Contents

Foreword 0
Setting Up 1
  Quick Start for New HPC Users at NYU Langone Medical Center 1
  Requesting Access to the Cluster 1
  Setting Up the Client Computer 2
    Setting Up on a Computer Running a Unix-like Operating System 2
      Generating an SSH Key Pair on a Unix-like System 2
    Setting Up on a Windows Client 4
      Generating an SSH Key Pair Using PuTTYgen 4
        Configuring PuTTY 7
Connecting to the Cluster (“Logging In”) 11
  Logging In for the First Time 11
  Logging In from a Computer Running a Unix-like Operating System 11
  Logging In Using PuTTY 14
  Network Security Issues when Logging In 15
  Disconnecting from the Cluster (“Logging Out”) 16
Working with Linux 17
  Managing Directories 17
  Creating and Editing Text Files 20
    Using the Nano Editor 20
    Using the Emacs Editor 22
  Managing Files 23
  File System Permissions 24
    Sharing Files with Other Users 25
Environment Modules 26
  Using Environment Modules on the HPC Cluster 26
  Specifying Versions 27
  Prerequisites 27
  Conflicts 27
Submitting and Monitoring Batch Jobs 29
  The Batch System: Sun Grid Engine (SGE) 29
  Batch Job Directives 31
Cluster User Guide

Foreword

The High-Performance Computing Facility is a service offered by Medical-Center Information Technology (MCIT) to the NYU SOM/LMC research community. The guiding vision of the facility is to enable step changes in the quality and range of research through the use of advanced computing hardware, algorithms, and best-practices-driven integrative services. Established in 2009, the facility maintains state-of-the-art, locally installed high-performance computing (HPC) infrastructure with substantial computing and data storage capabilities. The main components currently are:

- A high-performance computing Linux cluster, nicknamed Phoenix, consisting of 2 head nodes and about 70 compute nodes, of which 5 are equipped with Graphics Processing Units (GPUs), and one is a high-memory node (1 TB of RAM).
- Two petabytes of scalable, redundant, high-bandwidth data storage capacity split into a primary cluster and a backup cluster.

The HPC Facility is a catalyst for advanced biomedical research, allowing NYU SOM/LMC researchers to undertake important groundbreaking projects in basic and translational research and train students in biomedical informatics algorithms and methods. The facility is at the epicenter of present and future growth in biomedical informatics enabling researchers of the NYU Cancer Institute—a National Cancer Institute-designated cancer center—to translate knowledge about the roots of cancer into innovative therapies and advanced cancer care. By providing access to data storage resources that are attached to a powerful compute cluster, and a complete software environment capable of executing sophisticated analysis of genomic data, the facility provides a computing infrastructure that is leading to the development of new algorithms, initiating molecularly-informed clinical trials, and making discoveries that will inform how we will better diagnose, treat and prevent disease in the future.

The purpose of this document is to help new users of the HPCF with limited knowledge in the area of HPC, Linux, and batch systems get set up and running quickly. We hope that it be useful in a way that new users can simply cut and paste some basic commands and quickly start exploring the full potential of the HPC resources available at NYULMC.

This guide is a dynamic document that changes quite frequently as the cluster expands and new applications are deployed. The latest version of this document is available online.

When publishing results gained by using Phoenix, please remember to acknowledge the HPC Facility (http://wiki.hpc.med.nyu.edu/w/index.php?title=HPCF_acknowledgement_statement).

The NYULMC HPCF Team

January 2017

Acknowledgments

The following persons contributed to the composition of this user guide.

Efstratios Efstrathiadis
Yasmine Kieso
Loren Koenig
Eric Peskin
Ali Siavosh

Cover Photos:

NYULMC Web Services & Digital Media (http://webservices.med.nyu.edu/)
Setting Up

Quick Start for New HPC Users at NYU Langone Medical Center

New users of the HPC Facility need to take the following steps. These are further described in the sections below.

- Use a computer with Linux/Unix, Mac OS X, or Windows.
- Make sure to have access to the MCIT network (wired, or wireless “LMC Mobile” network, or remotely via VPN).
- Request an HPC account.
- On a Windows computer: Install PuTTY.
- Generate a key pair.
- Have your public key deployed on the cluster.
- Log in.

Requesting Access to the Cluster

A high-performance computing (HPC) user account is required to access HPC resources at NYU Langone Medical Center. HPC user accounts are separate from what is often called the Kerberos ID, the centralized user accounts created by Medical-Center Information Technology (MCIT); instead, HPC user accounts are managed by the administrators of the High-Performance Computing Facility (HPCF). To apply for an account, please send an email to the HPC administrators (hpc_admins@nyumc.org) with the subject line “HPC Account Request”, including the following information in the body of the email:

1. the new user’s full name
2. the new user’s MCIT-assigned Kerberos ID (usually the user’s initials or part of the user’s name followed by a number of digits)
3. the new user’s email address (to receive important announcements via the HPCF mailing list)
4. the department/center of the new user and their division/section within there
5. the name of the new user’s PI
6. the title of the project for which HPC resources are to be used
7. a description of the research conducted by the new user (one short paragraph)
8. a description of the HPC needs such as data storage, compute power, data transfers, remote access (from outside the MCIT network), etc. (one short paragraph)
9. the new user’s SSH public key (can alternatively be sent in a separate email; see also the instructions for generating a key pair on Unix-like systems and on Windows in the following subsection)

Notes:

- Requests for multiple accounts can be sent in a single email.
- Accounts are usually created within a business day once the request has been received and possible questions are answered. The HPC administrators will notify the requester about the account creation.

Upon request, a consultation for a prospective user can be arranged. The initial consultation is free and can be done over the phone. Possible topics include

- overview of the available resources and services,
- best practices on accessing and using the resources,
- user responsibilities and costs associated with HPC accounts, as well as
- any questions the prospective user may have, in particular concerning account creation/generating SSH keys.
Setting Up the Client Computer

Access to the cluster requires a computer running a Unix-like operating system or Microsoft Windows. This computer, called client, needs to be set up before establishing a connection to the cluster from this computer for the first time. The steps to be taken depend on the operating system on the client computer and are detailed in the following subsections.

Setting Up on a Computer Running a Unix-like Operating System

To connect to the head node of the cluster, phoenix.med.nyu.edu, from a Linux/Unix/Mac OS X computer, the user needs to first generate an SSH key pair. Linux/Unix/Mac OS X hosts have built-in SSH client software, so no additional software or SSH clients are needed.

Generating an SSH Key Pair on a Unix-like System

Try this: In Linux/Unix, open Terminal. In Mac OS X, the Terminal is located in your Applications/Utilities folder. A new terminal window will appear on the screen. At the command line in the terminal window, type: `ssh-keygen`, followed by pressing the `Enter` or `Return` key.

You will then see a reply line in the terminal stating that it has begun generating the keys. Once the pair of keys have been generated, the next line in the Terminal will ask where to save the key (labeled "1" in the figure below). At that point, just press `Enter` to accept the default file location in which the keys will be saved. (The default location of the public and the private key is in the directory: `~/.ssh/`)

The next terminal prompt will ask for a passphrase. Type a passphrase of your choice that is easy for you to remember but difficult for others to guess, and then press `Enter`. When prompted, re-type the same passphrase (labeled "2" in the figure below). Note that no indication of your typing will appear on the screen to protect your passphrase from curious onlookers.

The Terminal will reply with several lines indicating:

- The file name of the private key is `id_rsa`.
- The file name of the public key is `id_rsa.pub`.
- The key fingerprint

On the terminal, this step should look like the following:
Send the contents of the public key file (*id_rsa.pub*) in an email to the HPC administrators ([hpc_admins@nyumc.org](mailto:hpc_admins@nyumc.org)). (See label “3” in the figure above.)

Once you receive notification that your account is set up, you are ready to log in and use the cluster.
Setting Up on a Windows Client

To connect to the head node of the cluster, phoenix.med.nyu.edu, from a Windows computer, you will need to install an SSH client. A popular SSH client for Windows is PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/). It is preferable to download the “PuTTY Windows Installer” executable (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) because it will automatically install all binaries, including PuTTY and PuTTYgen.

Generating an SSH Key Pair Using PuTTYgen

Try this: To generate an SSH key pair you will need to run PuTTYgen, a utility program that comes with PuTTY. If you chose the option to make a desktop icon during the installation, you can start up PuTTYgen by double-clicking the PuTTYgen desktop icon. If you used the PuTTY installer, but did not choose to create desktop icons, you will find PuTTYgen by going to your Start menu, selecting All Programs, then opening the PuTTY folder where you’ll be able to click and open PuTTYgen.
In the PuTTY Key Generator window that pops up, select the RSA key type (or, in older version of PuTTYgen as in the figure below, “SSH-2 RSA”—either way, do not select any SSH-1 key type) and then click Generate. You will need to move the cursor around in the window to generate randomness.
Once the key pair is generated, a new window will pop up displaying the public key, key fingerprint, a blank field to enter a key comment, two blank fields for the key passphrase and options for actions and parameters.
Follow these important steps:

1. Enter and confirm a key passphrase. It is important to select a passphrase that it is easy for you to remember but also hard for others to guess. **Keep the passphrase secure and private. Never tell or send it to anyone. Similarly, never send your private key to anyone.**

2. Click on “Save public key” and then click “Save private key” to save the public and private keys. Save these files on your local hard drive. Remember where you saved them.

3. Finally, send an email containing the generated public key to the HPC administrators at hpc_admins@nyumc.org and requesting your key be deployed on the cluster. (Note the underscore “_” between “hpc” and “admins”). To do this, copy and paste the contents of the public-key field of the above window directly into the body of an email. Wait to hear back from the HPC administrators that your key has been deployed on the cluster before you attempt to log in to the cluster head node.

**Configuring PuTTY**

It is convenient to configure PuTTY before logging in for the first time. Doing so means that PuTTY will be prepared for all subsequent connections to the cluster, instead of the user having to enter the connection settings every time he or she wants to log in.

**Try this:** You can start PuTTY by double-clicking the desktop icon or by going to your Start menu, clicking on ‘All Programs’, then selecting the PuTTY folder and finally the PuTTY program. The first dialog box that pops up on your desktop will be titled the "PuTTY Configuration".
In this box, under the section labeled 'Category' on the left, make sure that 'Session' is selected. Then do the following:

1. To the right of the 'Category' section, in the 'Host Name (or IP address)' field, type 
   $user@phoenix.med.nyu.edu$ where $user$ stands for your user name which is your Kerberos ID unless you have been given a different user name when your account was created.

2. Select “SSH” under Connection Type.

3. In the 'Category' tree view, expand the node ‘SSH’.

A subtree becomes visible underneath 'SSH' in the 'Category' tree view.
1. Select 'Auth'.
2. To the right of the tree view, click the boxes to check 'Display pre-authentication banner (SSH-2 only)', check 'Attempt authentication using Pageant', and check 'Attempt "keyboard-interactive" auth (SSH-2)'. Make sure that 'Bypass authentication entirely (SSH-2 only)' is not checked. (Note that depending on which version of PuTTY you're using, the order of the checkboxes may differ from the one shown in the figure.)
3. Under "Private key file for authentication", click on 'Browse' and select the private key file that you saved from PuTTYgen.

Now go back to the Category list and carry out the following steps.
1. Select “Session”. The right side of the dialog window will change to match the above figure.

2. In the field underneath the label "Saved Sessions", type a name for these settings, e.g. Phoenix.

3. Select “Save”.

Once you have saved these configuration settings, you can reload these settings again the next time you connect to the cluster.

With PuTTY configured, the setup of a Windows machine is complete and ready for connecting to the cluster, as described in the section Logging In Using PuTTY.
Connecting to the Cluster ("Logging In")

Logging In for the First Time

When logging into a remote server like the head node of Phoenix for the first time, the local SSH client ‘doesn’t know’ the server yet. It therefore usually asks the user to establish the server’s authenticity by comparing the server’s fingerprint to a known value. If the user confirms, the SSH client ‘gets to know’ the server by recording the server’s fingerprint so it won’t have to ask the user before subsequent log-in attempts.

Hence, when asked whether to trust a server that is supposed to be Phoenix, it is important to make sure the server presents one of the following fingerprints:

- 66bt6kuSe/Kp18G//cNpNLo0cYJx8Wbz5Sdjw16oZM4
- BNVVvGXPJSbr54LlIa/NcEkoGSLkJVNdZ0zbqAIwE

If a server presents any of the above fingerprints, it can reasonably be assumed to indeed be Phoenix.

The next sections describe the actual log-in procedure depending on the operating system running on the local machine.

Logging In from a Computer Running a Unix-like Operating System

Try this: Open a new Terminal window. At the command line enter

```
ssh user@phoenix.med.nyu.edu
```

where `user`, your user name, is your Medical Center Kerberos ID (except if you have been assigned a different user name when your account was created).

If you are using Mac OS X, you may be prompted for the passphrase, although the actual wording may be "password for the key":

---
If this happens, type your passphrase in the field labeled "Password" and click "OK".
The terminal should then look like this:

```
[user@personal-computer ~]$ ssh user@phoenix.med.nyu.edu
Last login: Fri May 17 16:54:46 2013 from your machine
Welcome to Bright Cluster Manager 6.0

Based on CentOS release 6
Cluster Manager ID: #000002
```

Use the following commands to adjust your environment:

- `module avail` - show available modules
- `module add <module>` - adds a module to your environment for this session
- `module initadd <module>` - configure module to be loaded at every login

```
[ user@phoenix1 ~]$
```

If this is the case, you are ready to begin submitting jobs on the cluster.

If instead, within the terminal window, you are prompted for a password as in the following picture, something went wrong.
In this case, make sure you specified the right user name (for most users: the Kerberos ID). If you do not find anything wrong notify the HPC administrators (hpc_admins@nymc.org).
Logging In Using PuTTY

Having configured PuTTY, you can use it to log in to Phoenix. Start PuTTY by double-clicking the PuTTY desktop icon (or selecting its Start Menu entry). The PuTTY Configuration window appears.

1. Select the previously saved session that you made using your private key and the host name user@phoenix.med.nyu.edu.
2. Click the Load button and verify the host name is correct.
3. Click Open.

A terminal session window will appear. This should look similar to the following example in which the user demo is logging in.
In the line asking for your passphrase, type the passphrase you specified in the key generation step, and then press Enter on your keyboard. Note that no indication of your typing will appear on the screen to protect your passphrase from curious onlookers.

If you get prompted for a password instead of a passphrase, then something went wrong. First, make sure you specified the right user name (for most users: the Kerberos ID). Then double-check the path to your private key. If you do not find anything wrong notify the HPC administrators (hpc_admins@nyumc.org).

Otherwise, you are now logged in and connected to the front-end node and ready to begin submitting jobs.

**Network Security Issues when Logging In**

Apart from the log-in issues described above, a number of problematic situations can arise when attempting to log in. These situations are usually related to network security. Any of the following should therefore immediately be reported to the HPC administrators (hpc_admins@nyumc.org):

- The SSH client reports a key that *does not match* its earlier records. This may be indicative of a man-in-the-middle attack or other security breach.
- The SSH client asks to confirm the authenticity of the server's key even after previously having successfully logged in from the same computer. This may mean that the local computer has been compromised.
Disconnecting from the Cluster ("Logging Out")

Whenever work on the head node is done, log-in sessions must be terminated manually. This is called "logging out" and can be done by typing `logout`, followed by pressing Enter. When using PuTTY, the terminal window usually closes automatically on successful logout.
Working with Linux

The Phoenix cluster runs CentOS Linux as an operating system. The user interacts with the system through the command line, a text-based command-and-response mechanism. Basic commands allow the user to manage data stored in files and directories, edit files, as well as submit and monitor compute jobs.

Side Note: There are numerous introductory Linux tutorials available on the web. One such tutorial is the Cornell Virtual Workshop (https://www.cac.cornell.edu/VW/Linux/default.aspx?id=xup_guest). The High-Performance Computing Facility maintains a list of useful training resources and tutorials (http://wiki.hpc.med.nyu.edu/w/index.php?title=online_resources).

The following sections give a quick walk-through of the most important Linux basics.

Managing Directories

A fundamental concept in Linux (as well as in other Unix-like operating systems) is the file system. It consists of files, which are containers for data, and directories, which act as containers for any number of files and other directories. The only element of the file system not contained in anything else is the root directory, denoted by a single slash. In a Linux log-in session, at any point in time, exactly one directory is the current directory or working directory. This is often referred to as ‘being in [that directory]’.

Side Note: On a Windows system, folders would be the equivalent of Linux's directories.

The following tree diagram shows a part of the file system which is common to all nodes of the Phoenix cluster. The directory highlighted in yellow is the working directory right after logging in, called the user’s home directory. (Note that the home directory is not the same as the directory called home.)
Note how for each directory, the \textit{absolute path}, given in parentheses, can be obtained by prepending its name with the names of the \textit{containing} directories, starting with the root directory. This way, any element (or directory, for that matter) in the file system can be unambiguously referred to by its absolute path. Also note how absolute paths always start with a slash.

On the other hand, a \textit{relative path} points to a file system element relative to ("starting from") the current directory. A relative path never has a slash as its first symbol.

We will see and make use of absolute and relative paths shortly.

\textbf{Try this:} The \texttt{pwd} command tells you which directory you are currently in. For instance, when you have just logged in to the Phoenix head node, you are in your home directory.

To find out which directory is the working directory, simply issue the \texttt{pwd} command, that is, in the command line, type \texttt{pwd} and press \texttt{Enter}.

\begin{verbatim}
[user@phoenix1 ~]$ pwd
/ifs/home/user
\end{verbatim}

The \texttt{pwd} command replies by printing the absolute path of the working directory.

Now, in your home directory, create a new directory, called \texttt{project1}. To do so, you can use the command \texttt{mkdir}, for "make directory". The \texttt{mkdir} command takes as an \textit{argument} the path of the directory to be made. In this case, since you already are in your home directory (see the output of the \texttt{pwd} command above), you can use a relative path which consists of just the name of the new directory:

\begin{verbatim}
[user@phoenix1 ~]$ mkdir project1
\end{verbatim}

Upon successful creation of the new directory, the \texttt{mkdir} command does not produce any output. To see what it did, you can use another command, called \texttt{ls}, short for "list". The \texttt{ls} command prints the contents of the working directory:

\begin{verbatim}
[user@phoenix1 ~]$ ls
project1
\end{verbatim}

As you can see, the working directory (which should still be your home directory) now contains \texttt{project1}. The below figure highlights in red the new directory called \texttt{project1}: 

---
To navigate to other directories within home, you can use the command `cd`, short for "change directory". Try navigating to the directory `project1` by entering in the command line `cd project1`.

```
[user@phoenix1 ~]$ cd project1
```

To check your current directory and confirm you are in `project1`, enter `pwd` (short for "print working directory"):  

```
[user@phoenix1 project1]$ pwd
/ifs/home/user/project1
```

Navigate back to your home directory by entering the command `cd` without any arguments:

```
[user@phoenix1 project1]$ cd
[user@phoenix1 ~]$ pwd
/ifs/home/user
```

In the following, we summarize some rules about paths:

- A path beginning with slash `/` is an **absolute path**.
- `~` (a tilde) is a shortcut for home directory of the user that is currently logged in.
- `~user` is a shortcut for user's home directory.
- A path that does not begin with a slash `/` or a tilde `~` is a **relative path**.
- `.` (a single dot) refers to the current directory.
- `..` (two dots) refers to the parent directory (the directory that contains the current one).
- `/` (a slash) separates directories in a path.

Note that, like any file system identifiers on Linux systems, paths are case-sensitive.
With regard to the `cd` command, the following shortcuts are useful:

- The `cd` command without any argument changes the working directory to the current user's home directory.
- `cd ~` goes to the previous directory.

### Creating and Editing Text Files

While files essentially are containers for arbitrary data, an important kind of files are *text files*, holding human-readable text. (Text files are typically considered to include script files, which are in a way both human- and machine-readable, and which are covered later in this manual.)

Text files are created and manipulated using *text editors*—programs allowing for the user to enter printable characters (letters, digits, spaces, punctuation, *etc.*), composing them into words and lines, modifying the text, and saving it into files.

Under Linux, text editors come in a wide variety of flavors and with a wide variety of learning curves; we are going to introduce two of them in the following subsections: *Nano* is easy to master, but limited, whereas *Emacs* meets even advanced users’ requirements, but may not be the best choice for beginners.

### Using the Nano Editor

Nano is a simple text editor well suited to beginners’ needs (but in turn with some severe limitations making it an inept tool for advanced users). We will be using it in the exercise below to give a most gentle introduction to text editing on the Linux command line.

**Try this:** Make sure you are in the `project1` directory by entering `pwd`. If you are not, enter `cd ~/project1` to navigate into `project1`. Once you are there, we will create a text file called `first` in the `project1` directory using Nano. To do so, type `nano first` on the command line and press `↵ Enter`:

```
[user@phoenix1 ~]$ pwd
/ifs/home/user
[user@phoenix1 ~]$ cd project1
[user@phoenix1 project1]$ nano first
```

Your terminal should then look like the following figure; if so, you can type right away to enter the contents of your file. For practice, type *This is a simple file.*
To edit, use the arrow keys to move the cursor and ← Backspace to delete characters to the left of the cursor as needed. Press Home or End to move the cursor to the beginning or end, respectively, of the current line. (If that doesn't work, use Ctrl+A and Ctrl+E, respectively.)

When in Nano, note the two lines on the bottom of your terminal that read

```
^G Get Help   ^O WriteOut   ^R Read File ...
```

and so on. These refer to keyboard commands, where the caret (^) indicates a key combination with the Ctrl key. So in order to save your file, press Ctrl+O. Nano will ask for a name under which to save the file, but the previously chosen name first will be the default, so you can just press Enter.

To exit from Nano, press Ctrl+X (and, if modified, answer the question whether you’d like to save your file by pressing Y or N). Once you exited from Nano, you should see a command prompt again.

If you want to just view, but not edit, the contents of a file, you can use the cat command:

```
[user@phoenix1 project1]$ cat first
This is a simple file.
[user@phoenix1 project1]$
```

In order to pick up editing again of that same file, just restart Nano giving the name of the file:

```
[ demo@phoenix1 project1]$ nano first
```

To leave Nano again, press Ctrl+X as described above.
Using the Emacs Editor

Emacs is a text editor that can be used to create and edit text files. The exercise below teaches you how to create a simple text file using emacs.

Try this: Make sure you are in the project1 directory by entering `pwd`. If you are not, enter `cd ~/project1` to navigate into project1. Once you are there, we will create a text file called `first` in the project1 directory using emacs. To do so, type `emacs first` into the command line and press Enter:

```
[user@phoenix1 ~]$ pwd
/ifs/home/user
[user@phoenix1 ~]$ cd project1
[user@phoenix1 project1]$ emacs first
```

Depending on whether your local computer and your login session to Phoenix support the X Window System, the user interface of Emacs will appear in your terminal window or in a new window like the one below—either way the following example should work, as the keyboard commands are the same in each case.

Once the emacs editor has started, you can start typing the contents of the file. For practice, type `This is a simple file`. Save the file by pressing the `Control+X` key combination, followed by `Control+S`. When you are finished editing the text file, you can exit emacs by pressing `Control+X, Control+C`.

An emacs tutorial is also available within emacs. Start emacs:

```
[user@phoenix1]$ emacs
```

Press `Control+H`, then press `T`. Read and follow the instructions on the screen to navigate through the tutorial.
Once you have exited the emacs session, you will be back at the command line. To re-open the file you created (named first) and continue adding or editing text, you can simply enter `emacs first` in the command line:

```
[user@phoenix1 project1]$ emacs first
```

To view the contents of the file without editing text, enter `cat first` in the command line.

```
[user@phoenix1 project1]$ cat first
This is a simple file.
```

The `cat` command stands for 'concatenate' and displays all file contents. If you enter this command with the file name(s), and press ↵ Enter, you will see the text in each file. Ex.: `cat file1 file2 file3`

### Managing Files

Files are named containers for data. In the above section's example, we used emacs to put a short text in a file named first. The Linux command line has, amongst others, commands to copy, move, delete, and rename files.

**Try this:** In the `project1` directory, make a copy of the file `first`. You can use the command `cp`, short for "copy", which takes two arguments: the `source` (the path of the file to be copied) and the `destination` (the path of the copy to be created or of the directory in which to put the copy). The name of the copy must be different from that of the source, so pick a new name, for example `first-new`. Then use the `ls` command to check on the result:

```
[user@phoenix1 project1]$ cp first first-new
[user@phoenix1 project1]$ ls
first     first-new
```

Use the `cat` command to make sure the file `first-new` actually has the same contents:

```
[user@phoenix1 project1]$ cat first-new
This is a simple file.
```

Now say you want to move the new file to your home directory, maybe because it applies to projects other than project 1, too. The command to do so is the `mv` command, short for "move". Similar to `cp`, it takes the two arguments source and destination, but it moves source to destination instead of copying it.

To move a file to another directory, you can specify that directory as the destination, in which case `mv` will leave the name of the file unchanged. Since in this example, your home directory directly contains `project1`, where the file to be moved is currently located and which is also the working directory, you can refer to the home directory as the parent directory, denoted by the relative path `..`:

```
[user@phoenix1 project1]$ mv first-new ..
[user@phoenix1 project1]$ ls
first
```

Note how `first-new` doesn't show up in the output of `ls` any more. However, if you go to the parent directory (which in this case happens to be your home directory) using the `cd` command, you should be able to find `first-new`:

```
[user@phoenix1 project1]$ cd ..
[user@phoenix1 ~]$ ls
first-new     project1
```
In case you want to rename a file but not move it to another directory, you can do that with the `mv` command, too:

```
[user@phoenix1 ~]$ mv first-new first-try
[user@phoenix1 ~]$ ls
first-try    project1
```

Note how the output from `ls` reveals the file under its new name.

If you decide you don't need the file any more (and are really sure about it), the command `rm`, short for "remove", can be used. It *irrevocably* deletes the file(s) denoted by the argument(s) given to it. On Phoenix, before actually deleting anything, `rm` gives users a last warning. There, you have to enter `y` to delete the file:

```
[user@phoenix1 ~]$ rm first-try
rm: remove regular file `first-try'? y
[user@phoenix1 ~]$ ls
project1
```

### File System Permissions

Unix-like operating systems control who has what kind of access to files and directories. Traditionally, and also on Phoenix, this is implemented in the following way:

- Each file or directory is assigned to exactly one user (also called the owner) and exactly one group.
- Access is controlled separately for three categories of people: the user, the group, and others.
- There are three types of access: Read, Write, and Execute.

To view the permissions on a file you can add the `-l` flag to `ls`, for example `ls -l script.sh` to view the permissions on the file `script.sh`. The output might look like the following:

```
-rwxr-x--- tangz01 illuarch Jan 1 1970 script.sh
```

This is to be interpreted as follows:
Sharing Files with Other Users

To share files and directories with some users, you need to change their group to a group that comprises these users and then assign the appropriate group permissions. You could, for example, do this for a subdirectory of /ifs/data/labname/.

First, you need to identify the right group of users. To just find out who is in a given group, enter `getent group groupname` on the command line. To find out what groups you are in yourself, enter `groups`. To find out what groups a given user is in, enter `groups username`. This way, you may arrive at a group that comprises all the users with whom you intend to share the data but nobody whom you’d rather not have in that circle. If you do not find such a group, consult with the HPC administrators (hpc_admins@nyumc.org) as needed to have a group with the desired members created.

Suppose you have an existing subdirectory of your lab directory and that the subdirectory is /ifs/data/labname/x/y/z/ and you want to make everything under /ifs/data/labname/x/y/z/ readable by the group. You could run:

```bash
cd /ifs/data/labname/x/y/z/
chgrp -R groupname .
chmod -R g=rX,o= .
find . -type d -print0 | xargs -0 chmod g+s   # See note 1 below
find . -ls | awk '{print $3,$5,$6}' | sort -u # See note 2 below
chmod o=x /ifs/data/labname{,/x{,/y}}         # See note 3 below
```

1. The `s` bit on a directory makes sure that new files created underneath that directory will automatically get the group of the parent, in this case `groupname`.
2. This command finds all the unique combinations of permissions, owner, and group in a directory tree. In this case, you would expect to see, for example:

   ```text
drwxr-s--- user groupname
-rw-r----- user groupname
-rwxr-x--- user groupname
```

   So everything is owned by the user `user`, and everything has the group `groupname`. Directories (as indicated by the `d` as the first character of the line) are readable and executable by the group (and only writable by `user`), and have the `s` bit set (see note 1). Files are readable by the group, only writable by you, and in some cases executable by both you and the group. Others (outside of the group) have no permissions.

3. Let’s assume that you do not want to share everything in the lab directory or `x` or `y`, but you do want to share everything in `z`. To get at anything in a Unix-like file system, you need at least execute permission on all the ancestor directories. One way to do this is to give “others” (o) execute permission (and only execute permission) on the ancestors, namely `/ifs/data/labename,/ifs/data/labename/x`, and `/ifs/data/labename/x/y` in this example.

For details, see the man pages for `chmod`, `chgrp`, `find`, and `xargs`.
Environment Modules

The HPC cluster uses a system called Environment Modules (http://modules.sourceforge.net/) to manage applications. Each environment module has the information necessary to configure the user's environment for a particular application. From a user's point of view, *loading a module* makes a specific application available for use on the cluster for the duration of the log-in session or the duration of a script. Modules make sure that the user's environment variables are set up for the desired software, and that any prerequisite software—but no conflicting software—is available.

Using Environment Modules on the HPC Cluster

Here is a simple example session from a user looking for the application `bwa`.

```
[ep599@phoenix1 ~]$ bwa
-bash: bwa: command not found
[ep599@phoenix1 ~]$ module avail bwa

---------------------------- /local/modulefiles ----------------------------
bwa/0.7.3a bwa/0.7.5a
[ep599@phoenix1 ~]$ module load bwa/0.7.5a
[ep599@phoenix1 ~]$ bwa

Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.5a-r405
Contact: Heng Li <lh3@sanger.ac.uk>
...
```

The main commands related to environment modules are:

- `module avail` shows all the modules that are available on the cluster.
- `module avail name` shows just the modules that are available for the application named `name`.
- `module list` shows the modules that you have loaded at the moment.
- `module load name` adds the module named `name` to your environment.
- `module unload name` or `module rm name` removes the module named `name` from your environment.
- `module help name` shows any module-specific help.
- `module show name` gives a one-line description of the module and shows what it will do to your environment.
- `man module` provides details about the module command and all its subcommands.

**Note:** It is best to load modules at the beginning of each script that needs them. This is shown in the section on Submitting and Monitoring Batch Jobs.
Specifying Versions

If you do not specify a particular version for a module, the default version will be loaded. Generally, the default will be the newest installed version. If you need a specific version (from among those installed), you can specify the version, for example, `module load gcc/4.4`. It is generally advisable to specify the exact version of a module to avoid breaking functionality as new versions of modules are installed.

Prerequisites

Modules can specify other modules as prerequisites. In most cases, we configure each module to load its prerequisites when it loads, and hence unload them when it unloads. For example, the casava module depends on `gcc` version 4.4. Thus, we get the following behavior:

```
[user@phoenix1 ~]$ module list
Currently Loaded Modulefiles:
1) sge/2011.11  2) local  3) default-environment

[user@phoenix1 ~]$ module load casava
[user@phoenix1 ~]$ module list
Currently Loaded Modulefiles:
1) sge/2011.11  3) default-environment  5) casava/1.8.2
2) local  4) gcc/4.4

[user@phoenix1 ~]$ module unload casava
[user@phoenix1 ~]$ module list
Currently Loaded Modulefiles:
1) sge/2011.11  2) local  3) default-environment

Bowtie, Bowtie2, and TopHat constitute an interesting special case. Bowtie and Bowtie2 are considered separate applications — not just different versions, so we have separate modules for them. TopHat2 can use either one. You have to load at least one of the Bowtie modules before loading TopHat. You can load them both and then use TopHat's command-line switches to switch if you want.

Conflicts

Modules can conflict with each other. The most common case is a module conflicting with other versions of the same tool. For example, two different versions of `gcc` conflict with each other:

```
[ep599@phoenix1 ~]$ module load gcc/4.4
[ep599@phoenix1 ~]$ module load gcc/4.7.0
gcc/4.7.0(5):ERROR:150: Module 'gcc/4.7.0' conflicts with the currently loaded module(s) 'gcc/4.4'
gcc/4.7.0(5):ERROR:102: Tcl command execution failed: conflict gcc/4.4
[ep599@phoenix1 ~]$ module unload gcc/4.4
[ep599@phoenix1 ~]$ module load gcc/4.7.0
[ep599@phoenix1 ~]$ module list
Currently Loaded Modulefiles:
1) sge/2011.11  3) default-environment
2) local  4) gcc/4.7.0
```
As a shortcut to the unload...load sequence, there is a module switch old new command:

```
[ep599@phoenix1 ~]$ module load gcc/4.4
[ep599@phoenix1 ~]$ module list
Currently Loaded Modulefiles:
 1) sge/2011.11           3) default-environment
 2) local                 4) gcc/4.4
[ep599@phoenix1 ~]$ module switch gcc/4.4 gcc/4.7.0
[ep599@phoenix1 ~]$ module list
Currently Loaded Modulefiles:
 1) sge/2011.11           3) default-environment
 2) local                 4) gcc/4.7.0
[ep599@phoenix1 ~]$ 
```
Submitting and Monitoring Batch Jobs

The HPC cluster (Phoenix) consists of a head node and multiple (about 70) compute nodes. Many users share the cluster. If everyone just runs programs directly on the head node, the head node will be overloaded and the compute nodes will be idle. There needs to be some way of managing the workload and distributing work across the compute nodes. Rather than having users do this manually by hunting for compute nodes that are not too busy, Phoenix uses an automated batch system to manage the workload.

To run a job on the cluster, you need to create a shell script with the commands needed to perform your task or execute your workflow. Once you submit this script to the batch system, the batch system finds a compute node that is not too busy and has appropriate resources and runs your job there. Your job runs unattended in the background on the compute node. You can log out of the cluster while your job is running. Once the job is finished you can inspect the output in files.

Shell scripts are important in their own right as a way to automate and document recurring complex tasks. We introduce them here in the context of the batch system.

The Batch System: Sun Grid Engine (SGE)

The batch system that we use on Phoenix is called Sun Grid Engine (SGE) (http://gridscheduler.sourceforge.net/). To familiarize yourself with using SGE, try to copy an example script (simple.sh) to your home directory and have a look at its contents:

```
[user@phoenix1 ~]$ cd
[user@phoenix1 ~]$ cp /ifs/data/tutorials/userguide/simple.sh .
[user@phoenix1 ~]$ cat simple.sh
#!/bin/bash
#$ -S /bin/bash
#
# A simple SGE script
# Adapted from Sun Microsystems
#
# print date and time
date
# Sleep for 20 seconds
sleep 20
# print date and time again
date
```

The command `cd` (without any arguments) takes you to your home directory, and the single dot . in the second command refers to the current directory as the target directory for the file to be copied. Finally, the `cat` command shows the contents of the script file.

Since this script does almost nothing (nothing in terms of consumption of resources, that is; it prints the date twice and sleeps in between), it can safely be run on the head node. However, note that this is a special case, and that scripts should...
in general not be run on the head node. To run the script, you have to prepend its name with ./ to indicate that it is located in the current directory, and enter it at the command line:

[user@phoenix1 ~]$ ./simple.sh
Tue Oct 22 17:35:18 EDT 2013
Tue Oct 22 17:35:38 EDT 2013

Now instead of running the simple.sh script on the head node, try submitting the script using the command qsub:

[user@phoenix1 ~]$ qsub simple.sh
Your job 2141370 ("simple.sh") has been submitted

The output of the qsub command contains the job id (in the above example 2141370). Right after submitting the job, try the qstat command (without any arguments) to check the status of your job. The output of qstat varies depending on the status of your job:

Your job 2141370 ("simple.sh") has been submitted
[user@phoenix1 ~]$ qstat
job-ID  prior   name      user state submit/start at     queue                       slots
------------------------------------------------------------------------------------------
2141370 0.50500 simple.sh user r     07/09/2012 16:15:56 regular.q@compute-1-4.local 1

You may enter qstat in the command line a few times to track your job progress from qw (waiting) to r (running) to not listed at all (job done).

Once the job is done, look for output and error files. By default each submitted job will generate an output and an error file. The name of the output file ends in .o<job_id>. The name of the error file ends in .e<job_id>.

[user@phoenix1 ~]$ qstat
[user@phoenix1 ~]$ ls -ltr simple.sh.*
-rw-r--r-- 1 user user  0 2012-07-09 16:15 simple.sh.e2141370
-rw-r--r-- 1 user user 58 2012-07-09 16:16 simple.sh.o2141370

The -l flag to ls species long-form listing. This shows permissions, owner, group, size, and last modified time for each file. In particular, the 0 shows us that the error file simple.sh.e2141370 is empty. That is a good sign. The -t (time) and -r (reverse) flags to ls cause the files to be listed with the most recently modified files at the bottom. This makes it easy to find the new output and error files, even in a directory with many files.

Examine the contents of the job output and job error files. The above listing shows that the error file is empty. But you can see the contents of the output file with cat simple.sh.o2141370. This should contain the output of the script.

In the case of simple.sh we expect the output to consist of two dates separated by 20 seconds.

The HPCF website has some more information to help you get started (http://wiki.hpc.med.nyu.edu/w/index.php?title=online_resources). The links there can help you with Unix/Linux and also with SGE. Note that most of those links point to other sites developed by other people. So some of the details in the examples (like the names of clusters, machines, and queues) may be different; however, the main concepts will be the same, so the tutorials can still be helpful.

Also, the Open Grid Scheduler documentation (http://gridscheduler.sourceforge.net/documentation.html) is largely applicable to SGE as well.

Note: Please remember to not run compute-intensive jobs directly on the front-end node. Instead, submit them as batch jobs to Sun Grid Engine (SGE). SGE will schedule jobs for execution on the compute nodes.
Batch Job Directives

There are many options that can alter the behavior of SGE and how it runs each job. There are several places where SGE directives can be placed. These are as follows, in the following order of precedence:

1. On the command line to SGE commands such as qsub.
2. Embedded in a job script on a line with #$.
3. In a file named .sge_request in the directory in which you run qsub.
4. In a file named .sge_request in your home directory (~/.sge_request).

Some useful batch job directives to try are the following:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-cwd</td>
<td>Use the current working directory (execute job in same directory in which qsub was run).</td>
</tr>
<tr>
<td>-S /bin/bash</td>
<td>Specifies which shell to use.</td>
</tr>
<tr>
<td>-N JobName</td>
<td>Gives a name to the job. The name will appear in the output of qstat.</td>
</tr>
<tr>
<td>-M <a href="mailto:firstName.lastName@nyumc.org">firstName.lastName@nyumc.org</a></td>
<td>Specifies an email address where notification messages will be sent. See notes on email below.</td>
</tr>
<tr>
<td>-m abe (or any subset of a, b, and e)</td>
<td>Specifying when an email will be sent: a — abort, b — begin, e — end. See notes on email below.</td>
</tr>
</tbody>
</table>

Notes on email:

1. Do not use an external email address. Use @nyumc.org or @med.nyu.edu addresses.
2. The uppercase -M setting (which sets your email address) is best placed in your ~/sge_request file. Your email address will be the same for all your jobs. Furthermore, if you share a script with someone else, notices should go to the person who submits the jobs.
3. Use the lowercase -m flag sparingly! SGE makes it easy to submit many jobs simultaneously. These can result in many email messages, which can flood your email box and the network and can also look like a spambot to MCIT’s security programs. If you submit many jobs, either with a loop that runs qsub, or even with an array job, do not use -m. Only use -m when submitting a single job and when you really need email notification.

Let’s use emacs to modify the simple.sh script to add some commonly used directives to the job file.

```bash
[user@phoenix1 ~]$ emacs simple.sh
```

Use one directive per line and start each line that contains an SGE directive with a pound sign, dollar sign, and a space: "#$". Do not insert anything before the initial #!/bin/bash at the top of the script. That always needs to be the first line. An example of the finished script is below.

Once you have finished entering the directives into the file using Emacs, save the file by pressing Control+X,Control+S, then exit Emacs by pressing Control+X,Control+C. Use the cat command to check on the contents of the file:

```bash
[user@phoenix1 ~]$ cat simple.sh
#!/bin/bash
#$ -S /bin/bash
#$ -N SimpleTest
#
# A simple SGE script
# Adapted from Sun Microsystems
#
# print date and time
```
date
# Sleep for 20 seconds
sleep 20
# print date and time again
date

To submit the job simple.sh with the directives you entered, use qsub simple.sh in the command line. Then, check on your job and its directives entering qstat at the command line.

[user@phoenix1 ~]$ qsub simple.sh
Your job 2146096 ("SimpleTest") has been submitted
[user@phoenix1 ~]$ qstat

<table>
<thead>
<tr>
<th>job-ID</th>
<th>prior</th>
<th>name</th>
<th>user</th>
<th>state</th>
<th>submit/start at</th>
<th>queue</th>
<th>slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>2146096</td>
<td>0.50500</td>
<td>SimpleTest</td>
<td>user</td>
<td>r</td>
<td>10/22/2013 18:09:31</td>
<td><a href="mailto:all.q@node061.cm.cluster">all.q@node061.cm.cluster</a></td>
<td>1</td>
</tr>
</tbody>
</table>

Loading Environment Modules in SGE Scripts

SGE scripts must load environment modules for the applications they need. For example, here is a script using samtools and bwa. Note that the module load commands specify the specific version of each tool. This is good practice, because it ensures that the script will behave consistently, even if the HPC administrators install additional versions of samtools and bwa on the cluster later. This practice also documents which versions of the tools you used, making it easier for others (or you) to reproduce your work. (To see what versions of a given application are currently available on the cluster, use module avail appname.)

#!/bin/bash
#$ -S /bin/bash
module load samtools/0.1.19
module load bwa/0.7.5a
### your samtools and bwa commands go here ###
# Since the module commands have already set up the path,
# we do not specify the path to the tools:
samtools...
bwa...
Example Jobs

Example MATLAB Job

This section is intended for current MATLAB users who are familiar with the program.

Here is an example bash script to run an arbitrary MATLAB .m file using SGE.

1. Create a file called run_matlab.bash:

   ```bash
   [user@phoenix1 ~]$ emacs run_matlab.bash
   ```

2. Enter the contents shown below into the emacs editor.

   ```bash
   #!/bin/bash
   #$ -S /bin/bash
   #$ -cwd

   module load matlab/R2013a
   matlab -nojvm -nodisplay -nodesktop -singleCompThread -r "$1; quit;"
   ```

3. Create a MATLAB code file. Here is a simple example of matjob.m:

   ```matlab
   a = [1 2 3; 4 5 6]           % Matrix a
   sum(a)                       % sum all elements of each column in a
   mean(a)                      % Mean of each column
   max(a)                       % Max of each column
   max(max(a))                  % Obtaining the max of a matrix
   ```

4. To submit the job (assuming you have MATLAB code in a file called matjob.m), enter the command: qsub run_matlab.bash matjob Note that the .m suffix is not part of the command. Once you have submitted the job, you can check the status with qstat. Here is an example submission:

   ```bash
   [user@phoenix1 ~]$ qsub run_matlab.bash matjob
   Your job 2204233 ("run_matlab.bash") has been submitted
   [user@phoenix1 ~]$ qstat
   ```

6. Once the job is complete, review the contents of the job output file by entering: cat run_matlab.bash.o(enter here the number for your job), for example:

   ```bash
   [user@phoenix1 ~]$ cat run_matlab.bash.o2204233
   ```

   ```matlab
   < M A T L A B (R) >
   Copyright 1984-2013 The MathWorks, Inc.
   R2013a (8.1.0.604) 64-bit (glxv64)
   February 15, 2013
   
   To get started, type one of these: helpwin, helpdesk, or demo.
   For product information, visit www.mathworks.com.
   ```
Example R Job

To submit an R job to the cluster, first create a file containing R commands called, for example, `myscript.r`. You can use `emacs` to do so. For example, enter the following R commands into the `myscript.r` file:

```r
# The order() function makes an integer vector which is a correct ordering for the purpose of sorting.
D <- data.frame(x=c(1,2,3,1), y=c(7,19,2,2))
D
# Sort on x
indexes <- order(D$x)
D[indexes,]
# Print out sorted dataset, sorted in reverse by y
D[rev(order(D$y)),]
```

Save and exit Emacs to get back to the command line.

You will also need a bash file to run the job. You could create a bash script called `runr.bash` with the following contents:

```bash
#!/bin/bash
#$ -S /bin/bash
#$ -cwd

module load r/3.0.2
Rscript $*
```
Finally, to submit the job to SGE, type the following commands into the command line:

```
[user@phoenix1 ~]$ qsub runr.bash myscript.r
Your job 1090694 ("runr.bash") has been submitted
```

Once the job is complete, by default, SGE will place the output from your program in a file called `runr.bash.o jobnumber`.

```
[user@phoenix1 ~]$ cat runr.bash.o1090694
  x  y
1 1 7
2 2 19
3 3 2
4 1 2
  x  y
1 1 7
4 1 2
2 2 19
3 3 2
  x  y
2 2 19
1 1 7
4 1 2
3 3 2
```

**Example FreeSurfer Jobs**

FreeSurfer (https://surfer.nmr.mgh.harvard.edu/) is a software suite for the processing and analysis of MR images of the human brain. Since different versions of FreeSurfer have been shown to give different results on the same input dataset, it is essential to know if a particular version is needed and, if so, if it is available. To check on the versions available on Phoenix, use the `module avail` command:

```
[user@phoenix1 ~]$ module avail freesurfer
```

In the context of FreeSurfer's central processing pipeline (as implemented in the `recon-all` command), both original data and results are organized into what is called *FreeSurfer subjects*, or *subjects* for short, which consist of usually one, but possibly more, MR images from a single patient or volunteer at a given point in time as well as the data derived thereof in the course of the pipeline's many stages. Every FreeSurfer subject is represented by a directory and its files and subdirectories which strictly adhere to a structure defined by FreeSurfer. We will call this directory the *subject directory*, and it is its name that is used to identify the FreeSurfer subject.

**Side Note:** Hence, if a subject (that is, a patient or volunteer) has been scanned at multiple points in time, there are multiple *FreeSurfer subjects* for this single subject.

Several subject directories are usually contained in a *subjects directory*. Before working with FreeSurfer, the subjects directory needs to be configured by setting the `SUBJECTS_DIR` environment variable. Since this is a user-specific setting, it cannot be handled by the environment module, so that the following two commands are necessary in order to make FreeSurfer available for the duration of the current login session or script:
• module load freesurfer/version
  (Prepares the environment for FreeSurfer version version; replace version with the version identifier for the required version, cf. the output from the module avail command.)
• export SUBJECTS_DIR=path_to_subjects_directory
  (Sets an environment variable to point to your subjects directory; replace path_to_subjects_directory with the path of the directory that contains the FreeSurfer subject(s) you are about to work with, e.g. ~/freesurfer_studies/stroke.)

This, along with the command recon-all that carries out the actual processing, can be encapsulated in a batch job script (we use FreeSurfer 5.3.0 in this example):

```bash
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
module load freesurfer/5.3.0
export SUBJECTS_DIR=~/freesurfer_studies/test
recon-all -all -subj $1
```

The above script assumes that the FreeSurfer subjects are located in the subdirectory freesurfer_studies/test of your home directory. You can, of course, change that to match your needs. The script furthermore assumes that the name of the FreeSurfer subject to be processed is passed as an argument. Save the script in a file called, for example, run_freesurfer.bash.

To submit the job and process, for example, the FreeSurfer subject "bert", use the qsub command:

```
[user@phoenix1 ~]$ qsub run_freesurfer.bash bert
Your job 2090891 ("run_freesurfer.bash") has been submitted
```

Once the job has finished (check with qstat or wait for the email), check the subject directory ~/freesurfer_studies/test/bert for the results and log files as usual.

A more general approach would be implemented by the following script:

```bash
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
module load freesurfer/5.3.0
export SUBJECTS_DIR=~/freesurfer-test-subjects
recon-all "$@"
```

The special construct "$@" passes all the arguments given to the job script on to recon-all. This script hence behaves exactly like recon-all (except that it doesn't support spaces in file names). Assuming it has been saved as run_recon-all.sh, it could be submitted in the following way to create the FreeSurfer subject "ernie" from one of the sample images packaged with FreeSurfer:

```
[user@phoenix1 ~]$ module load freesurfer/5.3.0
[user@phoenix1 ~]$ qsub run_recon-all.sh -s ernie -i $FREESURFER_HOME/subjects/sample-001.mgz
```
Side Note: In the first line of the above terminal transcript, the FreeSurfer environment module is loaded so as to make the path to the FreeSurfer installation on Phoenix available in the environment variable `$FREESURFER_HOME`, which is used in the second line.

The same script could also be used to run the whole cross-sectional FreeSurfer pipeline on the subject "ernie", assuming that the latter already exists (see above):

```
[user@phoenix1 ~]$ qsub run_recon-all.sh -s ernie -all
```

As for the interactive FreeSurfer tools such as `tkmedit`, they cannot be encapsulated in scripts, but should be invoked from within a `qlogin` session, for example:

```
[user@phoenix1 ~]$ qlogin
Your job 2229626 ("QLOGIN") has been submitted
waiting for interactive job to be scheduled ...
Your interactive job 2229626 has been successfully scheduled.
Establishing /cm/shared/apps/sge/var/cm/qlogin_wrapper session to host node002.cm.cluster ...
[user@node002 ~]$ module load freesurfer/5.3.0
[user@node002 ~]$ export SUBJECTS_DIR=~/freesurfer-test-subjects
[user@node002 ~]$ tkmedit ernie norm.mgz
```

Note that for this to work, you have to have X forwarding enabled when logging in to the head node.

After closing the `tkmedit` windows, remember to log out from the `qlogin` session:

```
[user@node002 ~]$ logout
```

### Running Multiple Related Jobs

Multiple related jobs can be handled together using an *array job*. This launches many copies of the same job, but gives each one a unique task ID. The task ID can be tested to customize behavior. The task ID is given in the variable `$SGE_TASK_ID`. For example,

```
qsub -t 1-3 myscript.bash
```

launches three copies of `myscript.bash` in parallel. They are all part of the same job, but each one is a separate task. In one copy, `$SGE_TASK_ID` is 1. In another, `$SGE_TASK_ID` is 2. Finally, in the last copy, `$SGE_TASK_ID` is 3. For example, to sort several files `1.in`, `2.in`, ... and put the results in corresponding numbered files `1.out`, `2.out`, ...

one could write a job script like this:

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
# Simple SGE script to sort files named, for example, 1.in, 2.in, ... , 5.in
# with the sorted results in 1.out, 2.out, ... 5.out respectively.
# Call this with, for example, qsub -t 1:5 simple-array.bash

sort ${SGE_TASK_ID}.in > ${SGE_TASK_ID}.out
```

Suppose the above script is named `simple-array.bash`. Further suppose that one has three files to sort. Then the submission command would be

```
qsub -t 1-3 simple-array.bash
```
For more information on array jobs, see:
-  https://wiki.duke.edu/display/SCSC/SGE+Array+Jobs
-  http://docs.oracle.com/cd/E24901_01/doc.62/e21976/chapter2.htm#BGBHJCII

**Requesting Resources for Jobs**

**High-Memory Jobs**

If the total amount of memory consumed by all the jobs running on a given node exceeds the memory physically available on that node, performance can degrade dramatically. This problem can be avoided with the help of SGE, if we tell SGE how much memory to reserve for each job.

For example, suppose you have a job script called `myscript.bash` that you expect to need 2 gigabytes of memory. Please submit it as follows:

```
qsub -hard -l mem_free=2G -l h_vmem=2G -l mem_token=2G myscript.bash
```

What do these flags mean?
- **mem_free**: SGE will only schedule your job on a node that has at least this much memory available now.
- **h_vmem**: If your job grows past this upper bound, SGE will kill your job.
- **mem_token**: SGE records the fact that you intend to consume that much memory, and subtracts it from the pool available on that node.

Use these flags if your job is expected to use a significant amount of memory, for example, more than one gigabyte. A job submitted with these flags will wait in the queue until a node is available that has enough memory that is both currently free (`mem_free`) and not reserved by other jobs (`mem_token`). Once such a node becomes available, your job will be scheduled on that node.

Each regular node of the cluster has 128 GB of RAM. There are 64 of these nodes (plus an additional 5 GPU nodes each of which also has 128 GB of RAM). There is one high-memory node with 1 TB of RAM. `mem_free` looks at actual free memory, and it is unusual that a node will have all of its memory free. So if you want a job to have a shot at running on a regular node, it is probably best to keep your request to at most 120 GB. You can see what is available as follows:

```
qstat -f -F mem_free,mem_token
```

Typically, even a lightly-loaded node will have about 123 GB of memory free.

**Using temporary disk space `/tmp` for running jobs**

A lot of programs use the `/tmp` directory to store temporary files or data needed while the program is running. If not specified, the queuing system would not be aware of the amount of disk space a job will be taking up on `/tmp` and jobs may crash as a result of a lack of available disk space on the `/tmp` directory. To avoid this problem one should include the following parameters in the submission command:

```
qsub -hard -l tmp_free=40G -l tmp_token=40G -other_options myscript.bash
```

In the above example, 40 GB of `/tmp` disk space will be allocated to the submitted job and 40 GB will be subtracted from available `/tmp` space. The job will not be sent to a node that has less than 40 GB space available on the `/tmp` directory. These options guarantee that your job will not crash as the result of there not being not enough space in `/tmp` on a compute node.

One may find about the amount of available space for a queue as follows:
- To see the available parameters: `qconf -sc | grep tmp`
• For the tokens for a given queue: qconf -sq all.q | grep complex
• To see what is currently available: qstat -f -F tmp_token,tmp_free

**Multi-threaded Jobs**

Some programs are designed to spawn multiple threads or use multiple cores on the same node. To effectively use such programs with SGE on the cluster, you need to use the **threaded parallel environment**. For example (assuming the script to run is called temp.sh):

```
qsub -pe threaded 4 temp.sh
```

This will ask for four slots all on the same node. If there is no node with that many slots still available, your job will wait in the queue until one is available.

You can also specify a range of slots that is acceptable to you. For example,

```
qsub -pe threaded 6-12 temp.sh
```

means use at least six and up to 12 slots. That way, as soon as there is a node available with at least six slots, your script will be dispatched to that node, and use as many as are available there (up to 12).

In your script (temp.sh in the above example), you can use the variable `$NSLOTS` to find out how many slots you actually got. This can be used to set options to programs to limit the number of threads that they try to spawn. If the application that you are running has a flag that tells it how many threads, cores, or CPUs to use, chances are that it is using threads. In this case, you should use the threaded parallel environment, and your script should pass the value of `$NSLOTS` to the application’s flag that determines how many threads to spawn.

For example, **cufflinks** uses the `-p` command-line option to specify how many threads to use. So an SGE script for cufflinks might look like this:

```bash
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
module load cufflinks/2.0.2
cufflinks -p $NSLOTS ... # the rest of your cufflinks arguments go here...
```

If this script is called cuff.bash, and if anywhere from 2 to 32 threads is acceptable, the script would be submitted with

```
qsub -pe threaded 2-32 cuff.bash
```

**Requesting Resources for Multi-threaded Jobs**

If a job is allotted more than one slot, the distinction between the job and its slots becomes important when requesting resources (like memory or temporary disk space) for that job. For example, **mem_token** as introduced above is counted per slot (in SGE parlance, this is called a **per-slot consumable complex**), but others are counted per job. The following table gives an overview:
The amount of a per-slot resource requested with `qsub` options or by job-script directives is multiplied by the actual number of slots that the job gets assigned to determine the value for the job as a whole. This requires a user to think backwards when, say, the amount of memory needed by a threaded program as a whole is known: Divide that amount by the number of slots requested to find the value for the per-slot `mem_token`.

**Example:** Assume that a job script `memthread.sh` needs 40 GB of memory and runs with 4 threads. The correct `qsub` command would be:

```
qsub -pe threaded 4 -hard -l mem_free=40G -l h_vmem=40G -l mem_token=10G memthread.sh
```

This is because `mem_token` counts per slot, but `mem_free` and `h_vmem` count per job.

## Deleting Jobs

Jobs that have been submitted with `qsub` and that are currently waiting or running can be deleted. For waiting jobs, this will prevent them from ever running, whereas running jobs will upon deletion be stopped immediately without waiting for the job script to complete. Deleted jobs are also removed from the job queue. Deleting jobs is useful whenever it becomes apparent that a job does not behave as expected or has been submitted inadvertently or with the wrong parameters or directives. Deleting jobs frees up the resources that would otherwise be consumed by an unwanted job.

Jobs are deleted using the `qdel` command in the following form:

```
qdel jobid
```

where `jobid` is the job ID of the job to be delete. The job ID can be retrieved using the `qstat` command.

The `qdel` command can also delete an entire range of jobs, or specific tasks from an array job. For details, see `man qdel`. 
Transferring Data to and from the Cluster

To copy a small number of files (programs, configuration files, small data files) to and from the cluster within the NYU LMC network (often referred to as "the MCIT network") we use Secure Copy (scp).

Copying Data from and to the Client Computer

**Command-Line scp**

On a Mac or Linux machine, the command-line scp client is typically installed with the operating system. You can access it through a terminal running on your local machine. For example, on a Mac, start the Terminal application. Then, type an scp command. scp works very much like the Unix cp command. The difference is that either the source or the destination can be on a remote machine. A remote source or destination has the format user@host:file(s).

For example, suppose you have a Mac, and a local file called myfile on your mac that you want to copy to your home directory on the cluster. On your Mac, start the Terminal application and type:

```
scp myfile userid@phoenix.med.nyu.edu:
```

(Note the colon at the end.)

That copies it to your home directory on the cluster and keeps the name myfile.

You could instead do:

```
scp myfile userid@phoenix.med.nyu.edu:coolfile
```

Which would do the same thing, except that the remote copy of the file would be called coolfile.

If you have a subdirectory under your home directory, like mydir/, you could specify:

```
scp myfile userid@phoenix.med.nyu.edu:mydir/
```

in which case the remote file will be mydir/myfile.

Or combining:

```
scp myfile userid@phoenix.med.nyu.edu:mydir/verycoolfile
```

In which case, the remote copy will be mydir/verycoolfile.

These commands work in reverse to pull files from the cluster to your Mac:

```
scp userid@phoenix.med.nyu.edu:mydir/verycoolfile somefile
```

makes a local copy called somefile.

As with any Unix command that takes files or directories as arguments, the file or directory for either the source or destination can be specified as either an absolute path (one that begins with a slash) or a relative path (one that does not). For local files or directories, relative paths are relative to the current working directory. For remote files or directories, relative paths are relative to your home directory on the remote host.
Graphical Front-Ends to `scp`

For Windows users there are several free, GUI-based secure-copy clients, such as WinSCP (http://winscp.net/eng/index.php), FileZilla (http://filezilla-project.org/) or Cyberduck (http://cyberduck.ch/). Such clients can help to simply drag and drop files between the cluster front-end node and the user’s desktop or laptop.

Mac OS X users can use a GUI-based client, such as FileZilla, Cyberduck or Fugu (http://rsug.itd.umich.edu/software/fugu/). (As described above, Mac OS X users can also use the built-in command-line `scp` utility.)

Example of using Fugu on Mac OS X

It is convenient to create a favorite in Fugu so you won't have to enter you connection details every time you start up Fugu. To create a favorite, fill in the "Connect to" and "Username" fields and click "Add to Favorites":

You can then connect to the cluster by just selecting this favorite from the drop-down list. This also holds for subsequent launches of Fugu:
While Fugu is logging into the cluster, Mac OS X may ask you for your private key's passphrase (although Mac OS X will call it the "password for the key"). If this happens, type your passphrase in the field labeled "password" and click "OK".

Fugu should then show a window with two panes, showing files on your local computer and the cluster, respectively. To copy a file from your local computer to a directory on the cluster, for example the local file `demo.txt` to the directory `project1` on the cluster, simply drag it with the mouse from the left pane onto the respective directory in the right pane:
Copying files from the cluster to your local computer works analogously by dragging objects from the right pane to the left pane.

**Using WinSCP to Copy Data from a Windows Client**

A popular secure-copy client for Windows is WinSCP. You can obtain the latest version of WinSCP from the WinSCP download page (http://winscp.net/eng/download.php). Follow the installation package link and save the installer to an easily accessible location, such as your Windows desktop. You can start the installation by double-clicking on its icon. The installation is reasonably straightforward. There are instructions (http://winscp.net/eng/docs/guide_install) with screen shots that can guide you through the installation process.

The typical installation of WinSCP creates a shortcut icon on your Desktop. Start WinSCP by double-clicking the icon.

You will need to configure WinSCP the first time you use it. In the WinSCP Login window,

1. enter `username@phoenix.med.nyu.edu` under “host name” and
2. click on “Advanced”.

The Advanced Site Settings window will open. In its left pane, click on "Authentication".
1. Under "Private key file", select the private key file in PuTTY PPK format that you saved from PuTTYgen when first generating a key pair; then
2. click on “OK”: 
The Advanced Site Settings window will close, and you will return to the WinSCP Login window. It is convenient to save the session settings just configured. To do so, click “Save”.

![Advanced Site Settings window](image)
In the “Save session as site” window, click on “OK”. You have now configured WinSCP to access your files on Phoenix. To connect to Phoenix, from the WinSCP Login window,
1. select the session you saved in the last step, and
2. click on “Login”.
When connecting with WinSCP for the first time, you will probably be asked to establish the remote server's identity. When the following Warning window appears,

1. check if the server's key fingerprint is one of the following:
   - 66bt6kuSe/Kp18G//cNpNLo0cYJx8Wbz5Sdjw16oZM4
   - BNVVdGXPJSbr54LLIa/NcEKoyGSLkNVdJ2ozbqAIwE

2. If yes, click on "Yes". If not, click on "Cancel" and contact the HPC administrators (hpc_admins@nyumc.org).
You may be asked for your passphrase. In the following situation,
1. type the passphrase for your private key and
2. click on "OK":

Once you are logged in, a new WinSCP window pops up with two main panes: on the left, the 'local' pane contains your local directories/folders and files; the 'remote' pane on the right contains your directories and files on the cluster. To copy a file from your Windows PC, simply select the file so that it is highlighted and drag it to the remote pane.
After the file is dragged to the remote (or local) directory you will see this:

Click "Copy".
Using FileZilla to Copy Data from a Windows Client

FileZilla is a tool for copying data to and from a remote server using secure network protocols. A Windows version can be downloaded from its website (http://filezilla-project.org/).

FileZilla relies on Pageant, an SSH key tool that comes with PuTTY. Pageant needs to be running before FileZilla can be used to connect to the cluster.

Try this: Download and install FileZilla. Start Pageant, which should be available from the Start menu if PuTTY has been correctly installed.

In the Notification Area (which is usually located in the lower left corner of the primary Windows desktop), locate the Pageant icon and open its context menu. This can be done via the right mouse button. From the context menu, select "Add Key".

Use the dialog that comes up to open your private key file. Then, enter your passphrase when asked to do so.

Next, start FileZilla. Close to the top of its main window, in the field labeled "Host", enter

sftp://phoenix.med.nyu.edu

and in the field labeled "Username", enter your Kerberos ID or, if you have been assigned a different user name when your account was created, that user name. Then select "Quickconnect".
FileZilla may respond saying that the server's host key is unknown. If this happens, compare the key fingerprint as reported by FileZilla with the one in the following figure:

If they match, click "OK". Otherwise, inform the HPC administrators (hpc_admins@nyumc.org). Upon successful connection, the FileZilla main window should look like this:
To copy a file from the cluster to your local computer, locate it in the right pane and drag it to the left. To copy a file to the cluster, locate it in the left pane and drag it to the right. Information on how to use FileZilla together with Pageant can also be found in the FileZilla Wiki (https://wiki.filezilla-project.org/Howto).

**Performance of File Transfers**

Locally we should be getting several megabytes per second. Several factors determine network connectivity, hard drive speed, etc. If you need to benchmark please contact the HPC administrators (hpc_admins@nyumc.org).

**Copying Data from and to Shared Lab Directories**

Typical shared directories dedicated to a lab or similar organizational unit reside on a storage system similar to, but distinct from, the HPCF’s storage cluster. Therefore, such shared directories cannot be directly accessed from Phoenix (neither from the head node nor from compute nodes), but a bridge can easily be set up for users interested in copying large amounts of data from or to a shared directory. Please consult with the HPC administrators (hpc_admins@nyumc.org) as needed.
Disseminating Data to the Public WWW

Data that are suitable for dissemination to a world-wide public can be made available on the WWW directly from Phoenix. Every Phoenix user has their own user web directory at

/ifs/homewww/username/

—any files and directories placed there are immediately accessible from both the MCIT network and the public Internet under the URL of (note the tilde ~):

http://www.hpc.med.nyu.edu/~username/

(Note that HTTPS is not currently supported.)

**Warning:** Because data on the public WWW are subject to indexing, caching, archiving, *etc.* any files placed in user web directories for however short a period of time have to be considered irretrievably published. **Never place PHI in a user web directory.**

To place files (and directories as needed) in one’s user web directory, the files (and directories) have to

- be copied or moved to the user web directory, and
- have their permissions adjusted

as outlined below. Files and directories can be added to and removed from user web directory at any time.

**Side Note:** Adjusting the permissions is necessary to allow the web server to serve the user web directory. Because user web directories are served via HTTP, the filesystem permissions that apply elsewhere on Phoenix will not work for public access to user web directories. It is therefore not possible to restrict access to data in a user web directory to a certain group of Phoenix users by assigning that group to files or directories. Any data in user web directories are always accessibly by anyone, world-wide.

**Try this:** Placing data in user web directories works only while logged in to a headnode of Phoenix.

1. Make sure that the files (and directories, if any) you intend to disseminate via your user web directory do not contain any PHI or any other kind of data unsuitable for publication.

2. Copy or move the data in question:
   - To copy a single file from, say, your home directory, use a command of the following form:
     ```
     cp --preserve=timestamps ~/filename /ifs/homewww/username/
     ```
   - To copy a whole directory tree named *dirname* with all its subdirectories (if any) and files from, say, your home directory, use:
     ```
     cp --preserve=timestamps -r ~/dirname /ifs/homewww/username/
     ```
   - To move a single file or whole directory (tree) from, say, your home directory, for example because you are only interested in it being publicly accessible and don’t need it to be available for processing by batch jobs, use:
     ```
     mv ~/filename /ifs/homewww/username/
     ```

3. Adjust the permissions to grant "others" (that is to say, everyone) read permission (and additionally traverse permission on directories, if any):
   - To adjust the permissions for a single file by the name of *filename* that has been copied or moved to the user web directory, use:
     ```
     chmod -v o+r /ifs/homewww/username/filename
     ```
   - To adjust the permissions of a whole directory (including its subdirectories and files, if any) by the name of *dirname*, use:
     ```
     chmod -Rv o+rX /ifs/homewww/username/dirname
     ```

You can also create subdirectories in your user web directory using `mkdir` and copy or move files (or other directories) there. However, you will still have to adjust the permissions for the subdirectories thus created, as outlined above.

Files (and directories) can be removed from user web directories with the usual `rm` (or `rm -r`) commands.
Warning: Again, do not place PHI in a user web directory. It is every user’s individual responsibility to ensure compliance with this policy. If in doubt whether a certain file or dataset is allowed in a user web directory, the HPC administrators (hpc_admins@nyumc.org) can help.

Moving Large Data Sets to and from Remote Sites

Moving bulk data (such as entire outputs of sequencer runs, images, etc.) to and from remote sites (including public clouds, such as Amazon Web Services) can be time-consuming and/or expensive. SFTP and similar tools are fine to use locally but they are inefficient when moving large data sets to remote sites. More efficient tools include bbcp, Globus Online and Aspera. Please consult with the administrators (hpc_admins@nyumc.org).