ~]$
```

Notice that the command prompt indicates what node (user001) I'm currently connected to. The ~ (tilde symbol) indicates that I'm in my home directory /home/galen.

When logged-in to Palmetto, you're using the "bash" shell (a.k.a. the Bourne-Again shell). The shell is a simple interface program that:

1) Reads (interprets) the commands you type
2) Tries to make sense of them
3) Figures out what to do with those commands
4) Finds the programs you're trying to run

Basic Commands

Try running some basic commands to see what the output looks like:

`pwd` ("print working directory") prints your current working directory
`cal` ("calendar") prints a little calendar for the current month
`date` prints the current date and time
`ls` list the contents of the directory you're in
`env` list all environment variables/settings

Most commands accept or require arguments (a.k.a. "flags" or "options") that customize or specify how they work:

`cd /scratch/galen` ("change directory") in this example, changing to Galen's directory in the /scratch filesystem
`echo "Hello there, how are you?"` prints the string of characters inside the "" quotes
`echo $PATH` the $ indicates that this is a variable, so here echo prints the value of the PATH environment variable, which contains a colon-separated list of all places where the bash shell searches for programs or commands to run
`which gcc` prints the full path to the gcc command (gcc is the GNU C compiler)
`cp compute.log /home/galen/logfiles` ("copy") in this example, I'm copying the compute.log file to the logfiles directory inside my home directory

A manual page ("manpage") serves as the "user manual page" for any particular command, including detailed information about how to use or customize each command:

`man ls` in this example, I'm displaying the manpage for the `ls` command

Using the details presented in the manpage, I found arguments that I could use for customizing the `ls` command:

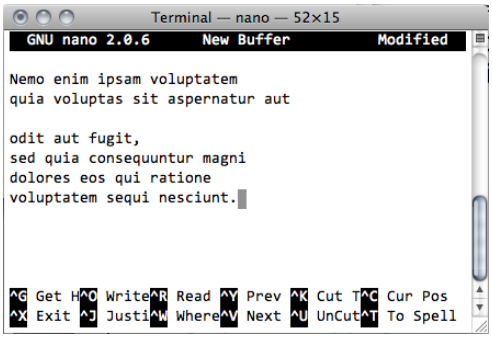
`ls -latr` lists the contents of the current directory, all files (including hidden files), and sorts them by time

Using a Text Editor (nano)

A very simple, easy-to-use text editor is nano. You can start nano and begin editing a file with this command:

`nano my-file.txt`

At the bottom of the nano interface, you'll see a menu of common commands, such as **^o** (that's CTRL-o) to save or "write" a file that has been edited, and **^x** (that's CTRL-x) to exit nano.



I/O redirection

I/O redirection is a way of manipulating the input/output of Linux programs, allowing you to capture output in a file, or send it to another program.

The **>** character instructs bash to take the output of a command and write it to a file:

`ls /scratch/galen > list.txt` writes the output of the '`ls /scratch/galen`' command to `list.txt`

Use **>>** to append to the end of a file without overwriting what's already there:

```
echo "Right now it's $DATE" >> list.txt
```

Another useful technique is to redirect one program's output (stdout) into another program's input (stdin). This is done using a "pipe" character (**|**).

`env` will print all of your current environment variables to the screen
`env | grep PBS` will send all of the '`env`'output to the '`grep PBS`' command, and the result will be all environment variables that contain the string "PBS"

Your .bashrc File

Every time you log-in, the **`.bashrc`** script in your home directory is executed (note the dot "." at the beginning of the name -- this means it's a hidden file, so use **`ls -a`** to see it).

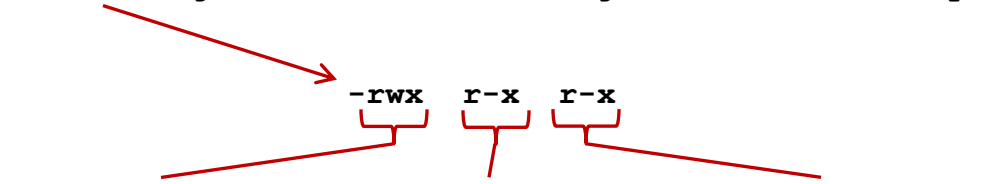
You can add lines to the bottom of this file to run additional, custom commands every time you log-in. For example, you can set a new environment variable with a line like this (here, using the **`export`** command) in your **`.bashrc`** script:

```
export MPI_HOME=/opt/mpich2/1.4
```

File and Directory Permissions

Control access to files and directories by setting permissions (r = read, w = write, x = execute). Use **`ls -l`** ("long listing") to see permissions settings:

```
-rwxr-xr-x 1 galen staff   622783 Aug 11 08:35 dictionary.txt
```




```
0 = ---
1 = --x
2 = -w-
3 = -wx
4 = r--
5 = r-x
6 = rw-
7 = rwx
```

You can set or modify permissions settings using 3-digit octal notation:

`chmod 744 dictionary.txt` here, I'm changing permissions for everyone but myself to read-only (**`r--`**), so now everyone can read or copy this file, but they cannot modify, move, or delete the original.

Linux Commands Reference "Cheatsheet" (on the next page…)

File Commands

`ls` = directory listing
`ls -al` = formatted listing with hidden files
`cd dir` = change directory to *dir*
`cd` = change to your /home directory
`pwd` = show current directory
`mkdir dir` = create directory *dir*
`rm file` = delete *file*
`rm -r dir` = delete directory *dir*
`rm -f file` = force deletion of *file*
`rm -rf dir` = force deletion of *dir*
`cp file1 file2` = copy *file1* to *file1*
`cp -r dir1 dir2` = copy *dir1* to *dir2*
`mv file1 file2` = rename or move *file1* to *file2*
`mv file1 dir` = move *file1* into *dir*
`ln -s file link` = create symbolic link to *file*
`touch file` = create or update *file*
`cat > file` = redirects stdin into *file*
`more file` = output contents of *file*
`head -8 file` = output first 8 lines of *file*
`tail -8 file` = output last 8 lines of *file*
`tail -f file` = output contents of *file* as it grows

Process Management

`ps` = display your currently active processes
`top` = display all running processes
`kill pid` = kill process with ID number *pid*
`killall proc` = kill all processes named *proc*
`bg` = lists stopped or background processes
`fg` = brings most recent process to foreground
`fg n` = brings process *n* to the foreground

File & Directory Permissions

`chmod octal file` = change permissions of file
4 = read(r) 2 = write(w) 1 = execute(x)
4 + 1 = 5 (read and execute)
`chmod 777 file` = read, write, execute for all
`chmod 700 file` = rwx for owner only
`chmod 755 file` = rwx for owner, rx for everyone else
`chown userid file` = change ownership of *file*
`chgrp group file` = change group ownership of *file*

SSH & SCP

`ssh user@host` = connect to *host* as *user*
`ssh -p port user@host` = connect using port number *port*
`ssh -X user@host` = connect with X11 forwarding
`scp file1 user@host:directory/file2` = secure copy *file1* to *file2* on remote system
`scp user@host:directory/file1 file2` = secure copy *file1* from remote system to *file2* on your local system

Searching

`grep pattern file(s)` = search for pattern in *file(s)*
`grep -r pattern dir` = search recursively for pattern in *dir*
`command | grep pattern` = search for pattern in output of command
`locate file` = find all instances of *file*
`find . -name "file.txt"` = recursively find all instances of *file.txt* starting in current directory

System Info

`date` = show current date and time
`cal` = show current month's calendar
`who` = display who is logged-in
`whoami` = display username of current login
`uname -a` = display OS and kernel version info
`cat /proc/cpuinfo` = display CPU info
`cat /proc/meminfo` = display memory info
`man command` = display manual page for *command*
`df -h` = show disk usage
`du` = show directory space usage
`which app` = show full path to *app*

Compression

`tar -cvf file.tar file(s)` = create a .tar file containing *file(s)*
`tar -xvf file.tar` = extract contents of *file*.tar
`tar -cvfz` = create .tar file with Gzip compression
`tar -zxvf` = extract contents of a Gzip compressed .tar file
`gzip file` = compresses file and renames it to *file*.gz
`gzip -d file.gz` = decompresses *file*.gz back to file

Installing Software

`./configure` = run the configure script
`make` = build the software based on configure info
`make install` = run the installation process
`rpm -Uvh package.rpm` = install an RPM package

VI and VIM

`vim file` = edit file with VIM
`i` = start "insert" mode (so you can edit the file)
`ESC` = return to "command" mode
`:q!` = force quit, from "command" mode
`:wq` = write (save) the file and quit, from "command"

Miscellaneous

`Ctrl+C` = halts the current command
`exit` = log-out of current session
`Ctrl+D` = log-out of current session (same as exit)
`!!` = repeats the last command
`cd -` = change directory to previous location

General Overview of the Palmetto Cluster

Both multicore shared-memory systems and multicore distributed-memory architectures
Benchmarked operating at over 96 TF/s
Ranked #128 on November 2011 “Top 500” list
Ranked #2 among public academic institutions
Currently consists of 1,616 compute nodes (14,168 cores)
Myrinet 10G network interconnect
Operating system on all nodes: Scientific Linux 6 (based on RedHat Enterprise Linux)
Job queuing system: PBS Professional 11.1

Data Storage

Each user account has 100 GB of backed-up space in the /home directory.
All users share 115 TB temporary “work” space in /scratch (not backed up), just create a directory for yourself.
SC-EPSCoR users share a 6 TB backed-up storage array in /common1/epscorci.

Compute Node Hardware Specifications

	Name	Count	Model	Processor	L2 Cache	Cores	Memory	Local Disk
compute node phase 1	node0001-0257	257	Dell PE 1950	Intel Xeon E5345 @2.33GHz x 2	4 MB	8	12 GB	80 GB (SATA)
compute node phase 2	node0258-0515	258	Dell PE 1950	Intel Xeon E5410 @2.33GHz x 2	6 MB	8	12 GB	80 GB (SATA)
compute node phase 3	node0516-0771	256	Sun X2200 M2 x64	AMD Opteron 2356 @ 2.3GHz x 2	4 MB	8	16 GB	250 GB (SATA)
compute node phase 4	node0772-1023,1108-1111	256	IBM dx340	Intel Xeon E5410 @2.33GHz x 2	6MB	8	16 GB	160 GB (SATA)
compute node phase 4.1	node1024-1107	84	IBM dx340	Intel Xeon E5410 @2.33GHz x 2	6MB	8	16 GB	160 GB (SATA)
compute node (former CCMS nodes)	node1112-1541	430	Sun X6250	Intel Xeon L5420 @2.5GHz x 2	6MB	8	32 GB	160 GB (SATA)
compute node phase 6	nodes 1553-1622	70	HP DL 165 G7	AMD Opteron 6172 @2.1GHz x 2	12MB	24	48 GB	250 GB (SATA)
large shared memory systems	nodelm01-nodelm04	4	HP DL 580 G7	Intel Xeon 7542 @ 2.66 GHz x 4	18MB	24	512 GB	146 GB (SAS)
math sciences large memory	nodemath	1	HP DL 980 G7	Intel Xeon 7560 @ 2.66 GHz x 8		64	2 TB	

Accessing Palmetto

Clemson users can request an account (a Base Allocation) at http://citi.clemson.edu/allocation_type

Connecting to the Head/Log-in Node

MacOSX & Linux/Unix users, simply open a Terminal window and connect via SSH:

```
ssh [your CU network user ID]@user.palmetto.clemson.edu
```

Windows users, connect using SSH Secure Shell (http://www.clemson.edu/softrepo/Win/SSH/sshclient.exe):

Host Name: `user.palmetto.clemson.edu`
User Name: *[your CU network user ID]*
Port Number: `22`
Authentication Method: *[none specified]*

Transferring Files to/from Palmetto

Many users choose to use the FileZilla SFTP client. You can download FileZilla from http://filezilla-project.org

When connecting to Palmetto using FileZilla (or any other SFTP client), you will use the same connection info that you use for the SSH Secure Shell client:

Host: `user.palmetto.clemson.edu`
Username: *[your Palmetto username]*
Password: *[your Palmetto password]*
Port: `22`

Alternatively, you can use SFTP, SCP, or any SFTP client you prefer.

First Thing to do with your New Palmetto Account

Setup Passwordless SSH

When you first log-in, setup passwordless SSH (you will only need to do this once):

```
ssh-keygen -t rsa
```

```
    Generating public/private rsa key pair.
    Enter file in which to save the key (/home/userid/.ssh/id_rsa):       [Accept defaults, just press Enter]
    Enter passphrase (empty for no passphrase):                           [Accept defaults, just press Enter]
    Enter same passphrase again:                                          [Accept defaults, just press Enter]
    Your identification has been saved in /home/userid/.ssh/id_rsa.
    Your public key has been saved in /home/userid/.ssh/id_rsa.pub.
    The key fingerprint is: 64:72:2a:7b:20:fa:a7:0c:91:26:a6:43:85:0b:1c:21
```

```
cd .ssh
cp id_rsa.pub authorized_keys
```

Test your SSH configuration by starting an interactive session:

```
qsub -I
```

If your interactive session starts, you'll be logged-on to one of the compute nodes. If you are NOT asked for your password, then you have successfully configured passwordless SSH.

You're finished now, so be sure to end your interactive session (type CTRL+d, or type **`exit`**).

Adding & Changing Environment Settings

Bash is the default shell on Palmetto.
Customize your /home/*username*/.bashrc file by adding aliases, commands, functions to the end of this file.
Your .bashrc file will be executed (it's a bash script) with every log-in session.

Example: `alias vi='/usr/bin/vim'`
Example: `export PATH=/scratch/galen/gcc/4.6.1/bin:$PATH`
Example: `export CXX=/scratch/galen/gcc/4.6.1/bin/g++`
Example: `export LIBRARY_PATH=$LIBRARY_PATH:/scratch/galen/gmp/5.0.2/lib`

Loading Software Packages with the Module System

Use the module system to manage general-purpose software packages:

List all packages available (on current system): **`module avail`**
Add a package to your current shell environment: **`module add`** ***`[package/version]`***
List packages you have loaded: **`module list`**
Purge currently loaded packages: **`module purge`**

Note: Avoid loading conflicting modules. For example, loading 2 different versions of GCC at the same time.

Submitting Jobs to Run on the Cluster

For batch jobs: `qsub [PBS job script]`
For interactive jobs (interactive sessions): `qsub -I [qsub parameters]`

Check the status of all active jobs: `qstat -u [username]`

Common Parameters for the qsub Command

These can be used for both interactive or batch jobs:

Parameter	Purpose	Example
`-N`	Job name (7 characters)	`-N maxrun1`
`-l`	Job limits (lowercase L), hardware & other requirements for job	`-l select=1:ncpus=8:mem=1gb`
`-q`	Queue to direct this job to (main is the default, epscorci is the higher-priority EPSCoR queue)	`-q epscorci`
`-o`	Path to stdout file for this job (environment variables are no longer accepted here)	`-o stdout.txt`
`-e`	Path to stderr file for this job (environment variables are no longer accepted here)	`-e stderr.txt`
`-M`	E-mail for messages from the PBS server	`-M galen@clemson.edu`
`-m`	Type of notification you wish to receive (b,e,a,n)	`-m ea`

There is a very large variety of qsub options (aka PBS directives) available in the PBS Pro User's Guide, which can be found here: http://www.pbsworks.com/documentation/support/PBSProUserGuide11.1.pdf

Things you should specify for your Job

Example: `-l select=4:ncpus=8:mpiprocs=8:mem=11gb,walltime=48:00:00`

Memory specification using `mem=##gb or mem=##kb` (Note: this is memory per node)

Cores per node and available MPI processes per node (these values should usually the same): `ncpus=8:mpiprocs=8`

You can specify hardware details using keywords like intel, xeon, amd, opteron, myrinet as in these examples:

```
-l select=1:ncpus=8:chip_manufacturer=intel:myrinet=true
-l select=1:ncpus=8:chip_model=opteron:myrinet=false
-l select=1:ncpus=8:chip_type=e5410:myrinet=true:mem=15gb,walltime=16:00:00
-l select=1:ncpus=8:chip_type=o2356:myrinet=true:mem=15gb
-l select=1:ncpus=1:node_manufacturer=ibm:mem=15gb,walltime=00:20:00
```

Possible **`qsub -l`** hardware specification options include:

```
chip_manufacturer=amd
chip_manufacturer=intel
chip_model=opteron
chip_model=xeon
chip_type=e5345
chip_type=e5410
chip_type=l5420
chip_type=x7542
chip_type=2356
chip_type=6172
node_manufacturer=dell
node_manufacturer=hp
node_manufacturer=ibm
node_manufacturer=sun
myrinet=true
myrinet=false
```

Wall time

Maximum wall time for all jobs in the default "workq" queue is 72 hours.

Your job will terminate abruptly when it reaches the wall time, so specify a wall time that is slightly longer than what your job will actually require (to allow for any delays in network communication or data I/O). For example, with an 8-hour job, use walltime=9:00:00.

Note: the wall time for an active job cannot be extended.

Owner queues (special queues useable by researchers and groups who have purchased dedicated resources) can run jobs up to 168 hours.

If a regular (non-owner) user needs to run a job for longer than the default 72-hour wall time, that user must submit an allocation request explaining what resources are needed (e.g., how many nodes), for how long, and some justification for why this work must be run in a single job without be divided into shorter jobs.

The online allocation request form can be found here: http://citi.clemson.edu/allocation_type

Example PBS Job Scripts

In these examples, optional modules used for running the desired programs are loaded within the job script, rather than doing this statically using a "**`module add`** …" command in your /home/*username*/.bashrc file.

It's not required that you load modules in this way (from within the job script), but it may be easier to troubleshoot.

Serial Batch Job Script

```
#!/bin/bash
#PBS -N example
#PBS -l select=1:ncpus=1:mem=2gb
#PBS -j oe
#PBS -l walltime=00:10:00

source /etc/profile.d/modules.sh
module purge
module add gcc/4.5.1 fftw/2.1.5-double

cd $PBS_O_WORKDIR

./my-program.exe some-input.inp > some-output.log
```

Parallel Batch Job Script

```
#!/bin/bash
#PBS -N example
#PBS -l select=2:ncpus=8:mpiprocs=8:mem=11gb
#PBS -j oe
#PBS -l walltime=00:10:00

source /etc/profile.d/modules.sh
module purge
module add gcc/4.5.1 mpich2/1.4 fftw/2.1.5-double
NCORES=`wc -l $PBS_NODEFILE | gawk '//{print $1}'`

cd $PBS_O_WORKDIR

mpiexec -n $NCORES ./my-parallel-program.exe some-input-file.inp > some-output.log
```

Examples of Interactive Jobs

Simplest example with only the defaults, 1 core on 1 node with 1 GB of memory for 30 minutes in the "workq" queue:

```
qsub -I
```

Testing a serial program, compiling code, or running analysis scripts with 1 core on 1 node with 6 GB of memory for 2 hours:

```
qsub -I -l select=1:ncpus=1:mem=6gb,walltime=2:00:00
```

Testing a parallel program with 16 AMD cores on 2 nodes with 31 GB of memory (on each node) for 45 minutes:

```
qsub -I -l select=2:ncpus=8:mpiprocs=8:chip_manufacturer=amd:mem=31gb,walltime=00:45:00
```

X11 Tunneling (for a GUI) from Palmetto

Running a GUI-based application using X11 forwarding via SSH with 1 core on 1 node with 11 GB of memory for 3 hours:

Part 1, in one terminal window, launch an interactive job:
```
qsub -I -l mem=11gb -lselect=1:ncpus=1,walltime=03:00:00
```

Part 2, in a second terminal window, ssh to your allocated node(s) with the -X option:
```
ssh -X node0931
```
From this second terminal window, the program that generates the GUI can be run while your PBS job is active.

Running an application on one of the large shared memory systems along with tunneling a GUI (X11 forwarding via SSH) with 4 cores on 1 node with 128 GB of memory for 2 hours:

Part 1, in one terminal window, launch an interactive job:
```
qsub -I -q bigmem -l mem=128gb -lselect=1:ncpus=4:mpiprocs=4,walltime=02:00:00
```

Part 2, in a second terminal window, ssh to your allocated node(s) with the -X option:
```
ssh -X node0934
```
From this second terminal window, the program that generates the GUI can be run while your PBS job is active.

Monitoring the Status of your Jobs (or Queues)

Check the status and details about a particular job: `qstat -xf jobID`
Check the status of all of your jobs: `qstat -u username`
Check the status and details about a specific queue: `qstat -Qf queue`

Killing Jobs

Killing a single job, or just a few jobs: `qdel [jobID]` Example: `qdel 3012334 3012338 3012339`
Killing all of your jobs: `qselect -u [username] | xargs qdel`
Killing jobs with a specified status (R for running, Q for queued): `qselect -u [username] -s R | xargs qdel`

Using the /scratch Filesystem

/scratch is writeable by all users, so create a directory for yourself there and call it whatever you want.

All computational work (active jobs, data analysis, etc.) and all reading & writing by jobs should be conducted within the /scratch filesystem. That means you must move all of your input (input files, executables, required libraries, etc.) to /scratch before launching your job there. All of your output must also be written within the /scratch filesystem.

Executables & libraries provided via the module system do not need to be moved to /scratch.

/scratch is not backed-up, so move important files/data to your /home directory when your job is finished.

Also, try keep the number of files per directory to less than 1,000 (for optimal performance).

Using ANSYS on Palmetto

Running ANSYS on Palmetto requires users to check-out an ANSYS license in their job script so the necessary license will be available when the job begins running.

There are 3 types of ANSYS licenses:

1) The High Performance Research license (**aa_r_hpc**). This is per core, and is required when running Distributed ANSYS. That is, they execute ansys like this: **ansys130 -dis -usessh -machines ...** There are a maximum of 240 licenses of this type.
2) The Teaching License (**aa_t_a**). This license can be used first 2 cores for 1 license, and one license for each additional core. They are using the teaching license if they invoke ANSYS like this: **ansys130 -p AA_T_A -n $NCPUS ...** (the **-p** means `package`). There are 100 of this type of license available, and only 95 of those are accessible through the PBS system on Palmetto.
3) The Research License (**aa_r**). This is the default if neither of the above two options are used. There are 25 of this type available, but only 10 are available through the PBS system on Palmetto.

Running ANSYS in a Batch job

Below is an example of a PBS job script used for running ANSYS. In this example, 16 compute cores (two 8-core nodes) are being used, so 16 High Performance Research licenses are reserved in the job limits (hardware selection) line:

```
#!/bin/bash
#PBS -N ANSYSdis
#PBS -l select=2:ncpus=8:mpiprocs=8:mem=11gb,aa_r_hpc=16
#PBS -l walltime=1:00:00
#PBS -j oe

source /etc/profile
module purge
module add ansys/13.0

cd ${PBS_O_WORKDIR}

machines=$(uniq -c $PBS_NODEFILE | awk '{print $2":"$1}' | tr '\n' :)
time ansys130 -dir ${PBS_O_WORKDIR} -j TOP -s read -l en-us -b -i newtop.txt -o
output.txt -dis -machines ${machines} -usessh
```

Also, users can check to see how many of which licenses are currently in use using the following command (this long command spans 2 lines here, but when you issue this command in your terminal window, it should all be in one line):

```
/opt/ansys_inc/shared_files/licensing/linx64/lmutil lmstat -a -c
28008@licensevm4.clemson.edu
```

Using MATLAB on Palmetto

Palmetto is configured so that MATLAB code should be compiled while logged-in to user001 using MATLAB's **mcc** compiler, then the resulting executable is run in a job submitted to the cluster (with the **matlab/2010a** module and, if needed, a GCC compiler module loaded). When you compile with **mcc**, you may see a "no ifconfig" error which is meaningless. You can ignore it. If the resulting executable was created, then the compilation succeeded (even if that error message is displayed).

So, once you've created your executable, you can run it in a job on Palmetto. Of course, you'll also need the same **matlab/2010a** and GCC modules loaded for your job's runtime environment.

Compiling a new MATLAB-based Executable

Below is an example of the steps for compiling and running a simple MATLAB .m program:

Here's my example .m file. This very simple program creates a plot and writes it to the fig1.ps file (and displays the plot on the screen if you run it using an interactive session and ssh -X):

```
[galen@user001 ~]$ cat matchx.m

function a=b()
i=2;
a=[1 2 3];
b=[1 3 5];
plot(a,b);
print fig1.ps;
end
```

To compile with MATLAB's mcc compiler, I start with no modules loaded, and nothing in my .bashrc file.

```
[galen@user001 ~]$ module list
No Modulefiles Currently Loaded.

[galen@user001 ~]$ module add matlab/2010a

[galen@user001 ~]$ which mcc
/opt/matlab/R2010a/bin/glnxa64/mcc

[galen@user001 ~]$ mcc -m matchx.m
which: no ifconfig in
(/opt/matlab/R2010a/bin/glnxa64:/opt/matlab/R2010a/bin:/usr/kerberos/bin:/usr/local/bin:/
bin:/usr/bin:/opt/mx/bin:/opt/dell/srvadmin/bin:/home/galen/bin)
/opt/matlab/R2010a/runtime/glnxa64/libmwmclmcrrt.so: undefined reference to
`std::basic_ostream<char, std::char_traits<char> >&
std::__ostream_insert<char, std::char_traits<char>
>(std::basic_ostream<char, std::char_traits<char> >&, char const*,
long)@GLIBCXX_3.4.9'
collect2: ld returned 1 exit status

mbuild: link of 'matchx' failed.

Error: An error occurred while shelling out to mbuild (error code = 1).
Unable to build executable (specify the -v option for more information).
```

Okay, because of that error, it looks like I need to load a compiler suite in addition to MATLAB. I'll load the gcc/4.4 compilers:

```
[galen@user001 ~]$ module add gcc/4.4

[galen@user001 ~]$ mcc -m matchx.m
which: no ifconfig in
(/opt/gcc/4.4/bin:/opt/matlab/R2010a/bin/glnxa64:/opt/matlab/R2010a/bin:/usr/kerberos/bin
:/usr/local/bin:/bin:/usr/bin:/opt/mx/bin:/opt/dell/srvadmin/bin:/home/galen/bin)

Warning: You are using gcc version "4.4". The version
currently supported with MATLAB Compiler is "4.2.3".
For a list of currently supported compilers see:
http://www.mathworks.com/support/compilers/current_release
```

That "no ifconfig" error will always be present, so just ignore that. My executable has been built successfully and now I'm ready to run it.

Running a new MATLAB-based exe Interactively

As a first example, since MATLAB is going to display the plot for me when I run this program, I'll do this using an interactive job (in one terminal window) with **`ssh -X`** to the node assigned to my interactive job (in a separate, 2nd terminal window):

```
[galen@user001 ~]$ qsub -I
qsub: waiting for job 722527.pbs01 to start
qsub: job 722527.pbs01 ready
```

Now, I need to find out what node my interactive job is running on:

```
[galen@node1170 ~]$ qstat -xf 722597 | grep exec_vnode
    exec_vnode = (node1170:ncpus=1)
```

Okay, so my **`ssh -X`** connection needs to be with node1170 (and I need to start Xming, if it's not already running):

```
[galen@user001 ~]$ ssh -X node1170
Warning: Permanently added 'node1170,10.125.5.162' (RSA) to the list of known hosts.
[galen@node1170 ~]$
```

Now, I can run my new executable and generate a GUI tunneled from **`node1170`**:

```
[galen@node1170 ~]$ cd /scratch/galen/mtest
[galen@node1170 ~]$ module add matlab/2010a
[galen@node1170 ~]$ ./matchx
```

At this point, the plot is displayed by the Xming X-server running on my desktop.

Running a new MATLAB-based exe as a Batch Job

For the next example, I'll run the same job as a batch job. Here's what my PBS job script looks like:

```
[galen@user001 ~]$ cat job.mtest.bash
#!/bin/bash
#PBS -N Mtest0
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -j oe
#PBS -l walltime=00:05:00
```

```
source /etc/profile.d/modules.sh
module purge
module add matlab/2010a

cd $PBS_O_WORKDIR

## Run the executable compiled with MATLAB:
./matchx
```

Now, I'll submit this job to run on Palmetto:

```
[galen@user001 ~]$ qsub job.mtest.bash
722401.pbs01
```

It's finished within about 1 minute, and here is a listing of the contents of my work directory after I ran this job (note that the fig1.ps file was created during the job):

```
[galen@user001 ~]$ ls -ltr
-rwxr-xr-x 1 galen staff 101 Aug 12 23:50 matchx.m
-rw-r--r-- 1 galen staff 72K Aug 12 23:53 mccExcludedFiles.log
-rwxr--r-- 1 galen staff 1018 Aug 12 23:53 run_matchx.sh
-rw-r--r-- 1 galen staff 3.7K Aug 12 23:53 readme.txt
-rw-r--r-- 1 galen staff 27K Aug 12 23:53 matchx.prj
-rw-r--r-- 1 galen staff 5.9K Aug 12 23:53 matchx_mcc_component_data.c
-rw-r--r-- 1 galen staff 3.3K Aug 12 23:53 matchx_main.c
-rwxr-xr-x 1 galen staff 49K Aug 12 23:53 matchx
-rwxr-xr-x 1 galen staff 597 Aug 13 00:03 job.mtest.bash
-rw-r--r-- 1 galen staff 8.2K Aug 13 00:04 fig1.ps
```

Running COMSOL 4.2 on Palmetto from a local Windows workstation

(Using "SSH Secure Shell" and "Xming" --or-- in batch mode)

Running COMSOL 4.2 on Palmetto:

There are 2 ways to run COMSOL on Palmetto. You can launch the COMSOL GUI (graphical user interface) and interact with the program that way, or you can run COMSOL in batch mode using a PBS job script. To run COMSOL as a PBS job in batch mode only, there is no need to install or use the Xming Windows-based X-server.

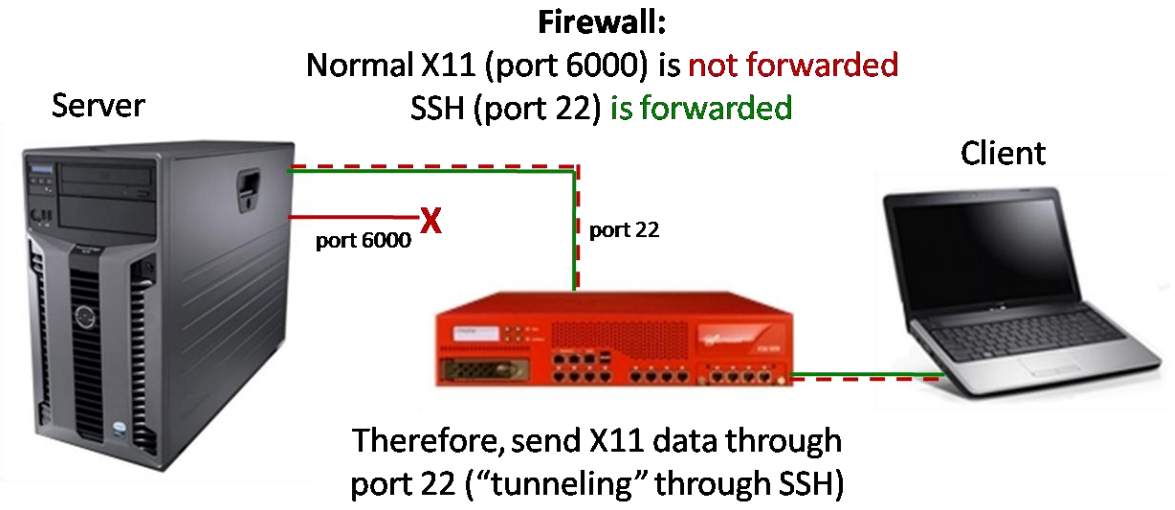
Prerequisites:

- ✓ Access to CCIT software resources (to access the SSH Secure Shell software)
- ✓ A user account for the Palmetto HPC system
- ✓ Administrator privileges in your local Windows-based OS

About tunneling (Forwarding) an X-Windows Session through SSH:

X-Windows sessions, which communicate the information needed to display and interact with a GUI using the X11 communications protocol, normally use port number 6000 for passing data back and forth. However, most HPC clusters use a firewall that will not forward data using port 6000.

So, to enable an X-Windows session through a firewall, we must "tunnel" our X-Windows data through a communication port that is forwarded by the firewall. Port number 22, which is normally used for SSH communication, will work for this purpose. The figure below illustrates this concept:

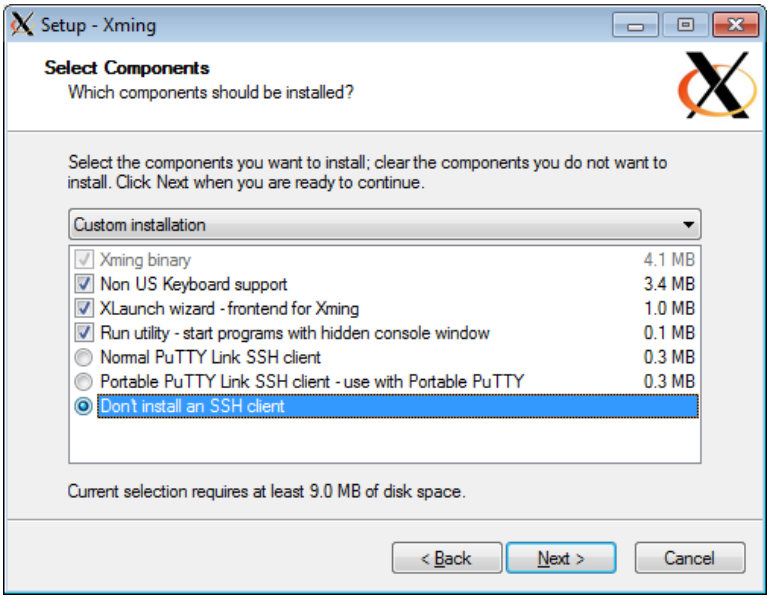


Download and Install:

SSH Secure Shell by SSH Communications Security Corporation, enhanced SSH client with CU site license (http://www.clemson.edu/ccit/software_applications/software/licenses/ssh.html)

Xming, Windows-based X-server
(http://www.straightrunning.com/XmingNotes select "Xming" under Public Domain Releases)

When installing Xming, using all of the default settings is recommended, but it is not necessary to install the built-in PuTTY SSH client:



Connecting to Palmetto Using SSH Secure Shell:


Launch the SSH Secure Shell client.

Click the Quick Connect button.

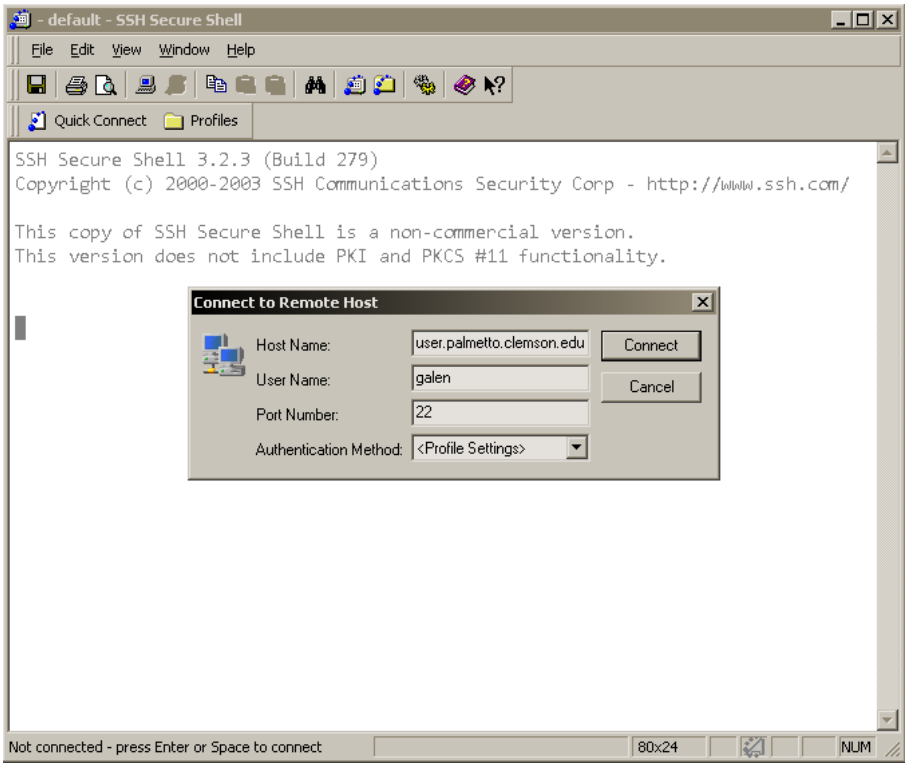
Connect to Palmetto with
user.palmetto.clemson.edu
as the host name.

Use your CU user ID or assigned user name
(you'll be asked for your password
after you connect successfully).

Port 22 is the default port assignment for SSH.

Once you log-in, choose Yes to
save Palmetto's host key in the local
database.

When asked to save your connection
Settings in a Profile, you can simply name
It "Palmetto" or something similar.

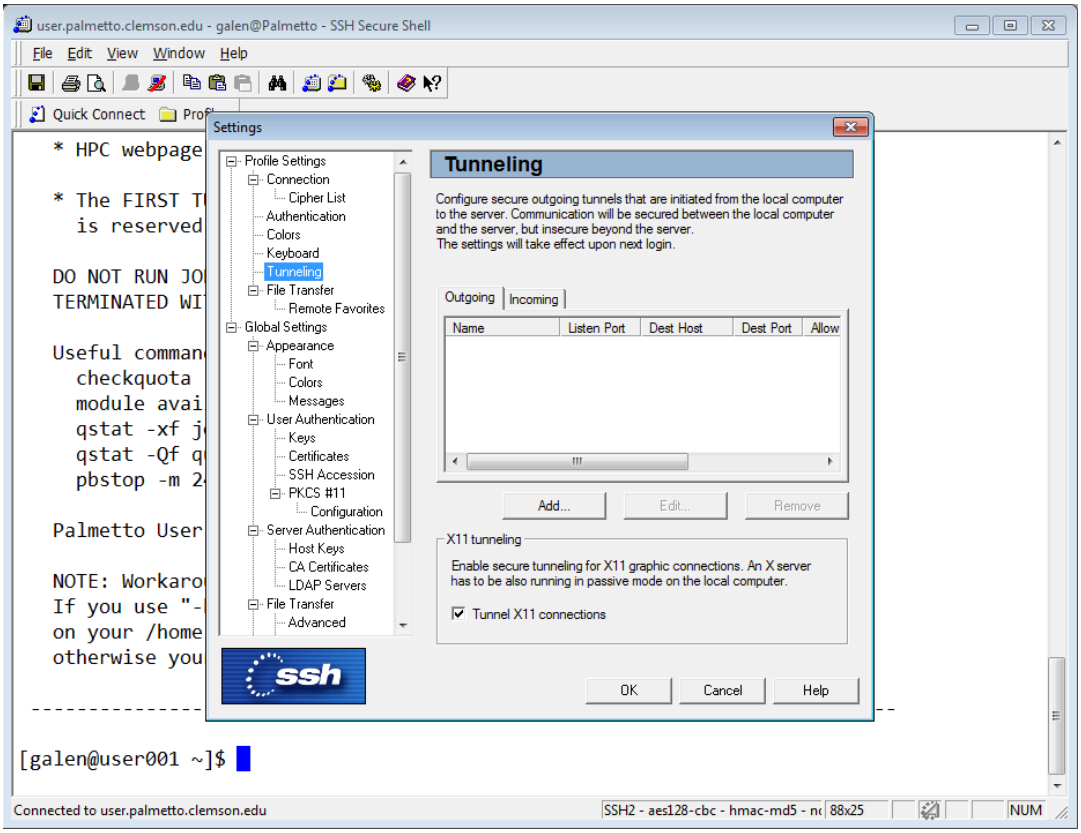


Once logged-in, you'll immediately see some messages posted by the HPC Administration staff (known as the "Message of the Day" or MOTD). It's very important to read these messages every time you log-in to the system because they often include notices of critical operational changes or scheduled maintenance periods.

Configuring SSH Secure Shell for tunneling X11:

After you have saved your connection profile, you can make changes to the settings of that profile. It's important to note that saved settings will be different for each saved profile.

In the SSH Secure Shell window, open Edit > Settings and select Tunneling from the list of options. At the bottom of the Tunneling settings window, check the checkbox for "Tunnel X11 connections" and click OK. Save this change by choosing File > Save Settings.



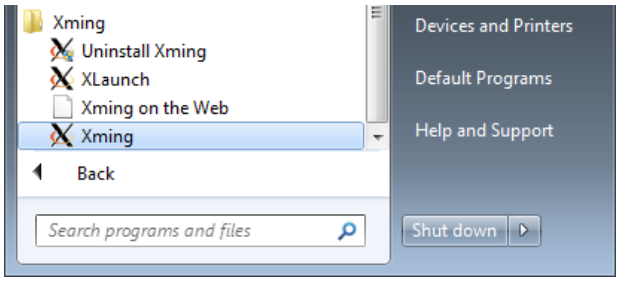
Once you've saved the updated settings, you MUST close SSH Secure Shell and restart it.

Launch an Xming X-server:

In order to receive incoming X-windows (X11 protocol) sessions, you need to have an X-server running on your local system. This will allow you to view and interact with the GUI windows generated by an X-windows-enabled application running on a remote server.

Your local X-server will "listen" for incoming X-Windows sessions and display the GUI components that it receives.

Launch Xming from the Windows Start menu:



When you launch Xming, it will be running in the background, listening for incoming X-Windows sessions. You can see that it's running by noting the Xming icon in the system tray:

Now, you are ready to launch a program on Palmetto that tunnels X11 to your local X-server.

Launching the COMSOL 4.2 GUI via tunneling from Palmetto:

Tunneling the COMSOL 4.2 graphical user interface will require you to open 2 SSH Secure Shell windows, both connected to Palmetto. If you already have one open and connected to Palmetto, simply click on the New Terminal Window button to open another window usi he same profile.

Let's refer to these 2 SSH Secure Shell windows as "terminal window #1" and "terminal window #2."

In terminal window #1, you'll launch an interactive job that will allocate the hardware (compute node) needed to run COMSOL. This can be done using a qsub command like this:

```
qsub -I -q bigmem -l select=1:ncpus=8:mpiprocs=8:mem=128gb,walltime=2:00:00
```

This command will launch an interactive job with one "chunk" of cluster hardware consisting of 8 compute cores, 8 MPI processes, 128 GB RAM, and this job will run for up to 2 hours. Since this job will be using more than 47 GB of RAM, it must run on one of Palmetto's 4 large shared memory nodes (nodelm01-nodelm04). Also, COMSOL 4.2 is only available on these nodes, and access to these nodes is only possible if you include the "-q bigmem" specification in your qsub command, as we have done here.

Once your interactive job begins running, terminal window #1 will be logged-in to one of the large shared memory nodes. With an active job running on any compute node, you can SSH directly to that compute node from another terminal window.

In terminal window #2, use the ssh -X command to connect to the large shared memory node where your interactive job is running:

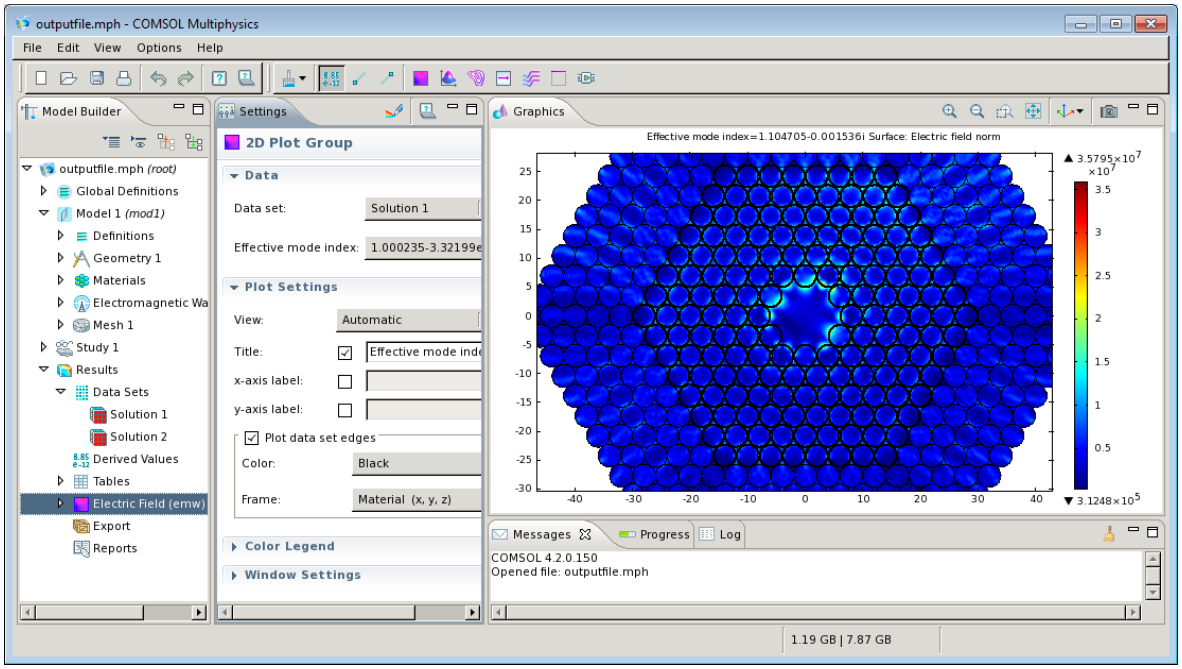
```
ssh -X nodelm04
```
(of course, here you should specify which node *your* job is using)

Since this terminal window is connected via SSH with X11 tunneling enabled, you will use this window to launch the COMSOL GUI. Load the COMSOL 4.2 module and launch the COMSOL GUI while specifying the number of cores you have reserved in your interactive job. In this example, I'm using 8 cores:

```
module add comsol/4.2
```

```
comsol -np8
```

Tunneling X11 is not very fast, so the performance of the GUI will be much slower than that of a program running on your local workstation. It may take approximately 15 seconds for the tunneled GUI to be displayed by your local Xming X-server. Interacting with the tunneled GUI (clicking buttons and using menus, etc.) will be slow, so please consider the amount of data moving back-and-forth between your workstation and Palmetto. The performance may be even worse if you're tunneling the GUI to an off-campus location.



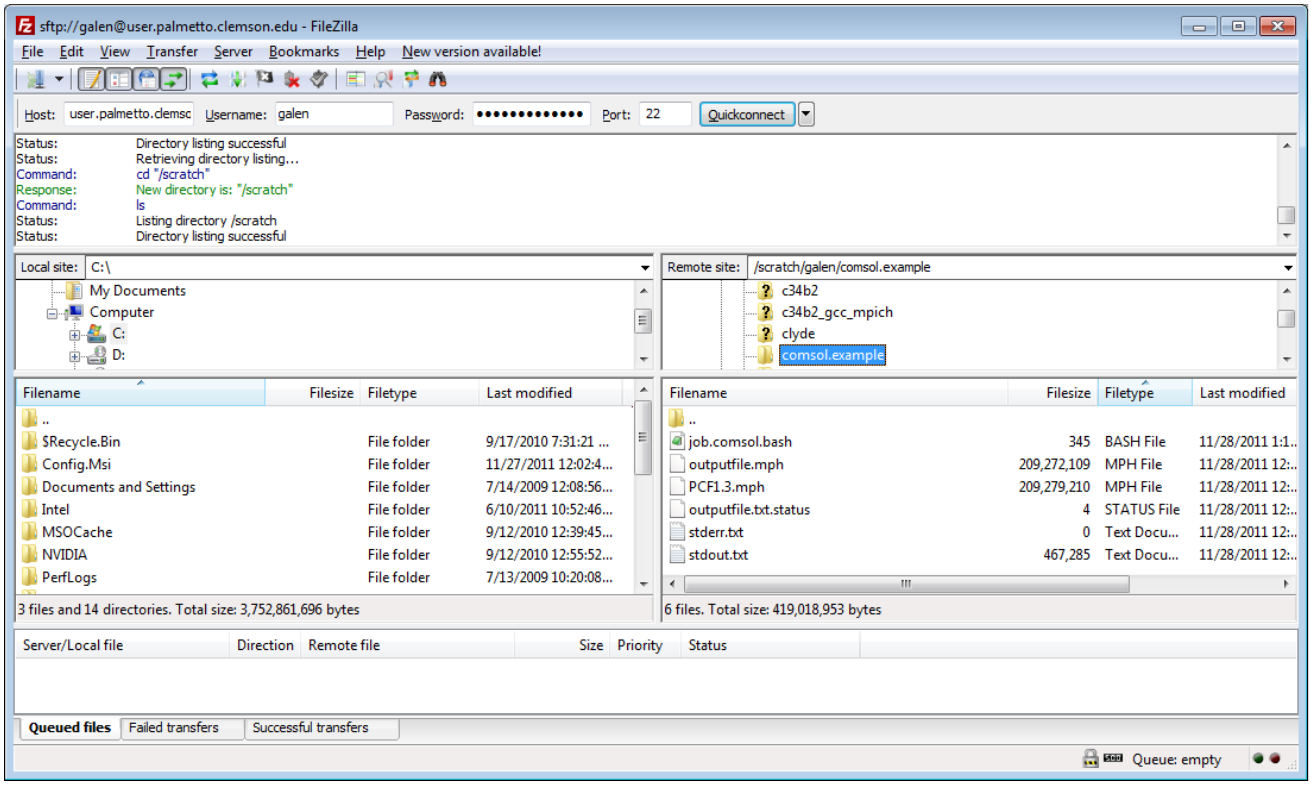
Important: Since COMSOL is running on a Palmetto compute node, browsing for files (using File > Open) will only allow you to access files that are located within Palmetto filesystems. If you have a file on your local workstation that you want to open with COMSOL running on Palmetto, you must transfer that file to Palmetto. Your /scratch directory might be the best place to put these files.

Transferring files to/from Palmetto:

FileZilla (http://filezilla-project.org/download.php) is recommended for moving files to and from Palmetto.

When connecting to Palmetto using FileZilla, you will use the same connection info that you use for the SSH Secure Shell client:

Host: user.palmetto.clemson.edu
Username: your Palmetto username
Password: your Palmetto password
Port: 22



Running COMSOL 4.2 batch jobs on Palmetto:

Below is an example PBS job script that can be used for running COMSOL 4.2 on one of the large shared memory nodes:

```
#!/bin/bash
#PBS -N COMSOL
#PBS -l select=1:ncpus=24:mpiprocs=24:mem=128gb
#PBS -q bigmem
#PBS -j oe
#PBS -l walltime=01:30:00

source /etc/profile.d/modules.sh
module purge
module add comsol/4.2

cd $PBS_O_WORKDIR

comsol batch -inputfile CMX-334-input.mph -outputfile CMX-334-output.mph
```

In this example, my job will run using 24 cores (the maximum available on one large shared memory node) and 128 GB RAM, and this job will run for a maximum of 1 hour and 30 minutes. I'm specifying that this job must run in the "bigmem" queue because of the large memory requirement.

Here, I'm adding the COMSOL 4.2 environment module in the PBS job script, and the **`cd $PBS_O_WORKDIR`** command ensures that my **`comsol`** command will be executed in the same directory where I issue the **`qsub`** command to submit this job to the scheduler.

Troubleshooting Common Problems

Error message: `qsub: Bad UID for job execution`
What it means: You're trying to submit a job from within another job (most likely an interactive session), or from a node that is not the user001 node.

Error message: `qsub: "-lresource=" cannot be used with...`
What it means: You have a typo in your qsub command (bad syntax, you might be using some of the old Torque syntax, or maybe you mixed up ":" and ",").

Error message: `mpiexec: Error: get_hosts: PBS reports fewer tasks...`
What it means: You're using /usr/bin/mpiexec and you should be using the mpiexec that comes with your MPICH module (just use 'mpiexec' instead of '/usr/bin/mpiexec').

Error message: `MX:node0589:recv req:req status 5:Data truncated due to undersized buffer MX:node0589:Remote endpoint is closed, peer=00:60:dd:46:f2:d3`
What it means: You didn't specify enough memory for your job, or you neglected to specify any memory requirement.

Error message: `-bash: /var/spool/PBS/mom_priv/jobs/633416[2].pbs01.SC: /bin/bash^M: bad interpreter: No such file or directory`
What it means: Look carefully. Do you see that `^M` character? That's a hidden character inserted into a job script because someone edited their script using Notepad, Wordpad, or Word on a Windows system, then uploaded the file to Palmetto. It is strongly recommended that all file editing be done on Palmetto (using a text editor like nano, vi, or vim). If you do run into this problem and need to convert a DOS file to a UNIX file, there is a utility called dos2unix that will remove these unwanted hidden characters. You can run this utility in this way: **`dos2unix script.sh`**