Alabama Supercomputer Authority

HPC USER MANUAL

Eleventh Edition

Alabama Supercomputer Authority
686 Discovery Drive
Huntsville, AL 35806

The Alabama Supercomputer Authority

HPC User Manual

11th Edition

Alabama Supercomputer Authority
686 Discovery Drive
Huntsville, AL 35806

Publication	Date	Description
1st Edition	February 1988	Original printing
Revision A	September 1988	Minor typographical and editorial corrections
2nd Edition	June 1990	Updates and modifications of 1st Edition
3rd Edition	June 1993	Complete rewrite
Revision A	October 1993	New procedures and locations
4th Edition	January 1994	Update for Cray C90 and editorial corrections
5th Edition	March 1997	Updates, modifications, and new format
6th Edition	July 1999	Updates and modifications of 5th Edition
7th Edition	January 2005	Updates for Cray XD1 and SGI Altix 350
8th Edition	October 2008	SGI Altix 450, DMC, and new format
9th Edition	December 2010	Updated to reflect hardware upgrades
10th Edition	June 2013	Added Ultraviolet, removed Altix
Edition 10.1	March 2014	Update for Ivy Bridge & Kepler chips
11th Edition	July 2016	Updated for SLURM

Preface

This manual is provided for the users of the Alabama Supercomputer Center (ASC) as the primary reference for use of the High Performance Computing (HPC) systems at the Alabama Supercomputer Center. The manual covers the supercomputer configuration, available software and hardware, access methods, and user support.

Suggestions for additions or corrections to this manual should be directed to **hpc@asc.edu** or to:

HPC User Manual
Alabama Supercomputer Center
686 Discovery Drive
Huntsville, AL 35806

This manual is supplemented by a set of policies, which cover various aspects of services provided by the Alabama Supercomputer Authority. Alabama Supercomputer Authority policies are available at
http://www.asc.edu/usermanual/policies/policymenu.shtml

Table of Contents

1. Introduction

This manual is intended for people who will use the supercomputers provided by the Alabama Supercomputer Authority (ASA). This manual gives an introduction to the computing hardware, applications, operating system, how to connect to the computers, and how to run jobs. More detailed information on each of those topics is available in other locations, and referenced as each is discussed. Other books are referenced in the bibliography.

High performance computing (**HPC**) is the currently trendy monicker for supercomputing. Thus the terms “high performance computing”, “HPC”, and “supercomputing” are used interchangeably in this manual. Likewise the term “supercomputer” and “computing cluster” or just “cluster” are all synonymous. A few companies use the word “cloud” as a synonym for “cluster”, but most use the word “cloud” as a synonym for “virtualized server”.

The Alabama Supercomputer Authority

The Alabama Supercomputer Authority (**ASA**) provides high performance computing resources to state academic users, state government agencies, industrial users, and federal government agencies. ASA is a public state nonprofit corporation that develops, maintains, and operates the Alabama Supercomputer Center (**ASC**) and the Alabama Research and Education Network (AREN). Technical services are provided through professional services and facilities management contractor CSRA (formerly CSC). See the website www.asc.edu for more information about ASA.

The Alabama Supercomputer Authority provides a host of services in addition to high performance computing. The Alabama Research and Education Network (AREN) is a statewide high-speed network installed and maintained by ASA. Information technology services provided by ASA include access to high performance computing resources, Internet access, website and email services, and training. ASA provides web development services for a number of websites, including the Alabama Virtual Library (AVL), Alabama Learning Exchange (ALEX), Educate Alabama, and Alabama Career Technical Education. A number of customers also host disaster recovery equipment at the Alabama Supercomputer Center.

ASA’s high performance computing resources include a SGI Ultraviolet 2000 supercomputer and a Dense Memory Cluster (DMC). Usage of these systems is free for academic usage by faculty and students at public educational institutions in Alabama. The majority of this manual is devoted to the description and use of these systems.

About this Manual

Some items of information in this manual deserve particular attention by the reader. These are denoted by the presence of one of the following icons in the left margin.

Tips are suggestions for ways to use the system more effectively. The user can usually get work done without reading the tips, but will find that the tips describe ways to make frequent tasks more convenient.

WARNING: Warnings indicate pitfalls that could cause significant problems for the user. All users should read the warnings and follow their advice.

As the name implies, examples show a specific usage of a tool. Text that is not denoted as an example is a description of how to use the tool. The example icon is used to indicate a significant size example, not just a single line of text.

Reminders indicate information that is presented in other locations, but is also particularly important to understand to fully appreciate the current discussion.

Figures are set aside from the text through the use of a box with rounded corners, a black border, and a pale green background. Tables are presented in a similarly shaped box with a pale blue background.

There are also sections of this manual that show text as it is displayed on the computer screen. This is denoted by the use of a Courier New font. Text in Courier New bold face indicates the command that the user actually types. The non-bold text indicates the text provided by the system, such as the command prompt, or results displayed by a command. Here is a short sample of commands and output to the screen.

```
asndcy@dmc:~> ls -l ls_test
-rw-r--r-- 1 asndcy analyst 742 2008-06-03 13:06 ls_test
asndcy@dmc:~> chmod +x ls_test
asndcy@dmc:~> ls -l ls_test
-rwxr-xr-x 1 asndcy analyst 742 2008-06-03 13:06 ls_test
```

In this example, the text "`asndcy@dmc:~>`" is the command prompt consisting of the user name, machine name, and directory (the tilde "~" means home directory).

Information that the user must fill in with the appropriate name is denoted by < > signs, as shown below, or in ALL CAPITALS if < > could be confusing.

```
ls -l <file_name>
```

Optional command line arguments are denoted by [ ] signs, like this

```
ls [-l] ls_test
```

The notation "`CTRL-D`" means to hold down the "**control**" key on the computer keyboard while pressing the **D** key.

This manual has been written with information for a variety of users. As such, some sections of the manual may not be of interest to you. The following suggests sections of the manual that may be of interest to you.

- If you are new to Linux and the Alabama Supercomputer Center, read Chapters 1-4 and 6-9.
- If you are new to high performance computing, but experienced with Linux and computer science read Chapters 1, 3-6, and 9.
- If you will be using the HPC systems to write software, read Chapters 2, 5, 11, and 13.
- If you are an experienced user of the ASC HPC systems, you might find interesting new information in chapters 4 and 9 and the paragraphs denoted with the green Tip! icon in the left margin.

Online Help

Nearly all HPC software packages come with electronic versions of the documentation, which is available once you login via ssh. These are kept on the system in the directory **/opt/asn/doc** and its subdirectories. These directories also contain README.md files with a description of how to configure your account to run the software, and how to submit jobs to the queue system.

The directory **/opt/asn/doc/index** has links to browse available software by discipline. The `ascdocs` command gives a menu driven interface to browse through these directories. `ascdocs` can display text files. For other files, ascdocs will output the full path, which is needed to copy or download the files.

Tip!

The best way to get started using a piece of software is to read the README.md file in the directory **/opt/asn/doc/PROGRAM_NAME** This can also be done with the **`ascdocs`** command. The **general_information** category contains information that might be useful to users in many fields.

Various online help facilities are available. Linux information can be obtained with the man command. For example;

```
man <command_name>
```

or

```
man -k <keyword>
```

The man command locates and prints the entry named command_name. The title is entered in lowercase. If local documentation is available, "man" will give you the option of seeing that as well. However, man pages for specific applications may be available only after loading the module for the program. The following example reproduces the description of man on the standard output:

```
man man
```

For example, to get information on the gcc command, type:

```
man gcc
```

The terms of the software license agreement for many of the software packages are online. To see the list of software packages that have license agreements online, or see the license agreement for a specific program, type.

```
show_license list
show_license <program>
```

The literature citations for many of the software packages are online. To see the list of software packages that have citations online, or see the citation for a specific program, type.

```
show_citation list
show_citation <program>
```

Technical Support for Users

Support for ASC HPC users combines central site (ASC) support with an applications analyst and optional training and collaboration services. This support allows the user to utilize the HPC systems productively as rapidly as possible. Ongoing support to overcome problem areas and in mapping high performance computing technology into the researcher's specific area of study may be available on a case by case basis.

An applications analyst based in Huntsville is available to provide support services to the HPC user community. A manned helpdesk is available 24 hours a day to answer user questions about the status of the ASA systems and AREN. In most cases, the helpdesk will take down the relevant information and pass HPC help requests on to the HPC technical staff.

The following means can be utilized to contact the technical support staff at the Alabama Supercomputer Center.

HPC help Email:	hpc@asc.edu
Network Help Desk Phone (in Huntsville):	(256) 971-7448
Network Help Desk Phone (outside Huntsville):	(800) 338-8320

Tip!

The HPC support staff can often give the fastest, best response if you send an email to **hpc@asc.edu** and include as much as you can of the following;

- The command you typed
- The error message
- Which cluster you were logged in on at the time
- The directory you are working in
- The job number from the queue

The phone numbers above are the most expedient way to report an after hours system outage.

The focal point for technical support is the applications analyst. The analyst provides the following services to both educational and industrial users across the state:

General Support: The analyst provides assistance in establishing user accounts, finding documentation and example inputs, program compilation and execution, and other user support as needed.

User Training: The analyst provides introductory lectures, classroom training, and one-on-one instruction on selected topics. This usually needs to be arranged in advance, since there is travel budget for one trip per year to campuses outside of Huntsville, usually spring semester.

Application Program Support: The analyst provides support for installation of programs, limited optimization of code, in some cases use of application packages, and resource management.

Outreach Support: The analyst assists in promotion of ASA resources to potential academic and industrial users, through formal technical presentations, demonstrations, and technical consultation.

Collaboration: A number of types of collaboration opportunities can be negotiated on a case-by-case basis. These include hosting services at ASA, joint ventures for acquisition and operation of systems, and specialized training.

2. Computing Basics

This chapter contains general information on a number of topics. It contains an introduction to computer science, and some topics of particular relevance to running large mathematical calculations on supercomputers, also called high performance computers. If you are new to using computing clusters with job queue systems, or aren't certain of the difference between data in memory and files on a disk, you should read through these sections. Advanced users may find some information of interest in the sections on data management, and processor compatibility.

Parts of a Computer

What do you see when you look at a computer... a keyboard, screen, and places to plug in various types of cables. However, whether a computer is slow or fast, can store large amounts of data, or do large calculations, is a function of unseen components like the processor, memory, video card, and hard drive. Here are some components of a computer that you should be familiar with.

The **processor** is a chip, sometimes several chips, on the computer motherboard. If you look inside a desktop computer, the processor probably has a fan on top of it. The processor is sometimes called the **central processing unit**, or **CPU**. The processor takes a list of commands, called a program, and executes each command in the appropriate order. Most computers today have multi-core processors. These are simply multiple CPUs on a single computer chip. Each **core** is separate processor. The SLURM queue system uses "CPU" to denote a processor core, not the entire chip.

Reminder

Note that this manual uses the terms "CPU" and "processor core" interchangeably.

Processors have an internal clock with a speed expressed in gigahertz which is how many billion instructions per second the processor can execute. The clock speed is a reasonable way to compare two processors from the same manufacturer and generation. However, clock speed can be misleading when comparing processors from different manufacturers or generations. These points will be discussed in more detail in the section on processor capability and compatibility later in this chapter.

To a computer everything is data, regardless of whether it is a program, music file, or results from a quantum mechanics calculation. These various types of data may be used in different ways, but they are all stored in the same way. The data in a computer resides on a disk drive, or in memory, and at times in both locations.

The **disk drive** is a device that stores data files long term. There are various types of disk drives such as the hard drive inside most computers, faster but smaller solid state disks, and drives that access removable disks like CDs and DVDs. The disk drive is non-volatile, which means that the files are still there when you turn off the computer. Most of the files on the hard drive are not in use at any one time. Hard drives usually have a very large storage capacity ranging from hundreds of gigabytes (billions of characters) to terabytes (trillions of characters). A **byte** is a computer term for enough memory or disk space to store a single character on a type written page.

When you start a program or open a file, a copy of the information is read from the disk into **memory**. The memory is a computer chip which also stores data. The computer processor can access the information in memory much, much faster than it can access the information on a disk drive. However, memory is more expensive than disk storage, smaller, and volatile, meaning that it goes away when the computer is turned off. Most computers have a few gigabytes of memory, although the supercomputers have several terabytes of memory.

Tip!

To use computers effectively you should understand the difference between memory and disk storage. Confusing the two in conversation can be an embarrassing mistake.

The following are things that refer to disk storage.

- The capacity of a hard drive, disk array, CD, DVD, or USB flash drive.
- The information shown by the Linux commands ls, df, du, scp, sftp, quota, or usage.
- Anything about your home directory disk quota, or tmp disk usage, or scratch disk usage.
- File sizes shown by the Macintosh Finder, or Windows Explorer (not to be confused with Internet Explorer), or other types of file browsers.

The following are things that refer to memory utilization.

- The information shown by the Linux commands top, ps, squeue, or sbatch.
- The amount of memory you request when submitting jobs to the queue system.
- The “max memory used” value in the jobinfo command output.
- The Macintosh Activity Monitor, or Windows Task Manager
- Segmentation errors are a problem with the way the software is accessing memory.

A USB flash drive has non-volatile flash memory, so it works like a disk drive even though it is made from computer chips. Many smart phones have flash memory only, although internally there is a difference between what is stored in that memory for the apps that are running and those that aren't.

There is often no correlation between memory utilization and input file sizes. A program that searches through a massive genome file may use very little memory if it only reads a few hundred base pairs into memory at a time. A quantum chemistry calculation on a small input file may require a large amount of memory if you specified a large basis set. Likewise, the size of the program file says little about the memory needs which include loading the program, data, and dynamic linked functions into memory.

Computing systems are also described by their data transfer rates. Data transfer is measured in bits per second, often megabits per second (millions of bits), or gigabits per second (billions of bits). One byte (character) is made up of eight bits. You will most often see these ratings when discussing computer network capabilities, but there are also data transfer rates for the processor to access data in memory and to access disk drives.

Computing Clusters

A desktop or laptop computer is meant for tasks that are instantaneous, or as close to it as possible. In order to make these computers responsive, they are often sized with enough memory to have many programs open at once. Also the processor is running at a few percent of capacity and only spikes near it's full capability when you perform very mathematically intensive tasks, such as rotating your point of view in a 3D video game. A personal computer tends to seem slow when waiting to access data on the hard drive, or over a network, and seems very responsive when performing tasks with data entirely in memory.

Now consider the task of running a quantum chemistry calculation on a significant size molecule. To compute the molecule's vibrational motion might take from one to three days, depending upon the capabilities of the computer. Obviously, nearly instantaneous calculation is not an option on any computer in existence. Rather than having your computer freeze up for a couple days while this work is in progress, you would like to send that work off to some other computer to be done and bring the results back to you when it is completed. This type of batch processing model is how computing clusters are designed to operate. There are standard ways of interacting with these computer systems, although it is different from running desktop programs. However, once you learn to use the HPC systems at the Alabama Supercomputer

Center, you will find that other computing clusters like those at the national labs, NASA, the Department of Defense, and big corporations are used in a similar manner. **High performance computing** (**HPC**) is the currently trendy name for **supercomputers**. These are also called **computing clusters**, **beowulf clusters**, or just **clusters**. The next three paragraphs describe other types of large computer systems.

A **cluster** is computer system that is made of many individual computers, but utilized as one large computer. The HPC systems at the Alabama Supercomputer Center (ASC) are clusters.

Grid computing is the name for a configuration in which multiple HPC systems are interconnected. The current generation of grid computing systems are much more complicated to work with than the more mature technology of computing clusters like the ones at ASC. Grid systems will not be discussed further in this manual.

An **on demand computing center** is a cluster that can be segmented into smaller clusters, typically to sell computing time to large businesses. These centers often charge relatively high prices to customers, such as banks, who want access to a system with heavy security.

The most common definition of **cloud computing** is being able put in a credit card and immediately buy access to nodes, which might be set up as an HPC system, an email server, or may require you to install and configure the operating system yourself. Most cloud systems use virtualized servers. However, some companies use the term **cloud** to refer to anything that is done on a server connected to the internet instead of on your local computer, and a few use “cloud” in place of “cluster”.

The HPC systems at the Alabama Supercomputer Center are clusters. These are not grid computing systems, on demand systems, or cloud systems. From this point on, this manual will only discuss clusters like those at the Alabama Supercomputer Center.

Today’s HPC clusters are not one big computer. They are a collection of computers, usually with many computers mounted in a single rack, which are configured to be used as one large computing system. Computers that mount in racks are often called servers. Each server is a reasonably powerful computer with memory, processors, and hard drives, but not individual screens and keyboards (they are accessed remotely via a network). Each one of the computers is called a **node** in the cluster. In a commodity cluster like the DMC, one of the systems at ASC, each node many have eight or twenty processor cores and over a hundred gigabytes of memory. In a high end system, like the SGI Ultraviolet at ASC, a node may have hundreds of processor cores and terabytes of memory.

In order get the most computing power for the dollar, HPC systems use hardware that is at the high end of the commodity hardware, meaning components that can be purchased from more than one company. This typically includes high end processors, such as those in the more expensive desktops sold for video gaming enthusiasts, large amounts of memory, and larger or faster disk systems than a desktop computer. As such, a piece of software that runs on a single processor core will often take about the same amount of time on a cluster as on a high end desktop computer. However, on a cluster, software can be written to run faster by using more than one core as part of a single calculation. It is also possible to do more work on a cluster by using a single core for a calculation, but running hundreds of such calculations at the same time.

In order to use an HPC system, you need a way to tell it what software to run, and what data file contains the input data. You also need a way to get the results when it is completed. This should be something you can do from your desktop computer at work, or your laptop computer at home. This is done through a **login node**. The login node is just one of the servers in a cluster. The login node is where you can type commands, read documentation, submit work to run, see the results, and access files for the work you have in progress.

The login node can be accessed remotely over a network. This is done through a network protocol called **ssh**. If you have a Macintosh or Linux computer, ssh is already installed. Free ssh clients can be downloaded for use on Windows computers. ssh gives you a text interface to type commands. Thus you type text commands, and see the results as text. Most work on an HPC system is done in this text mode. Only a few programs have graphical interfaces available. Most that do have graphic interface options use the graphical program to prepare inputs and display results, but still require you to exit the program and use the text interface to run the calculation. Installing and using ssh is discussed later in this manual.

There are often many people working on the login node at the same time. Each person has their own files in their own directory, called a **home directory**. Linux has a whole system of permissions that allow you to work with your own files, but not see other people's files.

The commands that you type when you are logged in via ssh are **Linux** commands. Linux is the dominant operating system in the HPC and server markets, where as Windows and OS X (the Macintosh operating system) are more popular for personal computers. Thus is it necessary to learn a handful of frequently used Linux commands in order to use a HPC system. A short list of important Linux commands is presented later in this manual.

Most of the work done on a supercomputer is not run on the login node. You can do small tasks on the login node, such as reading documentation. However, anything that takes more than 10 minutes of CPU time (not 10 minutes reading things on the screen) and a small amount of memory will be automatically killed on the login node. To run calculations taking more than 10 minutes and more than 2 processor cores, you submit the calculation to a job queue. You tell the **queue system** a few critical pieces of information like what program to run, the name of the input file, how many processor cores to use, and how much memory it needs. The queue system then assigns the work to run on a **compute node**. Often all of the compute nodes are busy and the job will wait a little while before it can run. However, you can submit many jobs, log out, and come back later to check on them. You may refer to your work as a calculation, or simulation, or something else, but the queue system refers to it as a **job**. The queue system will guarantee that your job gets the requested resources. The queue system provides commands for submitting jobs, seeing what jobs you have running or waiting, and to kill a job if you so desire. After the job completes, the queue system creates an extra file that contains a log of any errors and information about how much memory your job used, whether it efficiently used multiple processors and more.

The jobs actually run on **compute nodes** in the cluster, not the login node. HPC systems often have one login node and hundreds of compute nodes. You can only work interactively through the login node. You utilize the compute nodes only by submitting work to the queue system. If you purchase your own HPC system, you will also have to work with additional servers that manage the queue system, passwords, software licenses, home directories, and backup copies of the files in your directories.

Another aspect of a cluster is the file system. Your home directory is visible every node on every cluster at the Alabama Supercomputer Center. This means that there are over two thousand processor cores constantly accessing the same, large home directory file system where everyone has their home directory. Clearly, a SATA drive like you put in a desktop computer could not handle that type of use. The home directories, software, and scratch directories are stored on a high performance GPFS file system. This is a storage cluster which has processors, memory and hard drives. The file you are using right now is usually pulled into the memory cache, even though it still appears to be a file on disk. Because of this, the first time you access a file that hasn't been used in days there is a lag as GPFS pulls the file from slower, cheaper hard drives to faster, more expensive media. Then you can work with the file repeatedly for a few hours and the response is fairly quick.

Using a HPC system is fairly easy once you have learned a handful of commands. Building or managing an HPC system is much more complicated as you must become an expert at using many Linux operating system commands and understanding many technical details of how the system works.

Once you have learned a few of the basics of Linux, the queue system, and transferring files to/from the HPC systems, you can focus on running calculations on the system. If the software is already installed, there will be a README.md file explaining how to configure your account for that software, submit it to the queue, etc. You can find the README.md file using the **ascdocs** command. Another good source of help is the HPC technical support staff, who can be contacted via email at **hpc@asc.edu**

Numeric Precision

Consider the task of computing the area of a circle on a computer. You are aware of the formula

$$A = \pi r^2$$

However, what is π? π is 3.14159... it goes on infinitely. Our computer doesn't have an infinite amount of memory to store an infinite number of digits of π, so how many digits does it use?

Computer programs most commonly work with two different representations for numbers, called single precision and double precision. Single precision means that the number is stored in 32 bits of memory, which translates to about eight significant digits in a base ten number. Double precision means that the number is stored in 64 bits of memory, which translates to about sixteen significant digits in a base ten number. Each bit is a 1 or 0, a digit in a base 2 number system, called binary.

It is often necessary to use a higher precision in the computer program than the desired answer. Consider a computer program doing the following.

$$1.2476543 - 1.2475021 = 0.0001522$$

In this example, we started out with eight significant digits, but ended up with four significant digits. This is called a loss of precision error. However, if you print out the resulting number, it may print as 0.00015227834 Where did the extra digits come from? Those extra digits are just garbage data that was hanging around in memory from some previous calculation. This gives the illusion that you have an answer accurate to eight significant digits, when you really have an answer accurate to four significant digits. The field of numerical analysis is devoted to understanding these types of issues and writing software that avoids these problems.

Tip!

We highly recommend that scientists or engineers who will be writing software for mathematical simulations take a numerical analysis class.

In any floating point mathematical calculation the computer must deal with the last digit that can be stored in memory. It can't just round up if the next number is greater than five, because it doesn't have memory to store the next number and thus doesn't know what that number is. Statistically, that last number should round up half the time and round down half the time, so randomly doing each half of the time is better than always rounding up or down. This process is known as "unit round", and likewise introduces unit round errors when it guesses wrong. After multiple iterations of a complex simulation, these unit round errors will accumulate so that the second to last digit can no longer be trusted to be correct either. The exact algorithm used to perform this unit round operation is different from one chip model to another, thus resulting in getting slightly different answers when the program is run on different models of CPU chip. This will occur even if the software is written correctly and the hardware is functioning correctly.

Some programming languages use the term "float" for single precision and "double" for double precision. A few languages use double precision all the time, even when declaring variables as "float". Some programming languages also support smaller precisions with variables declared as "short" or "word" or some other name.

When 64 bits of precision are not enough it is possible to run higher precision calculations. The processor doesn't have hardware to work with larger numbers, so the software must operate on part of the number, then if necessary cary a bit to work on the next larger part of the number. In this way, any precision can be obtained but the calculation will take much longer to perform. A popular way of doing this is to write a program to use the GNU Multiple Precision Arithmetic Library (GMP). This works, but is much slower than double precision mathematics. A few special purpose software packages, such as mathematica, have built in support for high precision computations, but calculations done using these packages run even slower yet because of it being an interpretive language (not compiled directly to machine language).

To make things a bit more complicated, x86 series processors do double precision mathematics in 80 bit extended double precision inside the processor, then round down to 64 bits to store the result in memory. This cuts down on certain types of loss of precision errors.

Data Management

In recent years, computer technology has advanced significantly. The downside of this has been long term data storage. Removable storage has evolved from disk packs (removable hard drive platters that go in a hard drive the size of a washing machine), to various magnetic tape options, to eight inch floppies, to 5.25 inch floppies, to 3.5 inch floppies, to USB flash drives, and writable CDs and DVDs. At the same time, the capacity of hard drives and solid state disks (SSDs) has increased immensely. However, the lifetime of all of these media for long term storage of data is still rather poor.

Newer flash memory may be warrantied for 5 years. Hard drives last 1-6 years. The CD-Rs that you typically find in stores start to degrade after a few years. There are more expensive archival quality CD-Rs rated for decades or even a century. Since these haven't been around 100 years, those ratings are based on theoretical projections. Likewise, manufacturers claim that DVDs may have a life span up to 30 or 100 years, but there is no industry standard for verifying the lifetime.

An alternative to storing data on removable media is to keep it in a running computer. To keep the data safe, there should be multiple redundancies and a budget to upgrade storage systems and replace failed drives as often as necessary. The home directories on the supercomputers are stored on a system with built in redundancy so that you will not lose files if a single hard drive fails. There are also nightly backups of the home directories, which are kept for only two weeks. However, the free academic accounts on the supercomputers provide only a modest amount of home directory disk space. It has always been the policy that it is the user's responsibility to transfer the data back to campus for permanent storage.

The /scratch disk area is visible to all of the nodes on both HPC systems. The scratch area is for work in progress. Any data left on /scratch is automatically erased one week after the calculation completes. There is no backup of /scratch. Thus any data files left on scratch must be transferred elsewhere within a few days if they need to be kept. As such, your work flow in /scratch should look like this;

1. Create a directory in /scratch. Many people use their account name as the directory name, perhaps with something added on to identify which job is running in that directory.
2. Copy data needed by a job to your directory in /scratch.
3. Run **one** job that uses the files in that directory.
4. Within a week of when that job completes, copy any needed results back to your home directory or to campus.
5. Delete your directory in /scratch.

6. Repeat this process, creating a new directory in /scratch for every job you run.

Clearly it is easier to work from your home directory than from /scratch. Your home directory disk quota can be increased up to a point (currently 1 TB) by requesting such from **hpc@asc.edu** For more than that you need to pay for additional home directory space. The /scratch area gives you access to ten times as much disk space as your home directory at no monetary cost, but at the expense of the additional steps in your work flow.

WARNING: Do not try to use /scratch as long term storage. Data files left in scratch may be automatically erased if they are more than a week old. Trying to cheat that system with the "touch" command does not work. If you need to save large amounts of data and do not have storage space on campus, contact the HPC staff for information on purchasing additional home directory space.

Each compute node of the DMC also has a /scratch-local disk area that is not visible to any other node. This is for running jobs to use a disk system that is faster than home or scratch. The /scratch-local area also uses a file system that can handle extremely large numbers of small files. However, you cannot see the compute node scratch-local area from the login node (which has it's own, different /scratch-local area). To use the /scratch-local area, your **job script** should include the following steps;

1. Create a directory in /scratch-local. Use a unique directory name since two of your jobs may end up running on the same compute node.
2. Copy input data to the directory.
3. cd to the new directory.
4. Do the calculation in that directory.
5. Copy output data back to your home directory. If your job creates a very large number of files, the "tar" command can be used to archive many small files into a single large file in your home directory.
6. cd to your home directory
7. Erase the directory from /scratch-local.

Shared file systems like /home and /scratch do not handle very large numbers of files in a single directory well. Thus use of /home and /scratch is limited to 500,000 files per user. Even at 500,000 files per user, your work will run very slowly due to bogging down the file system metadata server (part of the file system server). Work requiring tens of thousands of files should be done on /scratch-local which handles that type of usage much better.

Files stored in shared directories count against all disk quotas of the user who originally wrote a given file to that directory.

There is a separate file storage area under the directory /opt/asn/bio. This is an area where publicly available genomes can be staged. This saves on home directory disk space since everyone in the research group doesn't need to use their home directory space for public genome files. All users of the HPC systems can read the files in this area, but not write any data to this directory. To have a genome file staged here, contact the HPC technical staff at **hpc@asc.edu** This is a static storage area, so none of the files here are updated unless you contact the staff to request a new version.

Processor Capability and Compatibility

In the 1980s and 1990s, computer processors became significantly faster as the internal clock speeds increased from a few kilohertz to megahertz to gigahertz. If this trend had continued, today's processors would be running at hundreds of gigahertz. However, fundamental laws of physics have put a cap on clock speeds with high end processors being around four gigahertz for a number of years now.

Computer processors have continued to get higher performance, by other means. One of these is implementing dedicated circuits for complex instructions. For example, very early processors had circuitry to add numbers then performed multiplication as repeated additions, thus taking many clock ticks to multiply. Later processors have had dedicated circuitry to perform a multiply in a single clock tick. Other tricks include having secondary math coprocessor chips, and building processors with multiple floating point math units.

The x86 series processors that are in most laptops and clusters today have improved performance by supporting vector math instructions. These are instructions that perform a given mathematical operation on multiple numbers at once. There have been multiple generations of vector instructions with names like MMX, SSE, and AVX built into various models of computer processor.

These vector instructions can be utilized fairly easily by recompiling software, sometimes with an additional compile flag. However, if you compile software to use the AVX instructions in the most recent computer chips in the UV, it will not run on some of the slightly older processors in the DMC. It is possible to compile software to be compatible with the 386 chips from the early 1990s, which allows it to run on every x86 processor in use today, but it will run slowly everywhere. When you purchase commercial software with an automated installation program, it installs the correct files for your computer, thus hiding these issues from you. However, when you start compiling your own software on a supercomputer, processor compatibility and performance becomes part of your job. The following are some notes on compatibility issues.

There is a text file on all Linux computer systems named /proc/cpuinfo. That file contains information about the model of processor and what types of instructions it supports. Knowing this, you can compile several copies of your program with compile flags that give optimal performance on multiple processor models. You could then have the script that you submit to the queue look for specific entries in that file, then select which executable to use appropriately.

MPI (message passing interface) is a message-passing standard for distributed memory parallelization of software. The UV and DMC use different MPI library implementations. Thus MPI software must be compiled to run on one or the other, but cannot be compiled to run on both. The DMC is usually the better system for running MPI parallelized software.

OpenMP, a shared memory parallelization model, is available on both systems, and compatible between them.

The UV has Sandy Bridge series processors, which support AVX instructions (256 bit vectors) as well as older vector instructions like MMX and SSE (128 bit vectors). Some nodes in the DMC have slightly older processors that support MMX and SSE instructions, but not AVX instructions. As such, software compiled for AVX on the UV will not run on older DMC nodes that lack support for these instructions. However software compiled for SSE on the DMC will often run on the UV.

Software that can make significant use of AVX instructions may run twice as fast as software that uses SSE instructions. Later in this manual, we will give examples of how to write a script to select which program to run if you have compiled software to utilize both.

The command "**`file_info <file_name>`**" is an extension of the Linux "**`file`**" command. file_info will tell you if an executable file contains SSE or AVX instructions, in addition to giving information about what type of file it is.

Benchmarking

Benchmarking is the process of testing software to see how long it takes to run. This is typically done when writing software to be used on a computing cluster. To do so, the software developer will use a tool called a **profiler** to determine which parts of the program are taking the longest to execute. Then the software developer will use a number of **optimization** techniques to make the program get the exact same answer more quickly. Then a profiling or benchmarking calculation is rerun to see how much faster the software runs.

The supercomputers are configured to be useful for many types of work, but benchmarking is not one of them. As such, the supercomputers at the Alabama Supercomputer Center are a poor choice for benchmarking work. If you run the exact same calculation multiple times on the supercomputer the time it takes will vary, sometimes significantly, from one run to the next. This is for a number of reasons.

- The HPC systems are upgraded annually, often with different processor models. As such, your calculations may be running on different processor models on successive runs.
- The HPC systems are a shared resource. Your job(s) accessing files in the home directory is one of thousands of jobs accessing the home directory, so expect variations in access speed from one second to the next.
- There are other programs running on the same compute node. This load on the node creates contention for memory access and local disk access, thus changing the execution speed of your program based on what the other jobs are doing.

All of these are due to the supercomputers being configured to maximize the amount of work that can be done on them. However, this means that the best way to benchmark your program is to run it on a desktop computer with nothing else running on it, or a cluster architected for benchmarking.

GPUs

The HPC community has used a number of types of math coprocessor chips over the years. The DMC cluster has nVidia GPUs (Graphic Processing Units). Support for these chips has been building momentum in the market for the past few years. As this version of the user manual was being written, eight software packages on the system were supporting GPUs, and software tools for writing software to use GPUs are available.

GPUs are an adaptation of the technology in a high end video card, like those that are ideal for video gaming. GPU chips now have hundreds or thousands of processor cores on them, initially so that game physics could be offloaded to the video card. However, many processor cores on a chip can also be useful for running mathematical simulations more quickly. nVidia makes GPU chips that are designed for HPC by giving them a significant amount of double precision mathematics capability (video cards only need single precision). nVidia also developed the CUDA programming language, which allows programming software for GPU chips in a language that is similar to C. Today there are a few different programming languages available for GPU chips.

The reason that GPU chips are popular is because these chips give more processing power for the dollar spent than conventional processors. At the time this was written, the small testbed of 32 GPU chips in the DMC provided more single precision processing capacity than the other 1800 conventional processor cores in the DMC. The trade off that allows GPU chips to be so powerful is that they are SIMD (single instruction multiple data) processors. This means that every core on the GPU chips has to be running the same function, but each can be running that function on a different piece of input data. As such, a GPU is not a general purpose processor, and cannot be the main processor for a computer, only a secondary coprocessor. The disadvantage of GPUs is that software has to be rewritten to use these chips.

There are other types of math coprocessors, although not available on the ASC machines. ASC had a Cray XD1 from 2004 to 2009 which had FPGA chips. The Xeon Phi chips currently under development are a many core version of the x86 series processors. At the time this was written, the Knights Landing processor, the successor to Xeon Phi chips, appeared promising, but were not yet available for sale.

3. Account Administration

In order to use the supercomputers, users must get an account which consists of a user name, password, and disk space to store files. From that account, small jobs can be run on the login node, and larger jobs can be run on the compute nodes via the queue system. There is a wide selection of software available.

Requesting an ASC Account

There are three types of accounts on the supercomputers; academic accounts, class accounts, and commercial accounts. Each user must have a separate account on the supercomputer.

Academic Accounts

Academic accounts are free for academic usage by faculty and students at the public educational institutions in Alabama. Academic usage can be class work, thesis research, or work to be published in the peer reviewed literature. Work that will become the unpublished property of the funding organization is not eligible to be done in an academic account, but can be done in a commercial account.

To request an academic account, the user should submit an ASA HPC Annual Grant Request Form. This form is on the web at.

http://www.asc.edu/cgi-bin/account_request.cgi

NOTE: You must use your campus email address when applying for an academic account. The account request will be denied if you use a commercial email address such as gmail, hotmail, or yahoo.

Many people ask about the CPU Hours item on the account request form. This CPU hours request is not a hard limit. You can still keep running jobs when that many hours are used up. The supercomputer center staff uses the CPU hours for planning based on users anticipated needs. The applicant needs only fill in their best estimate. If other people in the same research group are doing this type of work, they may know how much they are using (ask them to login and type "usage" which shows year to date usage). Student taking a parallel programming class typically use 10 - 100 hours. Graduate students occasionally doing calculations typically use 1000 - 5000 hours. Graduate students doing all of their thesis work on supercomputers typically use 200,000 - 500,000 hours per year.

After completing the form, click "**Submit Grant Application**". This will create a second page summarizing the information that was entered. This second page must be printed, signed, and faxed to **256-971-7491**. Once the account is created, the user is notified by email and given additional information on using the account. Users are typically notified within three business days of when the faxed form is received.

Each year after receiving an academic account, the user will receive an email reminder to again fill out the account request form on the web. It is not necessary to fax anything in when filling out an annual renewal. If the renewal form is not filled out, the account is locked, and the student's research adviser is contacted to see if they need the files, then the account is erased.

Class Accounts

Class accounts are for the use of students enrolled in a course using the HPC systems for homework assignments. Like academic accounts, access to class accounts is free. Unlike academic accounts, class accounts are deleted at the end of the semester. In order to obtain class accounts, the instructor should contact the HPC staff at **hpc@asc.edu** In order to create the accounts, the staff will need to know the name of the course, the number of accounts required, and any software packages that will be used. The account passwords are provided to the instructor. Course instructors are advised to keep track of which account has been assigned to each student.

Commercial Accounts

Commercial accounts are available for the use of individuals in industry, government, private academic institutions, and academic institutions outside of Alabama. Commercial account time is purchased in advance and charged by the dedicated hour. Hours purchased must be used within 12 months of the date that the purchaser gets their account on the machine.

To obtain a formal written quote for CPU time, contact Donna Daniel, ASA Director of Client Services, at **ddaniel@asc.edu** or **334-242-0175**.

Some of the software packages at the center can be used at no additional cost, while others require an additional license fee for commercial usage. For information on commercial software pricing, include the list of desired software packages in the request for quote that you send to Donna Daniel.

Feel free to contact technical staff at hpc@asc.edu if you have any technical questions. Contact Donna Daniel for financial questions.

Acknowledgement

We ask that any publications from work done with academic or class accounts contain an acknowledgement. Here is a sample, which can be reformatted as necessary for a given publisher.

> **This work was made possible in part by a grant of high performance computing resources and technical support from the Alabama Supercomputer Authority.**

Disk Quotas

Each account has a quota that limits how much data can be stored. There is a soft quota and a hard limit. When the soft quota is exceeded, an error will be displayed when the user logs in on the system, and queue scripts provided by the staff will refuse to submit new jobs to the queue system. When the hard limit is reached, no additional data can be written to the account, which can result in having the running jobs halt execution.

When an account is created, a small quota is put in place (20 GB). Users can request up to a 1 TB quota (1000 GB) at no additional charge. Users can purchase additional disk space. Requests for a larger quota can be emailed to **hpc@asc.edu** . Requests for quote to purchase disk spaces larger than 1 TB can be sent to Donna Daniel, ASA Director of Client Services, at **ddaniel@asc.edu** or **334-242-0175**.

Reminder

> It has always been the policy that the Alabama Supercomputer Center systems are not intended to be used for permanent archival or storage of data. The home directory on the supercomputers is intended to be used for work in progress. Completed work should be transferred back to campus for permanent storage on the appropriate system there. The formal policies of the Alabama Supercomputer Authority are on the web at **http://www.asc.edu**

The "**`quota`**" command presents disk utilization information to the user. By default, "quota -q" is run during the login sequence for all user accounts. The following are the most frequently used options.

```
quota [-q]
```

-q	show information only if the user is over their quota

The "**`filecount`**" command lists how many files are in each subdirectory tree under the current directory. This is useful for determining where you have many files stored if you are over the account limit of 500,000 files. The stored data that allows **`filecount`** to run quickly is updated nightly.

The usage command gives a larger listing of information about the users computer use. It is invoked by simply typing

```
usage
```

This shows information for a user including disk quota, CPU hours used, login status, queued jobs, unix group membership, and system access.

The amount of disk space taken up by individual files can be displayed with the command

```
ls -l
```

The amount of disk space entire directories take up can be displayed with the command.

```
du -sk <directory_name>
```

4. Computer Security

The high performance computing (HPC) systems are set up to allow you to focus on your area of expertise. The intent is that you will need to put a minimal amount of effort (but still some) into learning the nuts and bolts of the queue system and Linux operating system. However, there are some rules and security practices to be followed. This is to protect you and the other people who use the HPC systems.

The following is a discussion of good security practices, both for using the HPC systems and for using any computer. If you have any questions or concerns, contact the technical staff at the Alabama Supercomputer Center at hpc@asc.edu

Accounts

Every user should have their own account. If there is a reason you need to access data or software in another person's account, there are mechanisms for making that possible while still using your own account. Some preferable alternatives are:

- You can email **hpc@asc.edu** to request the creation of a shared directory for sharing files with research group members, or class members.
- If you want to allow others to access your account files from their account, set read access only. Setting global write access also allows anyone to delete all of your files, which can happen by accident.

Each year, you will get an email reminder to renew your account by filling out a web form. You need the account name (but not the password) to be filled in on that form. If you don't fill it out, you will get reminders again the next two months. If the web form is not filled out, the account will be deactivated.

When an account renewal is not done, we contact the research adviser listed on the account request form to see if they want the student's research files before deleting the account.

Passwords

Do not share your password with anyone.

It is not necessary to give a password to the Alabama Supercomputer Center technical staff when you contact them, or they contact you. Note that simply hitting "reply" to the email that sent you your password sends the old message with the password back to the person that sent it, which should not be done.

Yes, you might have to keep your password written down so you don't forget it, but don't leave it on a post-it note on the monitor where anyone walking into the room can read it. A list of passwords on paper can be well hidden, or there are good free encrypted password database programs such as KeePass.

WARNING: Having your username written on the same piece of paper with the password means that someone who finds it has everything they need to access your account.

It is easy to find yourself with dozens or hundreds of passwords. Yes, most people use some passwords in more than one place. However, it is highly recommended that any password attached to money such as a bank website, credit card site, or retirement account is a completely unique password that you don't use anywhere else. It is also best if the password you use for work related accounts is not also used for non-work accounts.

We recommend against allowing a web browser to store passwords as this is frequently the target of hacker attacks.

WARNING: In July, 2016 a federal appeals court ruled that sharing passwords can be a violation of the Computer Fraud and Abuse Act.

There are illegal hackers who use an automated programs to try using millions of passwords to break into accounts. Thus anything that those programs would try as likely possibilities are a poor choice for a password. Here are some examples of BAD PASSWORDS.

• no password	(don't leave any account or computer without a password)
• asndcy	(password same as the account name is easily guessed)
• Rhino	(a word in the dictionary)
• Fido	(hackers try words from your social media posts)
• abc123	(popular sayings, song lyrics, book titles, people)

When setting a new password on the HPC systems, the password program will tell if the word you chose is insecure, and what is insecure about it. Typically a good password is a seemingly random mixture of letters, numbers, and symbols.

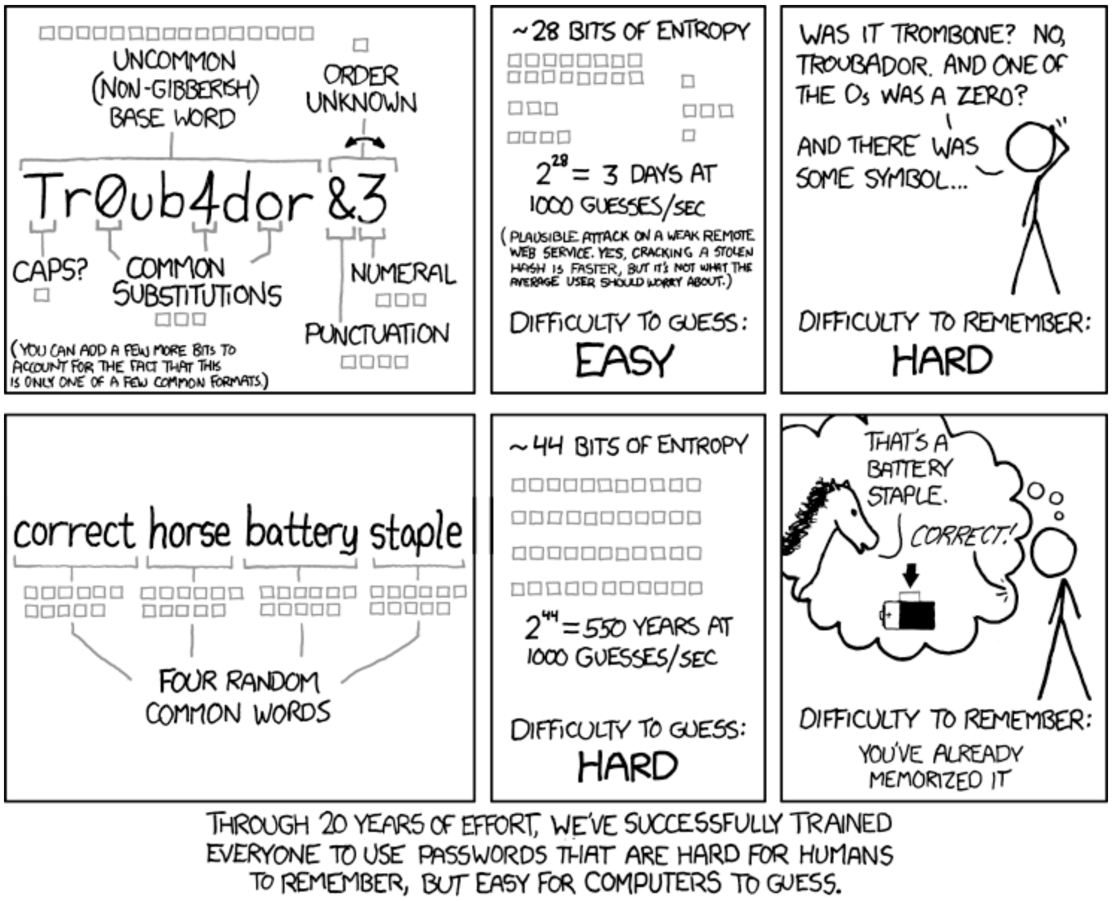

UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Image courtesy of XKCD

Data Security

What if you accidentally delete a file that you need?

- If the file is on /scratch or /scratch-local, it is gone and cannot be recovered. You will have to recreate the file.
- If the file has been in your home directory for 24 hours or longer, it can be restored if you contact the HPC staff SOON. There are nightly backups of the home file system, but those backups are only kept a couple weeks.
- If you created the file the same day you deleted it, again the file is gone and will have to be recreated.

There are times when you want to backup a file every few minutes or hours, such as when writing software. The nightly backups are not very useful for this, but here are some other options to consider.

- Some text editors can be configured to keep one previous version of the file with a slightly different file name.
- You can frequently copy the file to another directory.
- If you have a Subversion repository on campus, you can use the subversion commands to put new versions into the repository.
- You can use the Mercurial version control software to make a repository of old file versions in your home directory.
- You can utilize a web service for storing source code, such as bitbucket.org, sourceforge.net, or github.com

Note that the HPC systems are not a permanent archival or storage system. Once you are no longer using files, they should be deleted or transferred back to campus for permanent storage in the appropriate manner on campus.

In Linux systems, every directory and file has permissions associated with it. There are permissions for what you can do to the file, what your group members can do, and what everyone else can do. By default, no one can see the data in your home directory, but it is possible for you to change that. The Alabama Supercomputer Center uses the group permission for everyone in your department. Do not set global permissions (777 or rwxrwxrwx) unless you are sure you want everyone on earth to be able to read, run, and delete that file.

Monitoring

The account request that you signed includes a clause stating that you consent to monitoring of your account for the purpose of enforcing the acceptable use policies only. At present the technical staff goes into your account only when you ask for help or something indicates a problem with your account or usage.

The HPC technical staff may proactively contact you if there is a problem with your account or a queued job. However, the person contacting you will NOT ask for your account password.

Acceptable Use

Free academic accounts are available for faculty, staff and students at the public schools in Alabama. Academic usage includes;

- Work for class assignments
- Work for a thesis project
- Work to be published in the peer reviewed literature

Work paid for by an organization that will receive the unpublished results can not be done with a free account... but the Alabama Supercomputer Authority can sell you time on the HPC systems to do such work.

Things that should NOT be done with your HPC account include;

- Transferring music, video, images or other files for personal use.
- Sharing files with your friends.
- Anything that is a violation of a software license agreement. The software license agreements are in the documentation directory for any software installed on the HPC systems, which is accessible via the **`ascdocs`** command or **`show_license`** command.
- Any other illegal activity, such as hacking into computer systems you are not authorized to use.
- Mining for bitcoins or other digital currency.
- Anything that is not part of your assigned school work.

The official acceptable use policy is at **http://www.asc.edu/html/accusepol.shtml**

Fraud

There are many types of computer fraud in today's world. Identity theft fraud is very common. This is when someone steals or tricks you into giving information like your social security number, credit card information, birth date, mother's maiden name, passwords, or details of your credit history. This information will eventually be used to steal money from you in some way. It might also allow someone else to use your accounts, so that it looks like you were the one committing their crimes. Identity theft can also be used to gain access to something of value, such as usage of HPC systems.

No one should ask for your password. The HPC system administrators do not need to ask for it. Likewise, your bank or credit card company will not call you up or email you to ask for your password. If someone asks for your password, this is almost certainly a case of fraud. Do NOT give out your password.

One hallmark of fraud is when someone contacts you to ask for something, such as “verifying their records”. If in doubt, tell them you will get back to them. Then contact the legitimate organization in person or using a web address, email or phone that you know to be correct (NOT the one provided by the person or email that originally contacted you). Some fraud contacts will seem to know a large amount of information about you such as your credit card number, and may need just one more thing such as the security code on the back of the credit card to steal your money.

Likewise, websites you visit cannot see what is on your computer, so a website should not offer to fix some problem it claims to see on your computer. This is just another type of fraud disguised as a repair utility.

Other common fraud attempts include the following;

- An email or text from a loved one saying they need money urgently.
- Someone you never met wanting you to deposit their money in your account.
- You just won a prize for something you didn't enter.
- Here is a receipt for something you didn't buy. That might mean someone has already gotten your credit cards. Or it might be that the "receipt" is not a document but a program that will install a virus on your computer.
- Blackmailware is a virus that says it will do something bad unless you pay them money.
- You develop a long distance relationship with someone you never met, who then asks you for money.
- Ransomware may lock up your computer or encrypt your files, then asks for money to get it back. Some of these still delete your files even if you do pay.
- A call from the "customer service department" that doesn't specify the organization.
- You have been pre-selected for something.

If you see something odd that seems wrong, or unnecessary, or unexpected, it is often best to verify that it is legitimate from some independent source before giving any information. You can also ask them to send you the information through the U.S. Postal Service. Many fraudsters will not use the regular mail because it is easier to track and the penalties for mail fraud are more severe.

Other good places to find out if something is legitimate.

- The Better Business Bureau **https://www.bbb.org/** lists business ratings and complaints
- Angie's List will have reviews of other people's experiences with a business **https://www.angieslist.com/**
- Snopes **http://www.snopes.com/** lists internet hoaxes, many seen via social media or email
- Use **https://www.google.com/** to search on something along with the word "reviews" or "complaints". Keep in mind there may be a few good reviews posted by a fraudster or business owner, and a few bad ones posted by their competition.

- The National Do Not Call Registry **https://www.donotcall.gov/register/reg.aspx** allows you to exempt your phone number from sales calls. If your phone number is on this list, legitimate businesses will not call you, so any sales call you get is probably a fraud attempt.

Malware

The policies of the Alabama Supercomputer Authority require users to take reasonable steps to secure any computer connected to the Alabama Research and Education Network, which includes almost every school campus in Alabama, and home computers used to connect to the HPC systems. There are a number of implications of this.

Computers, especially those running Windows, should have some type of virus protection software installed. Updated virus definitions and operating system security patches should be installed on a regular basis.

There are some websites you should not visit, and web links you should not click. Some of these will attempt to hack your computer in some way. Some of the searches that are most likely to take you to such websites include “free music download”, “work from home”, and the names of popular celebrities.

Email can be another source of viruses. Do NOT open attachments unless it is something you were expecting to receive from someone you know.

A big step in protecting your computer can be to review the configuration settings in the web browser you use most frequently. There are a wide variety of options for various browsers on various operating systems. Here are some suggestions, most of which allow you to make exceptions if there is just one place that you need that item.

- Turn off third party cookies and site data. Unfortunately there are some legitimate services that require this, such as free email account which use those cookies for the advertising that pays for the service.
- Do not store passwords or form values in the browser.
- Do not allow sites to handle protocols.
- Block pop-ups.
- Do not allow sites to track your physical location.
- Do not allow sites to show desktop notifications.
- Do not allow sites to disable the mouse cursor.
- Do not allow sites to access the microphone or camera.
- Do not allow plug-ins to access your computer.

Government Security Requirements

The HPC systems are NOT certified for use with data that falls under federal controls, such as HIPAA, the DoD classification system, or ITAR.

A federal government law called FISMA (Federal Information Security Modernization Act) gives the National Insitute for Standards (NIST) responsibility for setting computer security standards that apply to all federal agencies. NIST has released NIST SP 800-53 and NIST SP 800-171, both of which set security standards for computing systems that handle "engineering and research data". Some federal agencies may issue additional clarification to these standards as they apply to that agencies work.

Starting in 2017, academic researchers who receive funding as a “subcontractor” to a federal agency must do that work on 800-171 compliant systems.

For academic researchers who receive funding from federal agencies as a “grantee”, the compliance date had not been set as of this writing (July 2016). However, the discussion in the security community is that this is only a matter of "when" not "if" it will happen.

Regardless of which category a researcher may fall in, the computer system as a whole either is or is not compliant. The same rules must apply to all users.

At the time this manual was written (July 2016) the HPC systems were not compliant with NIST SP 800-171, although it was under investigation. Contact the HPC staff at **hpc@asc.edu** to find out the current status of 800-171 compliance.

5. Supercomputer Hardware

There are two high performance computing systems at the Alabama Supercomputer Center. One is a shared memory system, consisting of a large SGI Ultraviolet 2000 node and it's associated login node. The other is a locally architected fat node cluster, called the Dense Memory Cluster (DMC). This section of the manual describes the hardware configuration of these systems. The systems are often upgraded annually. Thus the most recent specifications on the number of CPUs and amount of memory, can be found on the web at;

http://www.asc.edu/supercomputing/hardware.shtml

The supercomputers are connected to the Alabama Research and Education Network (AREN), which provides high speed network lines to the academic institutions in Alabama, as well as connections to the Internet and Internet 2. Connections to the supercomputers pass through a firewall, which excludes traffic from outside of the United States. Only encrypted connections are allowed to the supercomputers. Thus the primary means for connecting are ssh, scp and sftp. Telnet and ftp connections are not allowed.

There are several other security mechanisms in place. Users can change their password with the "**`passwd`**" command, but it will only accept passwords that are not readily broken by common computer hacking tools. The default home directory permissions prevent users from seeing files owned by other users. Users may alter these permissions to allow others to see all or part of their files.

Each cluster has a login node for interactive work (**uv.asc.edu** and **dmc.asc.edu**). The same password is used for both login nodes. Changing the password results in changing it both places. The same home directory and files are also visible on both systems.

Users compiling their own software must keep in mind that software compiled on the UV will not always execute on the DMC and vice versa. This is because the two systems use different MPI libraries, and the UV supports the use of AVX instructions, which are not supported on some nodes in the DMC. Both clusters are capable of executing generic 32 bit x86 linux executables, provided they do not use MPI. However, the systems are configured to compile 64 bit executables by default, and run them efficiently. Thus users writing their own software are advised to compile that software on the supercomputers in order to take advantage of the performance of these 64 bit systems and avoid library compatibility problems.

Both clusters share the same SLURM queue system. This means that, for example, jobs submitted from the DMC login node may get run on the UV. Jobs submitted with the run scripts provided on the system will run anywhere the desired software is available. Users writing their own software can submit it to the queue with the "**`run_script <filename>`**" command, which will prompt the user to specify where the job should be allowed to run. The queue system is described in more detail later in this manual.

`run_script <file>`	Runs a job on a single processor, or multiple processors on the same node.
`run_script_mpi <file>`	Runs a job using processors across different nodes, but all on the same cluster.

Supercomputer systems can be broadly categorized as shared memory systems, distributed memory systems, and hybrid systems. On a shared memory system, parallel software will run on a single node. A node is a collection of CPUs that run under the same instance of the operating system and can access all of the memory on the node. On a distributed memory system, parallel jobs can use CPUs on different nodes and communicate via some manner of message passing network. With the emergence of dual core and quad core CPUs, the majority of computing clusters today are hybrid systems which have multiple CPU cores on each node, but can also allow parallel jobs to run across multiple nodes. The SGI UV is a single, big shared memory node. The DMC is a hybrid cluster.

WARNING: Software will only run in parallel (using multiple CPU cores) if the software has been specifically written to run in parallel. In that case the documentation will talk about parallel execution using a given mechanism such as MPI or OpenMP.

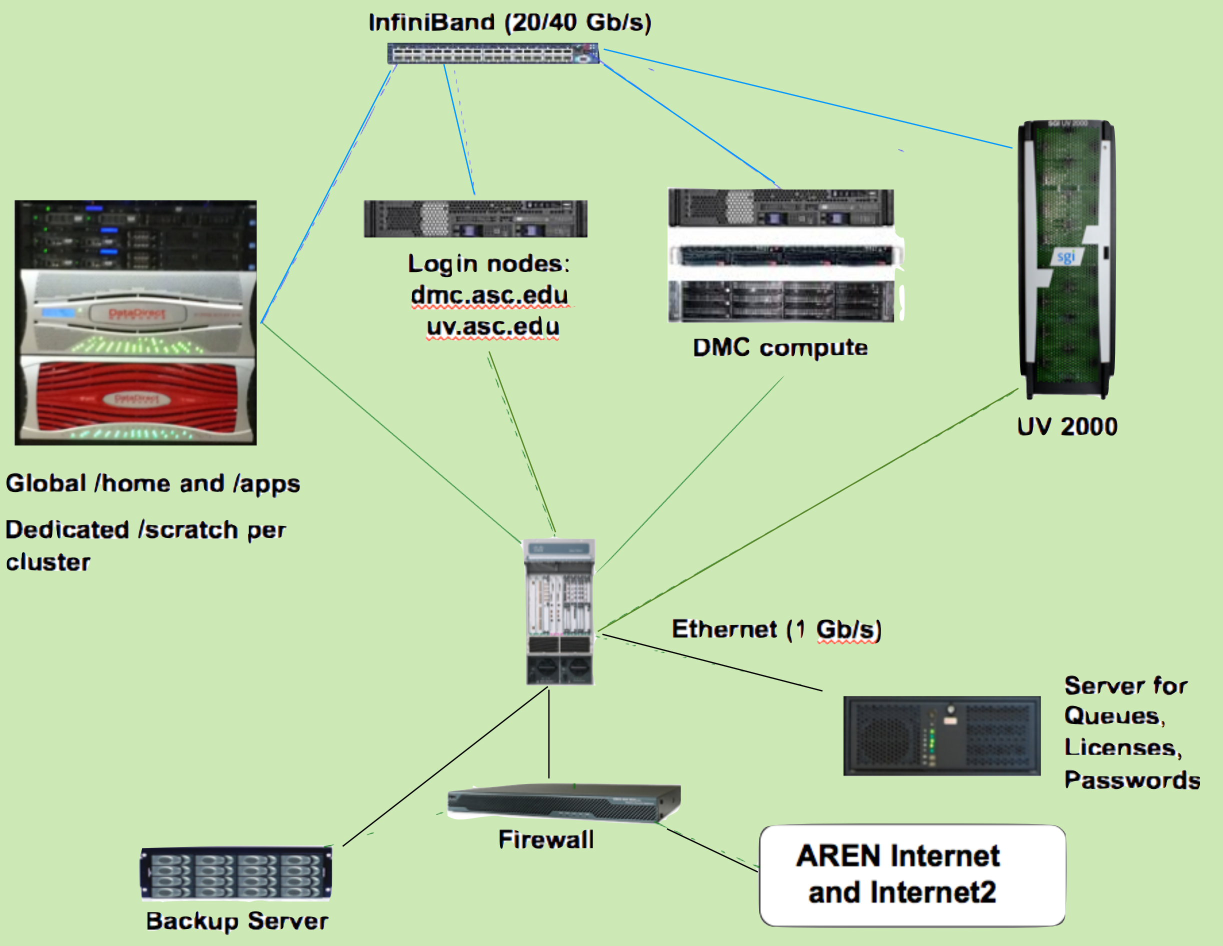
SGI Ultraviolet Shared Memory Supercomputer

The ASA SGI Ultraviolet 2000 system consists of a single, large compute node and a small login node. The compute node has 256 Sandy Bridge architecture Intel processors and 4 TB of memory. The compute node can be expanded if demand and funding should so dictate. The login node (uv.asc.edu) has 16 CPUs and 64 GB of memory. There could be hundreds of people logged in and using this login node simultaneously.

The SGI Ultraviolet is physically constructed of horizontal blades in order to fit many processors and memory DIMMs in a single rack. Communication between blades is handled by a NUMAlink switch on the back plane. These are 2.4 Ghz Intel Xeon processors, which gives a total maximum result rate of 5.19 TFLOPs for all of the cores combined. These Sandy Bridge architecture Xeon processors support the use of AVX (256 bit vector) instructions, which can potentially give a 2X performance improvement over the previous generation of Xeon processors which utilized SSE (128 bit vector) instructions. The website **http://www.asc.edu/supercomputing/hardware.shtml** should be consulted for the number of CPU cores and amount of memory currently installed.

Figure 5.1

The file system and network infrastructure.



Dense Memory Cluster

The Dense Memory Cluster (DMC) is a fat node cluster which was architected at the Alabama Supercomputer Center. It was put together from some commodity components, and some components that were already on hand at the Alabama Supercomputer Center. Hardware and software components for the DMC were obtained from Microway, Penguin, Cisco, Voltaire, SchedMD, Dell, Novell, Mellanox, Avocent, and DDN. This cluster was designed as a hybrid system with each compute node pushed as far towards a big memory, shared memory configuration as commodity hardware would allow. This was done to create a cluster that could run the majority of the jobs at the Alabama Supercomputer Center at an optimal price point. The DMC and UV are integrated into a common file system and queue system, as shown in Figure 5.1 .

The DMC nodes were purchased over several years. Each purchase was influenced by available technology, and the needs of the user community. The node configuration is listed in Table 5.1 . All of the nodes currently have processors that can execute four floating point operations per clock tick, through the use of 128 bit SSE vector instructions. The Xeon E5-2670v2 chips can execute eight floating point operations per clock tick, through the use of 256 bit AVX instructions. Most of the DMC nodes (except some with GPU chips) are physically configured as "twin-squared" systems with four complete nodes in each 2U of rack space, as shown in Figure 5.2 . All nodes have redundant power supplies.

Table 5.1 DMC Nodes

Nodes	Cores	Memory	Processors
dmc	8	24 GB	2.26 GHz Intel Xeon E5520 quad-core
dmc1-dmc4	20	128 GB	2.5 GHz Xeon E5-2670v2 + 4 Kepler GPUs
dmc5 - dmc40	20	128 GB	2.5 GHz Intel Xeon E5-2670v2 10-core
dmc61 - dmc124	8	24 GB	2.26 GHz Intel Xeon E5520 quad-core
dmc125 - dmc128	8	24 GB	2.26 GHz Xeon E5520 + 2 Tesla GPUs
dmc129 - dmc156	8	24 GB	2.26 GHz Intel Xeon E5520 quad-core
dmc157 - dmc172	16	128 GB	2.3 GHz AMD Opteron 6134 8-core
dmc173 - dmc196	16	128 GB	2.3 GHz AMD Opteron 6134 8-core
dmc197 - dmc200	16	32 GB	2.4 GHz Opteron 6136 + 2 Fermi GPUs

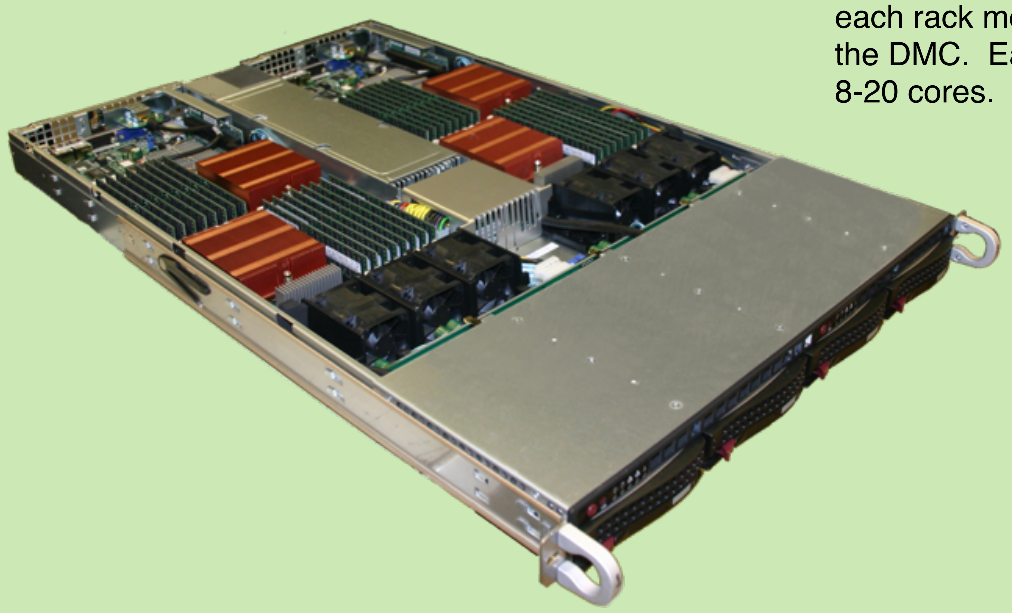


Figure 5.2

Multiple compute nodes fit in each rack mount server in the DMC. Each node has 8-20 cores.

At the time this manual was written, the DMC had a total of 2216 CPU cores and 12.6 Terabytes of memory. The website **http://www.asc.edu/supercomputing/hardware.shtml** should be consulted for the number of CPU cores and amount of memory currently installed.

The DMC is periodically updated and old nodes are decommissioned. Prior to January 1, 2013 nodes dmc1-dmc20 had 3.0 GHz AMD Opteron 8222 dual-core chips. Prior to Jan 1, 2014 nodes dmc21-dmc60 had 2.3 GHz AMD Opteron 2356 quad-core chips. The login nodes are also periodically updated.

The DMC can run shared memory parallelized applications up to the 20 CPU cores on a single node, or it can run distributed memory parallelized applications across multiple nodes. Message passing between nodes goes across an Infiniband network.

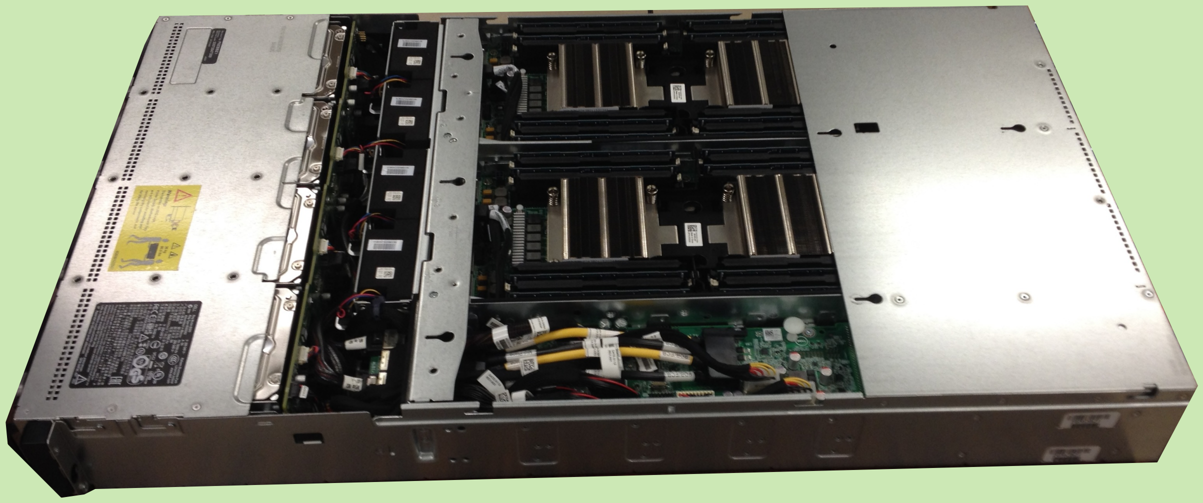
The DMC nodes connect to the same home directory and applications file systems that are used by the SGI Ultraviolet. Likewise, both see the same /scratch file system.

NVIDIA Tesla GPU Accelerators

Some of the DMC nodes also have specialized NVIDIA Tesla hardware accelerators attached. These accelerators leverage recent advances in commodity graphics processor technology to provide significantly higher performance than a traditional CPU for certain classes of applications. There are currently three generations of GPU chip installed. The oldest generation are Tesla 10-series processors with 240 cores on

Figure 5.3

Four Kepler GPUs in a densely packed server.



each chip, followed by Fermi processors with 448 cores, and Kepler chips with 2496 cores on each chip.

A single Tesla 10-series GPU supports a peak of 933 GFLOPs when performing single-precision floating point operations and 78 GFLOPs when performing strictly double-precision floating point operations. For comparison, the conventional 64-bit processors in the DMC nodes can provide up to 9.2 GFLOPs per core. To support this high rate of computation, each GPU also includes 4 GB of dedicated memory that provides 102 GB/s peak memory bandwidth compared to 10.6 GB/s per processor for the AMD Opteron processors in a DMC node.

Each Tesla S1070 contains four GPUs that are attached in pairs to DMC compute nodes via PCI Express cables. At the time this manual was written, two NVIDIA Tesla S1070s were installed. This amounts to a total of eight GPUs and 32 Gigabytes of dedicated GPU memory attached to four DMC compute nodes. The website **http://www.asc.edu/supercomputing/hardware.shtml** should be consulted for the number of GPUs currently installed.

There are also eight Fermi M2070 cards. Each of these cards has 448 cores and 6 GB of ECC memory. Each of these provides 1.03 TFLOP of single precision floating point capacity and 0.515 TFLOP of double precision capacity.

There are sixteen Kepler K20 cards. Each of these cards has 2496 cores and 5 GB of ECC memory. Each of these provides 3.5 TFLOP of single precision floating point capacity and 1.1 TFLOP of double precision capacity. There are four K20 GPUs installed in each GPU node, as shown in Figure 5.3 .

Software development is supported by a NVIDIA compiler suite known as CUDA that allows programmers to target NVIDIA GPUs using the standard C programming language with GPU-specific extensions for thread and memory management. NVIDIA also provides libraries that support a large number of functions from the standard BLAS and FFTW programming libraries, and allow users to leverage GPUs through minor code modification and linking against a different library. Further details on using GPUs may be found in the directory **/opt/asn/doc/gpu**

A more recent development is the ability to use OpenACC to program GPU chips. OpenACC is based on compiler directives before and after the loop to be parallelized, very similar to OpenMP. OpenACC directives can be used in C and Fortran with a compiler that supports OpenACC. An OpenACC extension to the Portland Group compilers is available on the HPC systems.

File Systems and Infrastructure Servers

There is a GPFS storage array for storage of data. GPFS is an IBM product, although the hardware it is running on is made by DataDirect Networks (DDN). This storage system hosts home directories, applications, genome storage, and /scratch space. The GPFS system connects to the nodes via Infiniband.

After the current file system was installed at ASC, IBM renamed GPFS as “IBM Spectrum Scale”. This documentation continues to use the GPFS name for consistency.

The infrastructure for the supercomputers also includes servers for passwords, group memberships, software licenses, operating system updates, security monitoring, and the queue system. Figure 5.1 shows a view of this infrastructure. The servers and network gear shown in the bottom third of this diagram are common to both the UV and the DMC.

6. Available Software

The Alabama Supercomputer Authority provides a selection of software to be used on the high performance computing systems. Commercial software packages are purchased based on the number of user requests, within budgetary constraints.

The software packages available on the HPC systems include both commercial and open source programs. The Alabama Supercomputer Center staff, installs these software packages, creates queue scripts to run them, and writes up README.md files with instructions on how to use the software. The extension .md means that the file is in markdown format. Markdown files are text file that can be read with any text file viewer such as **`more`** or **`nano`** or can have the additional formatting displayed using a markdown capable viewer such as the **`ascdocs`** command. The documentation and instructions on how to access each software package can be found on the system in the directory **/opt/asn/doc** or by typing **`ascdocs`**

Public domain software packages, such as those licensed under the GNU Public License (GPL), are available to all users of the supercomputers. The majority of the commercial software packages are purchased under licenses that allow academic usage only. Commercial customers may be required to pay an additional license fee to use the commercial software packages in order to cover the cost of obtaining the necessary commercial license.

Users may install software that is licensed for the use of their research group only, or any public domain software. These packages can be installed in the users home directory, or the users can request that the Alabama Supercomputer Center staff install the software for them by contacting the staff at **hpc@asc.edu** . The staff can install software in a centralized directory, then put a permission group on that software so that only authorized users can access it.

Software packages are added to or removed from the system from time to time. For a complete listing of the programs currently available login via ssh, then type the **`ascdocs`** command.

To request new software packages, or new versions of software contact the HPC staff at **hpc@asc.edu** Commercial software is upgraded as updates come out. Public domain software is updated by user request only.

7. Accessing the Supercomputers

Access to the supercomputers is available via encrypted connections, such as ssh, scp and sftp. Connections via telnet and ftp are not allowed. The ssh program allows the user to open a text console session on a remote computer. Thus ssh is essentially an encrypted version of telnet. The scp and sftp commands are for transferring files between computers. The sftp program works like an encrypted form of ftp.

When a user connects to a computer at ASC, the user must enter their user_id and the appropriate password. This is done from a shell or terminal prompt on a Linux or Unix system or Macintosh, from a Unix shell on a PC (using a Unix-in-windows tool, such as Cygwin or MKS), or using a graphical ssh program under windows, such as PuTTY.

WARNING: You may optionally add additional ssh keys to your account, but DO NOT remove the ssh keys provided with the account as this will cause your account to stop receiving output files from the queue system.

NOTE: You must be able to login with ssh before you can use scp or sftp. Any error messages generated on login with ssh will prevent scp and sftp from working correctly.

ssh connections from OS X, Linux, or Cygwin

Connections to the supercomputers via ssh can be made from a terminal window. On a Macintosh computer, the Terminal.app program is in Applications/Utilities. Some Linux distributions have a terminal icon on the menu bar, and others have a menu pick to open it, such as Applications->Accessories->Terminal in Ubuntu. A Windows computer with Cygwin installed will have a Cygwin icon on the desktop.

Once the terminal window is open, the commands syntax is usually the same on all systems. Typical variations on the ssh command line syntax are:

```
ssh <user_id>@hostname
ssh -l <user_id> hostname
```

where hostname is the name of the login node (uv.asc.edu or dmc.asc.edu) and <user_id> is replaced by your account name.

EXAMPLE

Examples:

```
local>ssh asndcy@uv.asc.edu
local>ssh -l asndcy dmc.asc.edu
```

These command should bring up another line asking you to type in your password. Enter your password, and press Return. Note that nothing is shown on the screen when you type the password, not even asterisks (which show an onlooker how many characters are in your password). If you get a message about password database being too restrictive, it means that you mis-typed the password. The password must be typed exactly as sent to you including the use of upper and lower case characters.

Once the correct password has been entered, a message will be displayed with any current announcements. You are now logged in on the supercomputer, and can use any of the Linux, module, or queue system commands described in this manual.

To logoff the supercomputers, type "**`exit`**" and press return.

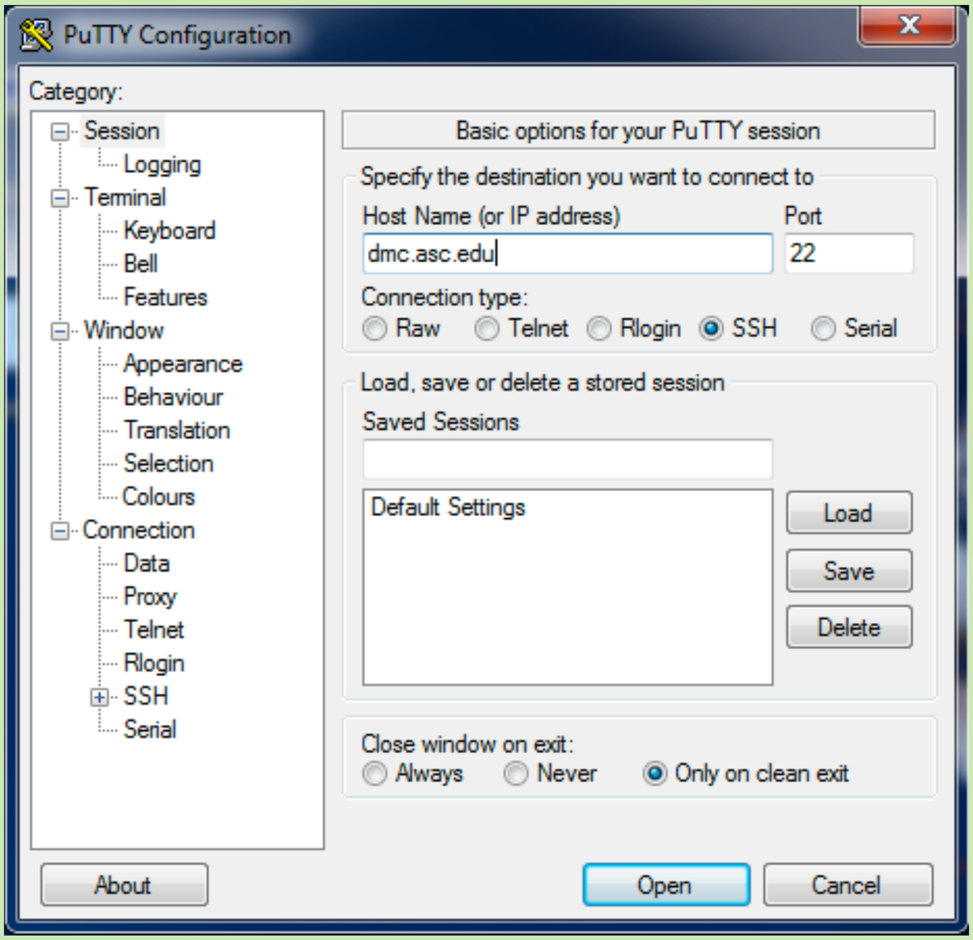
ssh connections from PuTTY

Windows does not come with a ssh program, but several free ssh programs are available. The easiest to use, free ssh program is PuTTY. The Cygwin program described later in this chapter is a more powerful system that installs a Linux environment on a Windows system. Cygwin is over kill for a simple ssh connection, but a powerful tool providing a Linux environment on a Windows computer and for running graphical programs on the supercomputers.

The free PuTTY program can be downloaded from **http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html** . This is a simple, free ssh program for Microsoft Windows. It can be used in conjunction with some X-window clients, such as X-Win32.

The file to download from the website is named putty.exe . Usage of psftp.exe and pscp.exe from the same website is discussed later in this chapter.

Unlike most software packages, putty.exe is not a package with an installation program. It is a single, executable file that needs to be run to use PuTTY. The putty.exe can be saved directly to the desktop. Another option is to save putty.exe to a directory of your choice, then create a desktop shortcut to it.



Specify the destination you want to connect to

Host Name (or IP address): dmc.asc.edu

Port: 22

Connection type: ( ) Raw ( ) Telnet ( ) Rlogin [✓ SSH] ( ) Serial

Default Settings

Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]

Figure 7.1

The PuTTY configuration window.

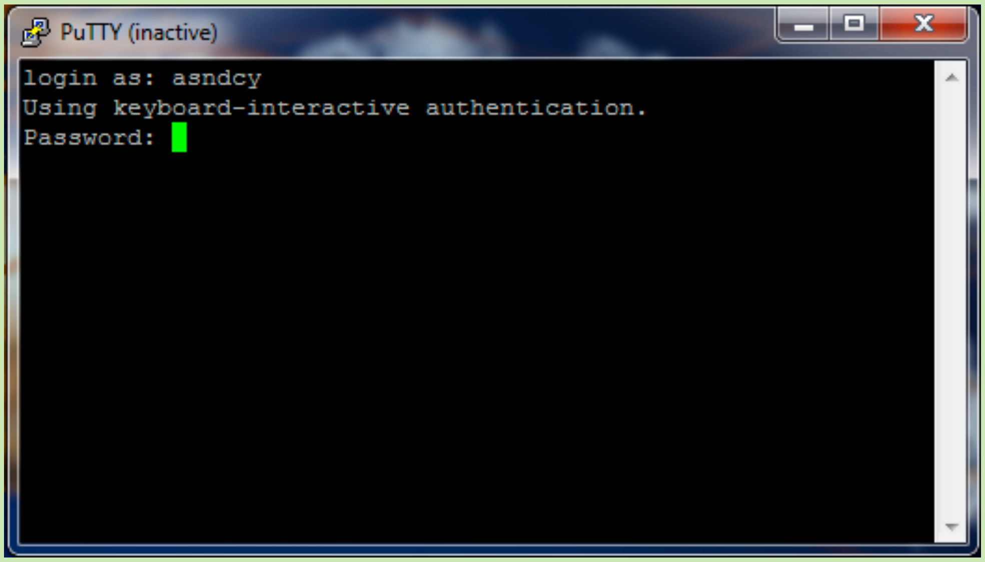
To connect to the supercomputers with PuTTY, first double click on the PuTTY icon on the Windows desktop. Windows may pop up a security warning message that requires you to click on “Allow”, or “Run”, or “Continue” in order to allow the PuTTY software to run. This will open the PuTTY Configuration window, shown in Figure 7.1 .

Enter the host name, either **dmc.asc.edu** or **uv.asc.edu** in the “Host Name (or IP Address)” box. The rest of the settings should be correct with the defaults. These are Port 22, SSH, and keyboard-interactive under Connection->SSH->Auth.

Reminder

When using commands involving a host name, such as ssh or scp, users should always specify the full name, e.g. **dmc.asc.edu**

Clicking on the “Open” button should cause the initial window to be replaced by the PuTTY terminal window shown in Figure 7.2



```
login as: asndcy
Using keyboard-interactive authentication.
Password:
```

Figure 7.2

The PuTTY terminal window.

The PuTTY terminal window will appear with the "login as:" prompt. Enter your supercomputer account name, and press Return.

Next, the "Password:" prompt will appear. Enter your password, and press Return. Note that nothing is shown on the screen when you type the password, not even asterisks (which show an onlooker how many characters are in your password). If you get a message about password database being too restrictive, it means that you mis-typed the password. The password must be typed exactly as sent to you including the use of upper and lower case characters.

Once the correct password has been entered, a message will be displayed with any current announcements. You are now logged in on the supercomputer, and can use any of the Linux, module, or queue system commands described in this manual.

To logoff the supercomputers, type "**`exit`**" and press return.

Transferring Files with sftp

Files can be transferred between systems on the network using the "sftp" command. The sftp program is very similar to ftp, except that sftp uses a secure, encrypted connection.

Files are usually transferred as ASCII files. Binary files can also be transferred. Use of sftp requires a valid userid and password on the remote system. The commands covered in this section are:

sftp	Establish a remote connection.
get	Move a file from the remote host to the local host.
put	Move a file from the local host to a remote host.
mkdir	Create a new directory on a remote host.
cd	Change directories on a remote host.
ls	List files in the current remote directory.
lcd	Change directories on the local host.
lpwd	Display the current directory path on the local host.
quit	Exit from ftp.

For additional information about the sftp command, enter "**`man sftp`**".

In Linux, OS X, or Cygwin; Open a terminal as described in the ssh directions earlier in this chapter. To establish a connection to a remote system, use the sftp command with the username and network address. After the connection is established, provide a valid password, as shown in the following example;

```
sftp asndcy@uv.asc.edu
Connecting to uv.asc.edu...
asndcy@uv.asc.edu's password:
sftp>
```

Using psftp.exe: Download psftp.exe from the PuTTY website the same way that putty.exe was downloaded. Double click on the psftp icon on the Windows desktop. A Windows security message may appear requiring you to click "Run", "Continue" or "Allow" to allow psftp.exe to run. Initiate the connection to the supercomputer with the command "**`open dmc.asc.edu`**" or "**`open uv.asc.edu`**". Type in your user name and press Return, then type in your password and press Return.

The `sftp>` prompt indicates that anything you type now should be sftp commands. This helps minimize confusion as some sftp command, such as cd and mkdir, are very similar to commands available from the regular login shell.

Using psftp.exe: The rest of the directions on using sftp work the same in psftp.exe with one exception. psftp does not always function correctly when directory names contain spaces. Thus you may not be able to transfer files in or out of the "My Documents" directory. The work around for this is to make a top level directory, such as C:\downloads and transfer all files in and out of that directory.

The get Command

The get command is used to transfer a file from the remote system to the local system. The syntax is:

```
get <remotefilename> <localfilename>
```

If the local file name is omitted, the remote file name will be used for both files. Below is how this should look.

```
sftp> get hello.f
Fetching /home/asndcy/hello/hello.f to hello.f
/home/asndcy/hello/hello.f 100% 65 0.1KB/s 00:00
sftp>
```

The put Command

The "put" command is used to send a file from the local host to the remote host. The syntax is:

```
put <localfilename> <remotefilename>
```

If the remote file name is omitted, the local file name will be used for both files. Below is an example of how this should look.

```
sftp> put test1.inp test_input.inp
Uploading test1.inp to /home/asndcy/test_input.inp
test1.inp    100%    5     0.0KB/s   00:00
sftp>
```

The mkdir Command

The “mkdir” command is used to create a new subdirectory on the remote host. The syntax is:

```
mkdir <new-sub-dir>
```

The following is an example of how this should look.

```
sftp> mkdir demosftp
sftp>
```

The cd Command

The “cd” command is used to change directories on the remote host. The syntax is:

```
cd <dir-name>
```

The following is an example of how this should look.

```
sftp> cd demosftp
sftp>
```

The ls Command

The “ls” command is used to get a listing of the files in the current remote directory. The syntax is the same as the Linux ls command syntax. For example;

```
sftp> ls
a.out            demosftp         hello.f          test.inp
sftp> ls -l
-rwxr-xr-x    1 asndcy   analyst    800106 May 19 11:58 a.out
drwxr-xr-x    2 asndcy   analyst         6 May 20 11:49 demosftp
-rw-r--r--    1 asndcy   analyst        65 May 18 13:49 hello.f
-rw-r--r--    1 asndcy   analyst         5 May 20 11:48 test.inp
sftp>
```

The lcd Command

The “lcd” command is used to change directories on the local host. The syntax is;

```
lcd <dir-name>
```

The lpwd Command

The “lpwd” command shows the full path to the current directory on the local host. No argument is required.

The quit Command

The quit command exits from the ftp session. The quit command does not require any arguments. After typing quit, the session should return to the command prompt, as shown in the following example.

```
sftp> quit
asndcy01@delldhp2 /scratch-local>
```

Transferring Files with scp

The following describes how the “scp” command works on Linux, OS X, and Cygwin. There is a pscp.exe program on the same website as PuTTY. pscp.exe works similarly, but does not work on all versions of Windows.

The commands scp and sftp can be used to copy files from your desktop computer to the supercomputers, or from the supercomputers to you desktop. Full documentation for these commands can be viewed on the supercomputers by typing “**man scp**” or “**man sftp**”.

The scp command works like the cp command, but requires that a username and computer network address be specified also. For example, if you are on your computer, in a Cygwin shell, in a directory containing myfile.txt and you want to copy it to the supercomputers, the following command would be used.

```
scp myfile.txt <login_id>@dmc.asc.edu:~
```

The <login_id> would be replaced by your user name. The tilde “~” designates that the file should be put in your home directory.

The scp command to copy the file back from the supercomputer to the directory you are currently in within a Cygwin shell would look like this

```
scp <login_id>@dmc.asc.edu:~/myfile.txt .
```

In this case the single period at the end indicates that the file is copied to the directory your are currently in.

Tip!

Savvy users who move large amounts of data around tend to use scp because it requires less typing than sftp, or rsync for its advanced features.

X-Windows

Some software packages at ASC have graphical interfaces, which require the use of an X-Windows server program. X-Windows is the graphical user environment used on UNIX and Linux computers. Unlike Microsoft Windows, X-Windows was designed from its inception to display graphical interfaces on a computer that is geographically removed from the one actually running the program. X-Windows software is usually included with Linux or Unix operating systems. There is an X-Windows server for Macintosh OS X systems called X11, which is included on the operating system installation DVD but not installed by default. For Microsoft Windows users, it will be necessary to install an X-Windows server.

If you wish to use a program that utilizes an X-Windows interface and run it from a PC, first contact your campus information technology office. Some campuses have X-Windows servers available at a reduced cost or no cost. There are several good commercial X-Windows clients, such as X-Win32 or Exceed. The free MobaXterm, Xming, and Cygwin packages also include X-Windows servers.

Cygwin, MobaXterm or Xming work with some Windows versions and not others. Check their websites for information about using the latest version on your version of Windows. MobaXterm is the easiest to install, although it has a couple quirks to work around. The Xming X-Windows client is easier to install than Cygwin.

Installing and running MobaXterm

There are both free (also called home) and professional versions of MobaXterm. The directions below are based on using the free version. MobaXterm can be used for ssh, sftp, and X-Windows, although it is overkill if you don't need the X-Windows functionality.

Installation

Download the MobaXterm zip file from the website
http://mobaxterm.mobatek.net/ We recommend using the free version.

Double click on the file name in Windows Explorer in order to unzip it.

In Windows Explorer, drag the program file to the desktop. It might have a name like MobaXterm_Personal_5.0

NOTE: The file you need is NOT the file with "Customizer" in the name. The long file names might be obscured as too long to be icon names.

Using SSH in MobaXterm

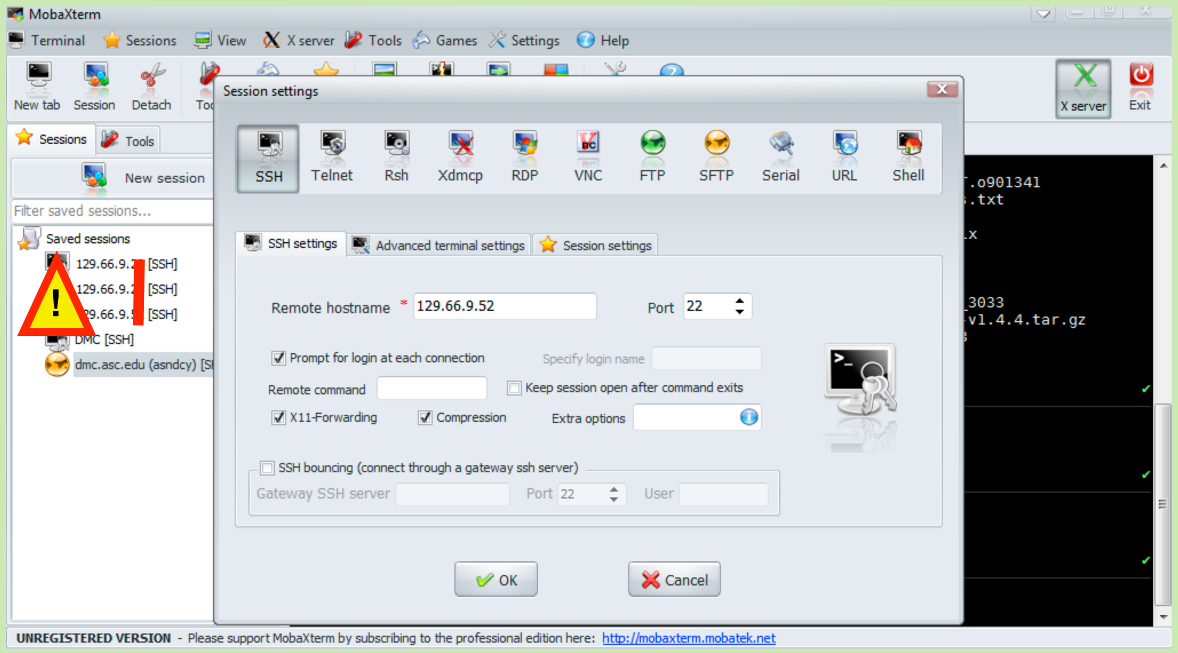
Double click on the "MobaXterm" icon on the desktop to start it.

Within MobaXterm, click on the "Session" icon. If it is the first time you have used MobaXterm, it might immediately give you a dialog box asking what type of session to open. Otherwise, click on "New Session". The dialog box for opening an ssh session is shown in Figure 7.3

In the Session dialog, click on "SSH"

NOTE: At the time this was written, MobaXterm may have issues talking to DNS on some computers, even if DNS names work from other applications on the same computer.

Figure 7.3 Starting an SSH session in MobaXterm.



Saved sessions: 129.66.9.1 [SSH], 129.66.9.1 [SSH], 129.66.9.5 [SSH], DMC [SSH], dmc.asc.edu (asndcy) [SSH]

Remote hostname * 129.66.9.52 Port 22

[✓] Prompt for login at each connection Specify login name
Remote command [ ] Keep session open after command exits
[✓] X11-Forwarding [✓] Compression Extra options

[ ] SSH bouncing (connect through a gateway ssh server)
Gateway SSH server Port 22 User

```
.0901341
.txt

x

3033
v1.4.4.tar.gz
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

In the "Remote hostname" box type dmc.asc.edu or uv.asc.edu and press the "Return" key. If you run into issues resolving DNS names on your computer, try substituting 129.66.9.52 for the DMC or 129.66.9.20 for the UV.

Type in your account name at the "Login:" prompt and your password at the "Password:" prompt.

NOTE: After your connect via SSH for the first time, your session will be saved in the "Saved Sessions" area on the left. You can open a new session by double clicking on these entries. You can right click on the session name to rename it.

Using X-Windows with MobaXterm

X-Window tunneling through ssh is turned on by default, as is the X-Window server and OpenGL support.

After logging in via ssh, type "**`xclock`**". This should display a small window with a dial clock. It may be displayed behind the MobaXterm window. Also, an icon for the XClock session should appear in the taskbar at the bottom of the screen.

If xclock displays properly, then X-Windows is working correctly.

Close the clock window.

You can now run X-Window programs through this ssh session.

Using SFTP in MobaXterm

MobaXterm can act as a sftp client to transfer files between your local computer and the supercomputers. However, the sftp graphical interface was broken at the time this was written, so sftp needs to be run from the command line, as shown in Figure 7.4

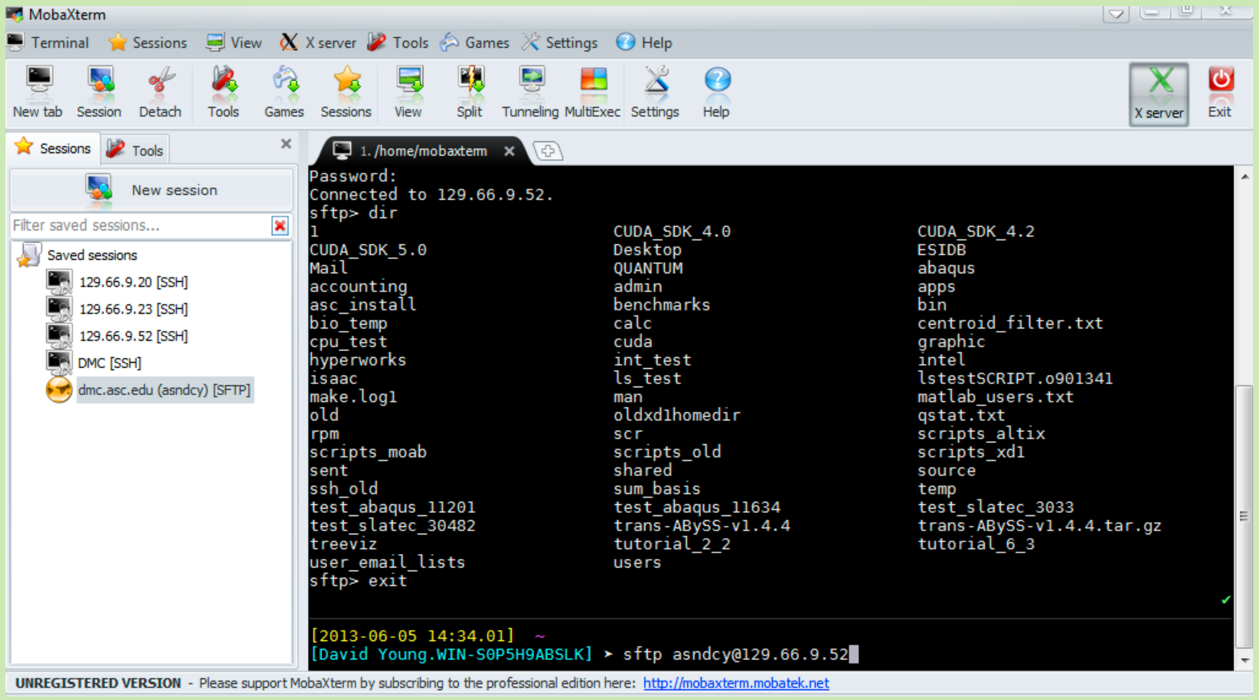
WARNING: SFTP cannot be run by starting from the "Session" icon. It must be started from the command line, with a command like this.

```
sftp myaccount@dmc.asc.edu
```

Type this in the left most tab (shell window), which is logged in on the local machine. Enter your password when prompted. If you run into issues resolving DNS names on your computer, try substituting 129.66.9.52 for dmc.asc.edu.

From here, follow the directions for using sftp that are found in this manual.

Figure 7.4 Starting an sftp session in MobaXterm.


MobaXterm
Terminal Sessions View X server Tools Games Settings Help
New tab Session Detach Tools Games Sessions View Split Tunneling MultiExec Settings Help X server Exit
Sessions Tools
New session
Filter saved sessions...
Saved sessions
129.66.9.20 [SSH]
129.66.9.23 [SSH]
129.66.9.52 [SSH]
DMC [SSH]
dmc.asc.edu (asndcy) [SFTP]
1. /home/mobaxterm
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

```
Password:
Connected to 129.66.9.52.
sftp> dir
1                             CUDA_SDK_4.0                  CUDA_SDK_4.2
CUDA_SDK_5.0                  Desktop                       ESIDB
Mail                          QUANTUM                       abaqus
accounting                    admin                         apps
asc_install                   benchmarks                    bin
bio_temp                      calc                          centroid_filter.txt
cpu_test                      cuda                          graphic
hyperworks                    int_test                      intel
isaac                         ls_test                       lstestSCRIPT.o901341
make.log1                     man                           matlab_users.txt
old                           oldxd1homedir                 qstat.txt
rpm                           scr                           scripts_altix
scripts_moab                  scripts_old                   scripts_xd1
sent                          shared                        source
ssh_old                       sum_basis                     temp
test_abaqus_11201             test_abaqus_11634             test_slatec_3033
test_slatec_30482             trans-ABySS-v1.4.4            trans-ABySS-v1.4.4.tar.gz
treeviz                       tutorial_2_2                  tutorial_6_3
user_email_lists              users
sftp> exit

[2013-06-05 14:34.01]  ~
[David Young.WIN-S0P5H9ABSLK] ➤ sftp asndcy@129.66.9.52
```

scp commands can also be run from the shell window.

Getting help in MobaXterm

The Help->Documentation menu pick in MobaXterm opens a web page with the MobaXterm manual in it.

At the command prompt, you can use a command like this

```
man command_name
```

Installing and running Xming

The Xming software can be downloaded from

http://www.straightrunning.com/XmingNotes/

Here are extra notes about doing the Xming installation.

- On the web page, scroll down to the "Releases" section and to the second table under the column heading saying "Public Domain Releases". Click on the link that just says "Xming". This should take you to a Sourceforge web page and open a download window. If the download window doesn't open, it may be necessary to set an exception to the pop up blocker in your web browser.

- The downloaded file can be run (i.e. from Windows Explorer) to install the software. During the installation, several security windows may require you to indicate that the installation should be allowed and unblocked.

- The default Xming installation settings should work.

- To start Xming, select Start->All Programs->Xming->Xlaunch Use the default Xlaunch options with the exception of; Start a program, Using PuTTY, and fill in the machine name (i.e. dmc.asc.edu) your userid and password. This should open an xterm window with you logged in on the DMC.

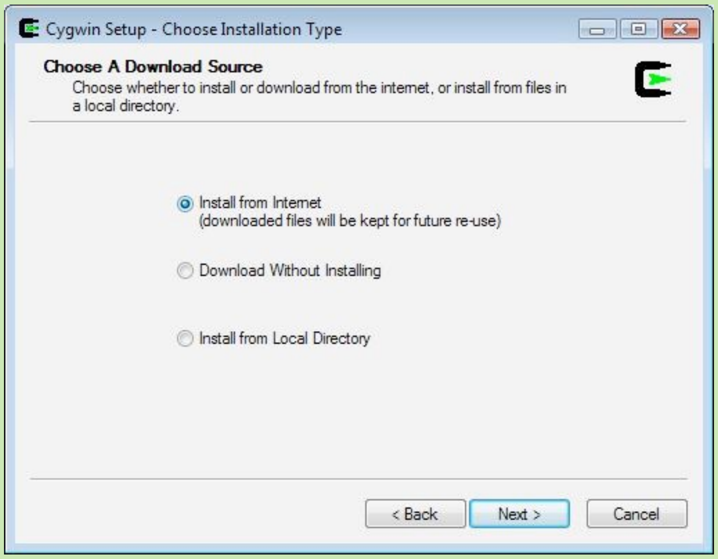
- Once you are logged in try typing “xclock”. If a window with a clock is displayed on the screen, Xming is working correctly.

Cygwin Installation

Cygwin is a more complex and powerful environment than Xming. Xming simply installs a X-Window server and ssh program. Cygwin sets up a complete Linux environment on a Windows computer. This allows you to run scripts, compile software, and run a large number of Linux applications on a Windows computer. Cygwin is a favorite of people who have a Windows computer, but would like to run Linux applications, develop software, or utilize the powerful scripting features of Linux. The following discussion of Cygwin installation and use is provided for the benefit of users that would like to use Cygwin for ssh, as an X-Window server and other functions.

You may obtain the Cygwin setup.exe file from **http://www.cygwin.com/** . Download the setup.exe file, then execute it. The images shown in Figure 7.5 show recommended settings in the Cygwin installation. There may be additional steps to verify that it is valid software, allow it to run and unblock it’s access to the internet.

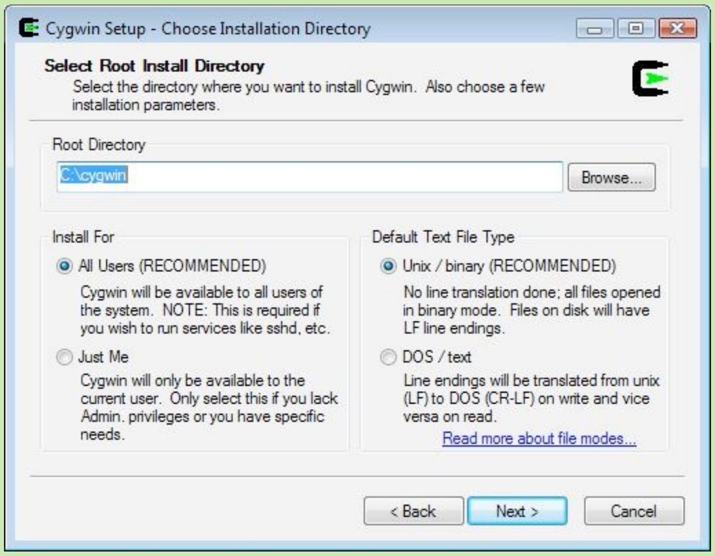
Note that the mirror site name shown here is at Virginia Tech (vt.edu). This address must be selected as shown in Figure 7.6.



Cygwin Setup - Choose Installation Type

Choose A Download Source
Choose whether to install or download from the internet, or install from files in a local directory.

[✓ Install from Internet (downloaded files will be kept for future re-use)]
Download Without Installing
Install from Local Directory



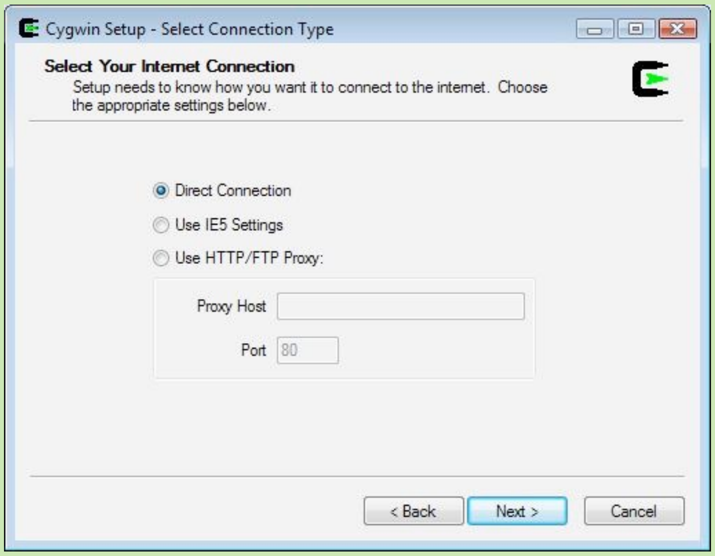
Cygwin Setup - Choose Installation Directory

Select Root Install Directory
Select the directory where you want to install Cygwin. Also choose a few installation parameters.

Root Directory: C:\cygwin

Install For
[✓ All Users (RECOMMENDED)]
Cygwin will be available to all users of the system. NOTE: This is required if you wish to run services like sshd, etc.
Just Me
Cygwin will only be available to the current user. Only select this if you lack Admin. privileges or you have specific needs.

Default Text File Type
[✓ Unix / binary (RECOMMENDED)]
No line translation done; all files opened in binary mode. Files on disk will have LF line endings.
DOS / text
Line endings will be translated from unix (LF) to DOS (CR-LF) on write and vice versa on read.
Read more about file modes...



Cygwin Setup - Select Connection Type

Select Your Internet Connection
Setup needs to know how you want it to connect to the internet. Choose the appropriate settings below.

[✓ Direct Connection]
Use IE5 Settings
Use HTTP/FTP Proxy:
Proxy Host:
Port: 80

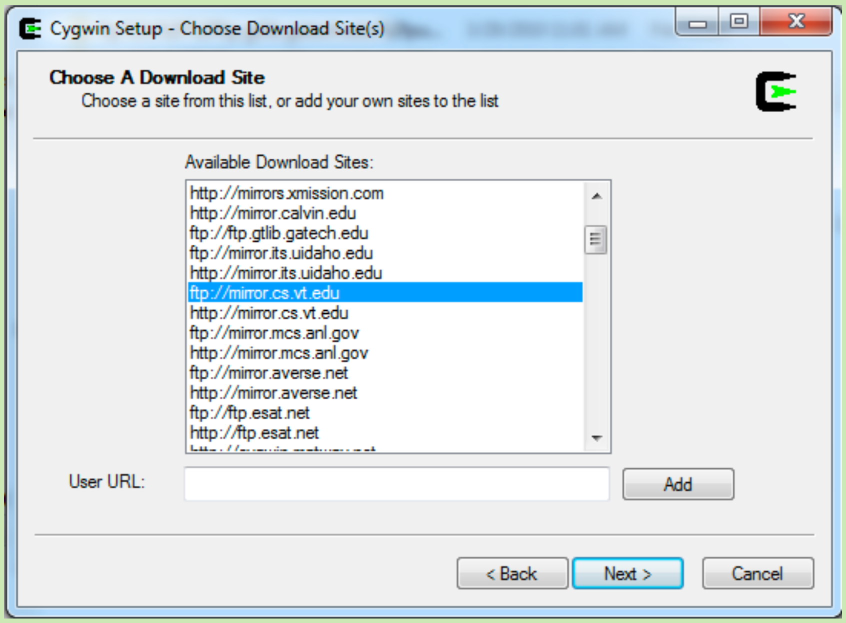
Figure 7.5

Recommended Cygwin installation options are;

Install from Internet
Default directory
Unix text file type
Direct Connection

WARNING: Cygwin will not install the X-Windows components by default. In order to get a Cygwin installation capable of running X-Windows software, you must follow these directions closely.

The default installation is a minimal installation. Additional tools to install can be added by clicking the packages (as shown in Figure 7.5). Additional tools can be added later by running the setup program again. Browsing this setup menu is a good way to find out about the available options in Cygwin. Additional packages can be added by clicking on the word to the right of the little circling-arrows icon.


Cygwin Setup - Choose Download Site(s)
Choose A Download Site
Choose a site from this list, or add your own sites to the list
Available Download Sites:
User URL:
Add
< Back
Next >
Cancel

http://mirrors.xmission.com
http://mirror.calvin.edu
ftp://ftp.gtlib.gatech.edu
ftp://mirror.its.uidaho.edu
http://mirror.its.uidaho.edu
ftp://mirror.cs.vt.edu
http://mirror.cs.vt.edu
ftp://mirror.mcs.anl.gov
http://mirror.mcs.anl.gov
ftp://mirror.averse.net
http://mirror.averse.net
ftp://ftp.esat.net
http://ftp.esat.net

Figure 7.6

The vt.edu mirror site usually gives good download speeds to locations in Alabama.

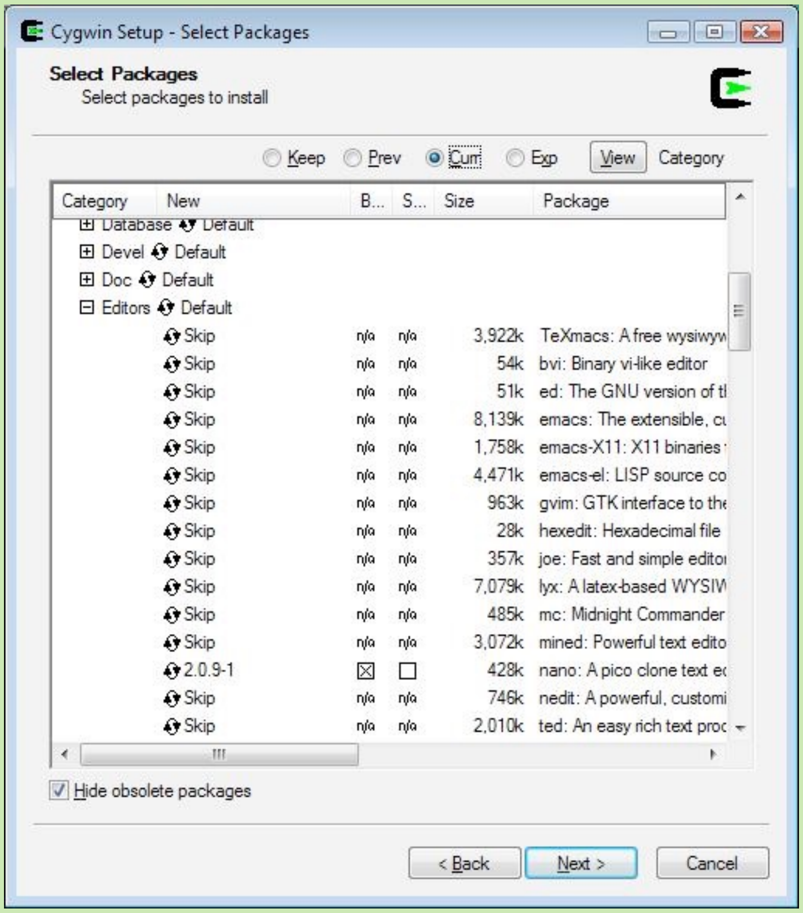
Adding the following packages to the default selections is recommended, as shown in Figure 7.7.

X11 -> Install All (the X Window server)

under Editors
Nano -> Install (an easy to use text editor)
Vim -> Install (the vi text editor for power users)

under Mail
Exim -> Install (send email from the command line)

under Net
Openssh -> Install (for SSH and X Window client)



Cygwin Setup - Select Packages

Select Packages
Select packages to install

Keep Prev [✓ Curr] Exp View Category

Category	New	B...	S...	Size	Package
⊞ Database Default					
⊞ Devel Default					
⊞ Doc Default					
⊟ Editors Default					
	Skip	n/a	n/a	3,922k	TeXmacs: A free wysiwyw
	Skip	n/a	n/a	54k	bvi: Binary vi-like editor
	Skip	n/a	n/a	51k	ed: The GNU version of tl
	Skip	n/a	n/a	8,139k	emacs: The extensible, cu
	Skip	n/a	n/a	1,758k	emacs-X11: X11 binaries
	Skip	n/a	n/a	4,471k	emacs-el: LISP source co
	Skip	n/a	n/a	963k	gvim: GTK interface to the
	Skip	n/a	n/a	28k	hexedit: Hexadecimal file
	Skip	n/a	n/a	357k	joe: Fast and simple editor
	Skip	n/a	n/a	7,079k	lyx: A latex-based WYSIW
	Skip	n/a	n/a	485k	mc: Midnight Commander
	Skip	n/a	n/a	3,072k	mined: Powerful text edito
	2.0.9-1	☒	☐	428k	nano: A pico clone text ec
	Skip	n/a	n/a	746k	nedit: A powerful, customi
	Skip	n/a	n/a	2,010k	ted: An easy rich text proc

☑ Hide obsolete packages

< Back Next > Cancel

Figure 7.7

The following optional components must be selected in order to get a Cygwin installation capable of running X-Windows and performing other tasks discussed in the supercomputer documentation.

X11
Nano
Vim
Exim
Openssh
Rxvt

under Shell
Rxvt -> Install (alternative to the dos prompt)

Other items that some users may wish to install include; emacs (under Editors), math tools, programming utilities such as make and the gcc C/C++ compiler (under Devel), and TeX (under Publishing).

Follow the installer prompts to finish the installation with default options.

Additional packages can be added into an existing Cygwin installation later by rerunning the installation program.

Create a shortcut to the X-Windows program by clicking the right mouse button on the desktop background. From the menu that appears, select New then select Shortcut from the submenu. If you used the default installation paths, the target will be **C:\cygwin\bin\startxwin.bat** . On Windows Vista systems, the desktop icons may not show up until the next time the computer is restarted.

Using Cygwin X-Windows with SSH

Start an X-Windows terminal on your PC by double clicking on the **startxwin** icon on your desktop. Alternatively, it can be started using
Start->All Programs->Cygwin-X->XWin Server It may be necessary to use either of these start options several times in order to get the software to start on a Windows Vista system. The Start menu option is used for newer versions of Cygwin on Windows 7.

Open a secure connection to your supercomputer account with a command like this

```
ssh –Y –l <user_id> uv.asc.edu
```

At this point, you should be able to run X-Windows commands and have them display on your local computer screen. You can test this by typing `xclock` . This should display a clock in a small window on your PC screen.

If this X-Windows client setup does not work, the most common problem is limitations imposed by network firewalls. Tunneling X-Window connections through ssh avoids these problems, as long as ssh access is available.

Using the screen terminal multiplexer

Tip!

This section describes the use of a terminal multiplexer. Advanced users can find this to be a valuable productivity tool. Beginning users can safely skip this section.

Consider the following scenarios:

- You have an ssh session exactly the way you want it with modules loaded, commands you are using a couple steps back in your history, and in the desired directory. It can be aggravating when the ssh connection is lost due to a network interruption.
- You want to leave an ssh session open for long periods of time so you can check on work in progress. However, the connection gets broken when the laptop is closed.
- You would like a handy key sequence to jump between multiple ssh sessions.

All of these are problems that can be solved with a terminal multiplexer. This section will describe the "**screen**" command which is included with many Linux distributions and on OS X. There are other, more sophisticated, terminal multiplexers such as tmux that have similar functionality and more, such as split screen capabilities. The older version of screen included with many Linux distributions does not have split screen capability, but newer versions do.

Here is an example of using screen.

Login via ssh.

Create a screen session named project1 with the command

```
screen -S project1
```

Change directories "**cd /opt/asn/doc**", and type "**ls**"

Create a second shell in this session by pressing "**CTRL-A**" then "**C**"

In this second session, change to a different directory, and type "**ls**"

Switch between the two shells by pressing "**CTRL-A**" then "**N**" for next.

Exit the screen session by pressing "**CTRL-A**" then "**D**" for detach. Your shells are still running. You just aren't attached to them.

Type "**`exit`**" to log out of the system, then log back in via ssh. You must log back in to the same system. Screen sessions started on dmc.asc.edu will not be visible on uv.asc.edu . Screen sessions are lost if the supercomputer system is rebooted, such during a maintenance shutdown.

Find out what screen sessions you have already in process with the command

```
screen -ls
```

One of them should be your session with a number assigned, perhaps like this 19520.project1 . Resume this screen session by typing

```
screen -r 19520.project1
```

Again, you should be able to jump between your two shells.

Screen saves environment, history, and directory. You can even leave a screen session in the middle of editing a file. However, it does not save a scroll back of information previously displayed (which is handled by the local computer).

If you lose your network connection, the screen session is still there. Sometimes it can take a while to see that you are no longer connected to it.

There are additional options for screen, such as attaching to a session that is in use from another computer. The documentation can be seen with the command

```
man screen
```

8. Working with Linux

The Linux operating system is a public domain operating system, which mostly conforms to the POSIX standard (the technical specification for the UNIX operating system). It was developed by large number of volunteer programmers around the world. The original inception and much of the project coordination is attributed to Linus Torvalds, then a student at the University of Helsinki, Finland. Both the DMC and the UV are running versions of Linux. The Linux version on the UV comes with additional tools developed by SGI. The operating system on these machines is very much like Linux operating systems on a wide variety of other computers.

Most of the differences between the operating systems on the SGI Ultraviolet and DMC are items that concern the system administrators, but are not directly visible to the users of these systems. The one exception to this is that users compiling their own MPI parallelized programs will have to follow slightly different procedures in order to use the version of MPI applicable to each system.

Linux provides the standard UNIX commands, libraries, and features, such as user shells, pipes, tees, and filters. Also included are text editors (nano and vi), communications programs (ssh, sftp, and X-Windows), and compilers (C, C++, and Fortran90). The job queuing system (SLURM) is a third party add-on to Linux systems. SLURM allows users to create a file of commands for the computer to execute at a later time rather than simply typing in the commands one by one from a terminal. Effective use of the computers, particularly for large jobs, requires use of the SLURM queue system.

Many popular Linux guides and textbooks also provide valuable information and are generally applicable to Linux. Most documentation for Linux can be accessed online via the **man** command. See the “Online Help” and “Technical Support for Users” sections of this manual for information about getting help.

Using Linux interactively is described in the following pages. The SLURM queue system is designed to accept the same commands as for interactive use. You can prepare a file for submittal to the SLURM system with one of the text editors and then place it into an appropriate batch queue.

Files and Directories

The same rules apply to both file and directory names in Linux.

WARNING: Linux is case sensitive! Uppercase is distinguished from lowercase. For example, **prog.c** is not the same file as **Prog.c** .

Directory and file names may be from 1 to 254 characters long. Periods and underlines may be used to substitute for blanks (which are strongly discouraged) to clarify what the names mean. For example, a documentation file might be designated **read.me** or **read_me** . Nearly any keyboard character may be used in file and directory names.

Tip!

When naming files and directories, it is best to avoid the characters that Linux uses in other situations, such as / \ " ' * ; - ? [ ] ( ) ~ ! $ { } < > tab and the space character, which all can potentially create confusion. Period and underscore are safe to use in file and directory names.

Directories in a Linux system are organized in a tree structure. The root directory is the top of the directory tree. The root directory is designated / Users on the system have their directories and files placed in a branch of the root corresponding to their account. Other important directories, such as apps (where applications are stored) opt (where additional information is stored) and bin (where built in programs reside), also branch directly from the root directory. These directories are designated /home, /apps, /opt, and /bin.

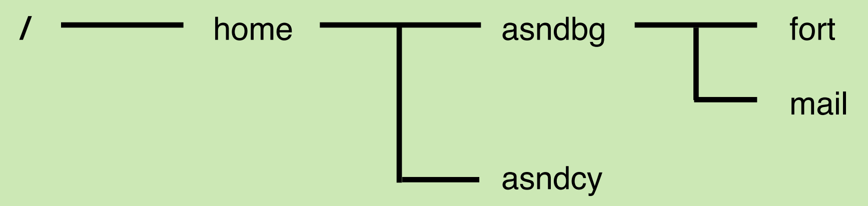
Subdirectories are indicated with a / preceded by the name of their parent directory. If there is a user subdirectory called asndbg in home, for example, the full designation for that subdirectory, starting from the root, would be **/home/asndbg** . The user asndbg might have organized Fortran programs into a subdirectory called fort, and one of those programs might reside in the file prog.f within the fort subdirectory. The full path designation for the file prog.f thus would be:

/home/asndbg/fort/prog.f

Figure 8.1 shows a sample directory tree. This example includes the directory described above, the mail directory also belonging to user asndbg and the home directory of user asndcy.

Figure 8.1

A graphical depiction of a directory tree.



Any given directory has a parent directory, in which it is a subdirectory. Shorthand for the parent directory is "**..**". To get to the directory mail from directory fort in Figure 8.1, one uses the change directory command "**`cd`**" in Linux. The cd command is used to move up one directory, then down (indicated by the slash) into a different directory like this

```
cd ../mail
```

The shorthand designation, "**.**" represents the current directory. The shorthand designation, "~" represents the users home directory.

The asterisk "*" is a wild card character which indicates any number of characters at that point. For example, to copy all files with extension .f from the parent directory to the current directory, use the following command:

```
cp ../*.f  .
```

ASC Linux File Organization

The Linux filesystem is best described with the Linux Filesystem Hierarchy. This standard can also be found on the web at **http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/**

In addition to the standard file system ASN uses the directory structure shown in Figure 8.2 for locally installed software and user accounts.

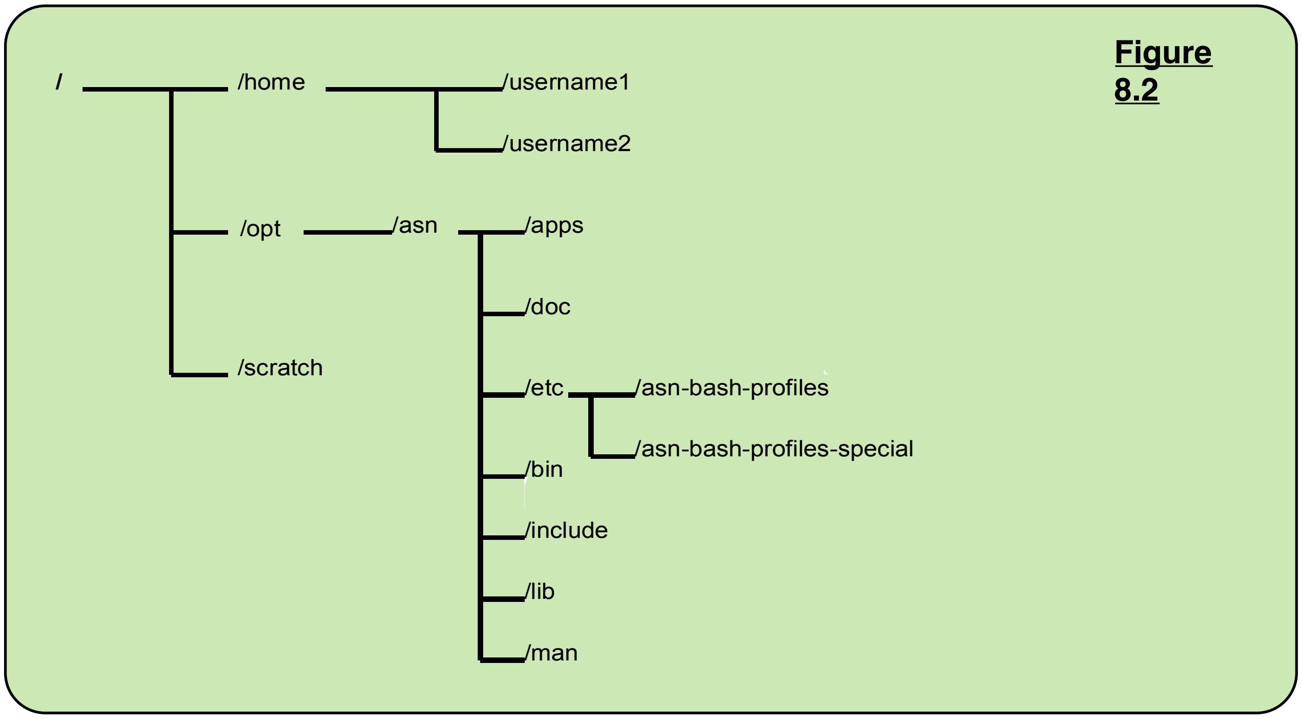


Figure 8.2

The following are some of the directories on the HPC systems at the Alabama Supercomputer Center, which are of importance to users of the system.

/home – Used for storing users' files

/opt/asn – All software we install goes here, as well as other material available to users

/opt/asn/doc – Documentation and directions for running many third party software packages.

/apps/bio/unzipped - Genomic databases

/opt/asn/etc/asn-bash-profiles – Files sourced from the users .bashrc file

/scratch – A large shared temporary file system for storing calculation data. Files on this file system are automatically deleted if they have not been accessed in 7 days. Users may create and delete files here. There are currently 45 TB of scratch, with a limit of no more than 30 TB being used by any one user.

There are some hidden files in each users home directory. These files are put there before the user ever logs in. Commands can be put in the account setup files to customize the behavior of the account, or to configure the account to use certain software programs.

Files are hidden files if the file name begins with a period. However, the "ls" command that displays a list of files can be told to include the hidden files by using a -a flag like this.

```
ls -a
```

If the user account is brand new and has no files, typing "ls -a" would still result in showing the following;

```
.               .bashrc             .ldaprc
..              .bashrc.local
.alias          .bashrc.local.dmc
.asc_queue      .bashrc.local.uv
.bash_history   .flexlmrc
```

The . (period) entry is a pointer to the current directory. The .. (period period) entry is a pointer to the directory above the current directory. Here are notes on a number of these files.

The .alias file can contain aliases, which create shortcut versions of commands. An alias can also be used to create a new default behavior of a command, or tell your account to use an alternative version of the same command. Here is an example of an alias entry to change the default behavior of a command.

```
alias file="file_info"
```

The .asc_queue file can be altered to change the default behavior of the commands used to submit jobs to the queue system. It has an explanation of each setting in comments.

The .bashrc file is the primary point of user account customization on most Linux systems. The Alabama Supercomputer Center has a site specific account configuration. Thus the .bashrc file should never be altered.

The .bashrc.local file can be altered to install customizations that will be seen on both of the high performance computers at the Alabama Supercomputer Center. The .bashrc.local.uv and .bashrc.local.dmc files can be edited to customize the behavior on just one of the high performance computing systems at the Alabama Supercomputer Center.

The files .flexlmrc and .ldaprc configure your account to use certain software packages. These files should never be altered.

WARNING: The .bashrc .flexlmrc and .ldaprc files should never be altered on the supercomputers.

Manipulating Files and Directories

The following are some of the Linux commands that are most often used to create, move, copy, or delete files and directories. Each command must be in lower case as shown. The examples shown here are the simplest, most frequently used command line options. Most of these commands have additional command line options, which can be displayed on line with the command “**`man <command>`**”.

cd

Typing “**`cd`**” without arguments puts the user in the user's home directory. With a directory name as an argument, the command moves the user to that directory. If the directory name starts with a slash, it is a full path name from the root directory. For example

```
cd /opt/asn/doc
```

If the directory name does not start with a slash, it implies a subdirectory of the current location. For example

```
cd gaussian
```

To go up one directory, use two periods like this

```
cd ..
```

cp

The “**cp**” command makes copies of files in two ways. This example makes a copy of filea and names it fileb.

```
cp filea fileb
```

The following example puts copies of all the files named into the directory.

```
cp [list of files] <directory>
```

The cp command can be given an asterisk “*” as a wild card character to move multiple files. For example, the following command would copy every file with a name ending in .c to the directory named source.

```
cp *.c source
```

The cp command makes a second copy of the file, unlike the mv command which leaves only one copy of the file but moves it to a new location.

file

The “**file**” command examines the contents of a file to see what type of file it is. This is called with the file name as the only argument like this.

```
file <filename>
```

If the program is compiled to run on the DMC, the file command will show that is is an x86-64 architecture executable like this.

```
asndcy@dmc:x86_64> file environ
environ: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), for GNU/Linux 2.4.0, statically linked, not stripped
```

file_info

The “**file_info**” command is used the same as “**file**”. file_info shows all of the information that file does, plus identifies the presence of SSE and AVX instructions in an executable. However, file_info takes much longer to run that the file command.

ln

The “**ln**” command creates a symbolic link when the **-s** flag is included. A symbolic link is a pointer so that a file or directory will appear to be in a second location. In Windows, symbolic links are called shortcuts. Symbolic links can be used to make frequently used directories or files conveniently accessible. For example, the following command makes a symbolic link so that the documentation directory appears to be a subdirectory of your home directory

```
ln -s /opt/asn/doc $HOME
```

After typing the above command, you can type “**cd doc**” from your home directory and you will see all of the documentation directories.

ls

The “**ls**” command lists the files in the current directory or the directory named as an argument. The ls command can be used without arguments, but there are many options available. For example;

```
ls -a  [directory]
```

lists all files, including files whose names start with a period. If a directory name is not specified, ls lists files in the current directory. Files that have names starting with a period are hidden unless the -a option is given. These are account setup files which should be altered seldom if ever.

```
ls -l  [directory]
```

lists files in long form: links, owner, size, date and time of last change.

```
ls -C  [directory]
```

lists files in columns using full screen width.

```
ls -R  [directory]
```

recursively lists files in current directory and all subdirectories.

mkdir
mkdir makes a new subdirectory in the current directory. For example;

```
mkdir fort
```

makes a subdirectory called fort.

more (or less)
The "**`more`**" command is used to display a text file. For example, filea.txt can be displayed with the command

```
more filea.txt
```

Within more, the next page of text can be viewed by pressing the space bar. The next line of text can be viewed by pressing the Return key. You can also use the "**`f`**" key to move forward and "**`b`**" key to move backwards by a page. Typing "**`q`**" exits more. Some Linux systems have a "**`less`**" command that is an enhanced version of "**`more`**", but the differences are relatively minor.

mv
The "**`mv`**" command moves or changes the name of a file. The following example;

```
mv filea fileb
```

changes the name of filea to fileb. If the second argument is a directory, the file is moved to that directory but keeps the same name.

When the mv command is used, there is only one copy of the file, in contrast to using the cp command which makes a second copy of the file.

pwd
The "**`pwd`**" command returns the name of the current working directory. It tells where the current directory is in the directory tree. No arguments required.

rm
The "rm" command removes each file in a list from a directory. For example, the file program.cc can be deleted with the command.

```
rm program.cc
```

All of the files ending in .cc can be deleted with the command

```
rm *.cc
```

WARNING: Using the asterisk with the rm command can result it deleting much more than desired. Before using rm with an asterisk, it is advisable to check a listing of which files will be deleted with the ls command like this “**`ls *.cc`**”.

Option -i causes rm to inquire whether each file should be removed or not, like this;

```
rm -i *.cc
```

Option -r causes rm to delete a directory along with any files or directories in it. For example;

```
rm -r source
```

rmdir

The “rmdir” command removes an empty directory from the current directory. For example;

```
rmdir fort
```

removes the subdirectory named fort (if it contains no files).

To remove a directory and all files in that directory, either remove the files first and then remove the directory or use the “rm -r” command described above.

tar

The “**`tar`**” command was created to store many files in one place, and thus tar stands for “Tape ARchive”. Although the ASC HPC systems do not use data storage tapes, this command is incredibly useful for packaging many small files into one larger file.

It is also common to use the -z flag to gzip compress the files to take less disk space at the same time. Although it is not a requirement, files created by the tar command are often given the extension .tar if not compressed or .tar.gz or .tgz if gzip compressed as well.

The **`-x`** flag specifies extracting files from a file.

The **`-v`** flag activates a verbose mode so that the command will tell you what it is doing.

The **-f** flag indicates that the next argument will be a file name for the tar file.

If you downloaded the source code for a piece of software off the the internet, it is often in a tar file. People in the business refer to tar files of source code as tar balls. You would extract the files from this file with a command like this

```
tar -xzvf filename.tgz
```

If you have the files associated with a research project in a directory named myproject_directory and would like to pack them into a single file so they can be moved back to campus for permanent storage, the command would be

```
tar -czvf myproject.tgz myproject_directory
gunzip -t myproject.tgz
```

Warning: There is one issue with the tar command. If something like a disk quota or CPU time limit kills the process that creates a tar file, there may be no error message to tell you there is a problem. The gunzip command above verifies that the file is good (no output) or gives an error message. If tar fails because of CPU time limits on the login node, run it through the queue. This is usually necessary for more than a few gigabytes of data.

Frequently Used Linux Commands

The previous section of this manual listed commands that are used for manipulating files and directories. This section lists a number of other Linux commands. This is by no means a comprehensive listing. There are thousands of Linux commands, many of which are only used by system administrators, and a few of them a career systems administrator will have never had occasion to use. However, the commands listed here are the small set of commands that a user of a Linux system will utilize 99% of the time.

As in the previous section of this manual, the examples shown here are the simplest, most frequently used command line options. Most of these commands have additional command line options, which can be displayed on line with the command "**man <command>**".

apt-get

apt-get is a command for installing software on computers running a Debian variant of Linux such as Ubuntu. The supercomputers use SUSE Linux, which does not use apt-get. If you think you need apt-get, try reading the documentation on the software package, provided through the "**`ascdocs`**" command, or contact the HPC technical staff at **hpc@asc.edu**

awk

The "**`awk`**" command can be used to run an entire script written in the awk language, or used as a command that runs one line of awk line functionality from the command prompt or in a shell script. Most often, awk is used to insert a single line of awk script code in a shell script. This is done because Bourne shell or bash shell scripts are easier to write, but lack awk's facility for floating point mathematics and formatting columns of data. This is very powerful tool for writing Linux scripts, but too complex to use to be covered in this manual. Users intending to use Linux scripts heavily should look up the awk command in a book on shell programming. Several good books on shell programming are referenced in the bibliography at the end of this manual.

cat

The "**`cat`**" command concatenates (combines) and prints the files given as arguments. The output goes to the standard output, which is usually the screen. For example, the following command would print the contents of filea.txt to the screen.

```
cat filea.txt
```

Note that printing a text file to the screen is a sensible thing to do. There is seldom any reason to print the contents of a binary file, such as an executable program. Indeed printing binary files to the screen often resets terminal settings, thus making it necessary to close your session and reconnect to the supercomputer.

If no file is given, input is taken from the keyboard. A **`CTRL-D`** terminates keyboard input and brings you back to the command prompt.

Often output is redirected with the operator >. For example;

```
cat filea fileb > filec
```

concatenates filea and fileb and places the result in filec.

chmod

The "**chmod**" command changes the file permission status of a file. Permissions may be granted to read, write, or execute the file. That permission may be given to the user, the user's group, or to the world. When one uses the "**ls -l**" command, these permissions are listed at the left as a series of r's, w's, or x's, with - indicating that permission is not granted.

For example, -rwxrwxrwx indicates read, write, and execute permission is granted to all three groups; -rw-r----- grants the owner of the file read and write permissions, the members of the owners group read permission, and no access to users not in the owners group. The chmod command changes the status of these permissions. The form is the following:

```
chmod [ugo] [+-] [rwx] files
```

The flags u, g, or o stand for user, group, or others. The + or - indicate whether the permission is to be given or denied. The r, w or x indicate whether read, write, or execute permission is to be given. For example;

```
chmod +x myfile
```

will give myfile execute permissions for everyone. Or the command;

```
chmod ug+x filea
```

will give filea execute permission to the user and the user's group.

When a new file or directory is created, the default permissions granted are those specified by the user's umask (or default permission mask). The operating system has a default umask, which is set on all user accounts. The user can change the umask value to suit the user's needs. See the man page on umask for details by typing "**man umask**".

date

The "**date**" command outputs the date and time. The date command does not require any arguments.

echo

The "**echo**" command repeats whatever text is given to it on the standard output. For example,

```
echo Whats up, doc?
```

will print `What's up, doc?` on the screen (default for standard output). If echo is used inside of a script (discussed later) there would need to be double quotes around the words to print.

The echo command can also be used to display the values of environment variables. Environment variable names are denoted by a dollar sign, like this

```
echo $PATH
```

finger

The "**finger**" command allows users to see who owns a given user account. This is done by typing "finger" followed by the account name, like this

```
finger asndcy
```

grep

The "**grep**" command finds lines of text that contain a given string. For example, to find the lines containing the word "energy" in the file water.out use

```
grep energy water.out
```

The "-i" flag to grep specifies a case insensitive search.

head

The "**head**" command prints the first part of a file given to it as an argument. For example, the following would print the first 30 lines of the file water.txt

```
head -30 water.txt
```

By default head outputs 10 lines of text.

ldd

The "**ldd**" command prints the list of dynamically linked libraries called by a program, or another library. For example

```
ldd /opt/asn/bin/f2c
```

This is used to analyze the problem when a dynamically linked program can't find it's libraries.

mail

The "**mail**" command invokes an electronic mail system. The mail command can be used to send mail. The ASC supercomputers do not allow mail to be received. For example

```
mail myname@my_school.edu
```

sends mail to your campus email. The user names are arguments to the command. Users are prompted for a subject after the command. The letter should be entered line by line and finished with a line containing only a period or a **CTRL-D** character.

Alternatively, the redirection operator < may be used to route a letter prepared with an editor to mail. For example

```
mail myname@my_school.edu <errors.txt
```

would send the previously prepared letter to the your campus email address.

Tip!

For contacting the technical support staff, it is better to use your regular email account, which allows them to reply to you. However, the mail command can be convenient for writing scripts that automatically email information to your regular email account.

man

The "**man**" command provides online documentation for all UNIX commands and utilities, making quite detailed descriptions instantly available. The commands or utilities are arguments to man. For example;

```
man cat
```

will give a summary of the use of the concatenate command.

Another very useful feature is illustrated in this example

```
man -k <keyword>
```

which will give information on all commands relevant to the given keyword.

nm

The "**nm**" command gives a list of functions in an executable or library. For example

```
nm /usr/lib/libjpeg.a
```

This is used to determine which libraries need to be linked when a compile command gives an unresolved symbol error.

passwd

The "**passwd**" command allows the user to change her password. No arguments are required for the passwd command. The passwd command will prompt the user to supply the old and new passwords.

The passwd command in use at the Alabama Supercomputer Center is slightly different from the way this command is set up on other systems. The command at this center gives the user the option of inputting a new password or letting the passwd program generate a secure password. It only accepts passwords that cannot be broken by common computer hacking utilities. If an invalid password is entered, it will explain why the password is invalid. It is necessary to use a word that is not in the dictionary and include a number or non-letter symbol.

ps

The “**ps**” command shows what processes are running in the current shell. The ps command can be run without any arguments.

Reminder

Note that the ps command does not show processes belonging to jobs submitted through the queue system. Those jobs are running on other nodes in the cluster and thus can be seen with “squeue” but not with “ps”.

sed

The “**sed**” command copies files (standard input by default) to standard output, edited according to a script of commands. This is very powerful tool for writing Linux scripts, but too complex to use to be covered in this manual. Users intending to use Linux scripts to do string processing should look up the sed command in a book on shell programming. Several good books on shell programming are referenced in the bibliography at the end of this manual.

sort

The “**sort**” command can be used to sort lines of text. The simplest form of this command would look like this

```
sort <filename>
```

which sorts the lines in the file in lexicographical order, meaning that the alphabet is extended to include special symbols and digits.

A command line flag can be put between the word “sort” and the file name. Here are some useful flags for controlling the behavior of sort.

-b	ignores initial blanks
-d	uses a dictionary order, without special digits or symbols
-f	treats upper and lower case as equals
-n	sorts numerically
-r	reverses the sort order

One frequent use of sort is to put the results in a different file, like this:

```
sort -o <output_file> <input_file>
```

sudo

The "**sudo**" command allows a user to run a single command from the administrator account (called the root account on Linux). sudo is used to install software and perform other tasks on personal computers running some versions of Linux. However, the Alabama Supercomputer Center has a technical staff to do system administration on the supercomputers, so there is no reason that you will ever have to use sudo from a user account. If you think you need sudo, try reading the documentation on the software package, provided through the "**ascdocs**" command, or contact the HPC technical staff at **hpc@asc.edu**

tail

The "**tail**" command prints the last part of a file given to it as an argument. For example, the following would print the last 30 lines of the file water.txt

```
tail -30 water.txt
```

By default tail outputs 10 lines of text.

top

The "**top**" command can be typed without arguments to see what processes are using the most CPU time on the current node. The top command is exited by typing the letter "**q**".

Reminder

Note that the top command does not show processes belonging to jobs submitted through the queue system. Those jobs are running on other nodes in the cluster. The top command only shows processes on the current node, the login node, and thus cannot see queued jobs. The queued job status can be seen with the "squeue" command, which is described in the section of this manual about using the queue system.

wc

The "**wc**" command counts lines, words, and characters in files. By default, the wc command will return counts for all three. The parameters -l, -w, and -c will limit the report to lines, words, and characters, respectively. In the simplest form, the wc command can be given just a file name, like this;

```
wc .bashrc.local.dmc
 100  276 3399 .bashrc.local.dmc
```

The output indicates that the file .bashrc.local.dmc. contains 100 lines of text, 276 words, and 3399 characters.

which

The "**which**" command gives the full path to the location of an executable file. There is also a locally written "whence" command that works with aliases as well. For example;

```
which grep
```

who

The "**who**" command lists the login names of all users currently on the system, their terminals, and when they logged on. The who command does not require any arguments.

Linux also allows users, or at least their shell scripts, to ask existential questions with the following commands;

```
who am i
whoami
```

Using Pipes and Regular Expressions

The previous section of this chapter discussed using Linux commands by typing single command at the command prompt. One of the great sources of convenience and power in the Linux operating system is the ability to easily use multiple commands together.

Let's look at an example of how to use this ability. Login on the supercomputer, and check what software is available by typing the following.

```
ls /opt/asn/doc
```

The /opt/asn/doc directory has a subdirectory for every piece of software. Because of this, the ls command found hundreds of software packages. It is inconvenient to wade through all of that information to find a specific application. However, Linux gives a convenient way to find just the desired information. For example, to find information relevant to the Gaussian software type the following;

```
ls /opt/asn/doc | grep gauss
```

What happened here? The vertical bar "|" is called a pipe. That pipe tells Linux to take the output from one command and feed it into the input of another command. A large percentage of the commands in the previous section of this chapter can be used in this way. The "grep" command outputs only the lines of text containing its search query, called a regular expression. In this example, the regular expression is "gauss".

For an example of piping more than two commands together, we could count how many directories are found by typing the following;

```
ls /opt/asn/doc | grep gauss | wc -l
```

Here the word count command (wc) is used to count how many lines of text are output by the previous commands. Likewise pipes can be used to combine even 20 commands.

The grep command has a very rich set of options for selecting which lines to output. The -v flag causes grep to display everything that does not fit the pattern.

The regular expression should be enclosed in single or double quotes if it contains one or more blanks. A regular expression contains the usual ASCII characters, but some characters have special meanings, depending on their location within the expression. The characters with special meaning are **`. * [ ] \ $`** and **`^`**.

The period substitutes for any character in one position in the expression. (Expression matching with the ls command uses a question mark in place of a period.) For example, .abc will match with aabc, babc, 7abc, and so on, and a.bc matches a1bc, a2bc, azbc, etc.

Multiple occurrences of a character may be matched with an asterisk. For example, a*bc will match patterns with zero or more characters between the "a" and "b" characters, such as occurrences of axbc appbcd abc aaaaaaaaaaaaaabc etc.

A backslash before a special character will remove the special meaning from the character, so that it can be matched for itself. For example, **`\*`** within a regular expression will match * in that position.

The carat ^ means the start of a string or line. The dollar sign means the end of a string or line.

Redirection of Input and Output

Standard input and output may be redirected for any process with the use of the symbols **<** and **>**.

Putting the following after a command "**`<file.txt`**" redirects standard input (by default, the keyboard) so that input is taken from the file named after the symbol. This can be used to automate tasks that normally require input from the keyboard. The exact input from keyboard, including returns, can be put in a file and piped into the command.

Putting the following after a command "**`>file.txt`**" redirects standard output (by default, the screen) to the file named after the symbol.

WARNING: When output is redirected to a file with **`>filename`** any information currently stored in the file specified is overwritten and lost.

Here is an example of putting the output of the ls command into a file.

```
ls >ls.txt
```

The special symbol **>>** appends new data to the named file. Thus the following command would double the size of the ls.txt file from the previous example and result in having two copies of the same information in the file.

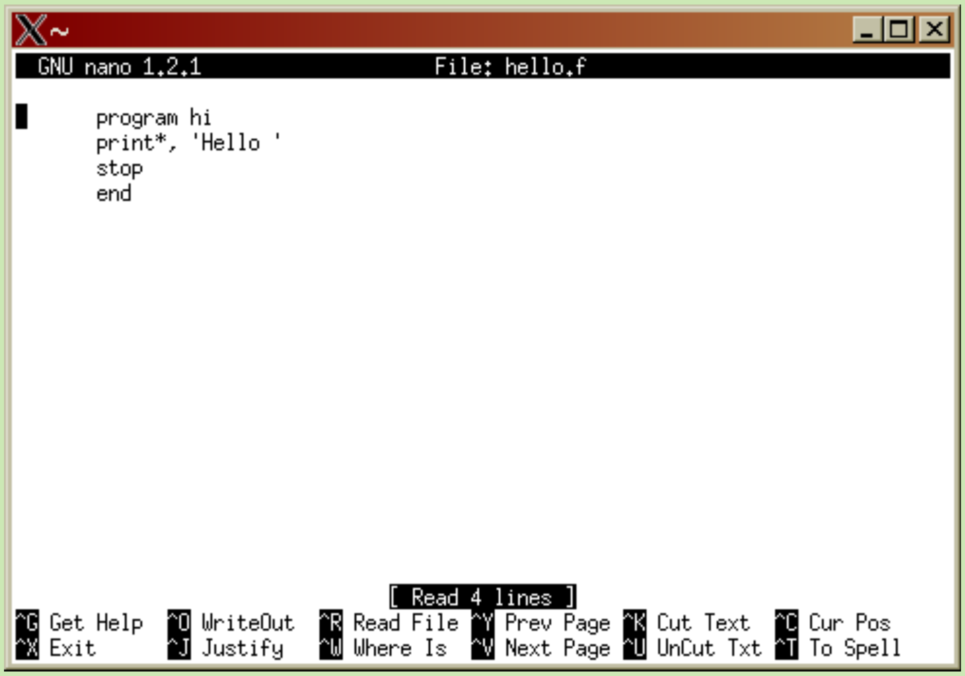
```
ls >>ls.txt
```

Putting the following after a command **`>&file.txt`** redirect both stdout and stderr to the file.

Putting the following after a command **`<<xxx`** redirects input until it encounters a line with only xxx, beginning in column 1. This can be used in scripts to put an input file inline within the script.

Introduction to the nano Text Editor

Programming language source code, scripts, and the input files used for many applications are text files. It it convenient to be able to create, edit, and view text files via an ssh connection to the supercomputers. This is done with a text editor. A text editor is like a word processor that only creates text files, similar to Notepad on a Microsoft Windows system or TextEdit on a Macintosh.



```
GNU nano 1.2.1                    File: hello.f

      program hi
      print*, 'Hello '
      stop
      end

                                  [ Read 4 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Txt ^T To Spell
```

Figure 8.3

The nano text editor.

There are a number of text editors available for Linux systems. This manual will discuss two of those editors named “nano” and “vi”. The nano text editor is an excellent choice for first time users. Power users often expend the additional effort to learn the more powerful vi editor. This section of this manual discusses nano. The next section of this manual discusses vi.

The nano text editor is an easy to use editor for creating ASCII text files from a terminal window. The nano program is not included with all distributions of Linux and Unix, but it can be downloaded free from **http://www.nano-editor.org/** On the ASC supercomputers, nano is in users default path.

To invoke nano type the following command:

```
nano [options] <filename>
```

A complete list of the nano command line options can be seen by typing “**man nano**”. This description discusses the most frequently used options.

Most frequently, users edit files without using any of the optional command line arguments. For example, the file myscript.sh could be edited with the command

```
nano myscript.sh
```

The nano program fills the whole terminal window, as shown in Figure 8.3. The top line shows the nano header including the name of the file you are editing in reverse video. The second line of the screen is left blank to be more visually appealing…. this does not mean that the first line of your file is a blank line. The text of your file is displayed in the middle of the screen. The bottom of the screen displays most used nano commands in inverse video. The carat ^ means to hold down the **control** key on the keyboard while typing the following letter. The prompts at the bottom of the screen display upper case characters, but the same letter in lower case works as well.

The cursor position is indicated by a single character in inverse video. Move the cursor around the screen with the arrow keys. Any text you type will be inserted to the left of the cursor. You can position the cursor one line beyond the end of the file to append new text, even though the file may not have a blank line at the end. The Backspace key deletes text to the left of the cursor, and the Delete key deletes the character that the cursor is positioned over.

The changes you make are now written out to the file until you tell nano to write them. Typing **`CTRL-O`** results in writing out to the file.

If you type **`CTRL-X`** to exit without first writing the changes to the file, nano will ask if you want the changes saved to the file.

EXAMPLE

Here is a sample nano session.

The user should be in the directory in which the new file is to be located.

To begin editing the sample file from the previous exercise called hello.f in the current working directory, type:

```
nano hello.f
```

The editor will display a screen like the one in Figure 6.3.

Position the cursor at the end of the word **Hello** then type another word, say **`World`**. Type **`CTRL-X`** to exit nano. The nano program will ask you want to save the changes that were made to the hello.f file. Type **`Y`** to save the changes.

Recompiling the program will now give an executable that prints out "**Hello World**".

Introduction to the Screen-Oriented Editor vi

The “vi” editor is a very powerful text editor for Unix and Linux. The vi command is part of the Unix and Linux operating system, thus it should be on every Linux computer. The vi editor is designed to be used over a text only connection, such as ssh. In order to get a very powerful editor without the benefit of a mouse, and pull down menus, vi uses a rather complex set of keyboard commands. This makes vi powerful, but difficult to learn.

Tip!

Users who will spend a significant amount of time creating text files on Linux systems (i.e. systems administrators and programmers) will find the effort expended learning vi to be well worth the time invested. Users who only occasionally create or edit small files on Linux computers are advised to use the nano editor until the time comes that they find nano no longer meets their needs.

The vi editor has a command mode and a text input mode. When vi is first started it is in command mode. In command mode any key pressed is interpreted as a command to perform some function. Entering commands to insert, append, or overwrite text put the editor into text input mode. When the editor is in text input mode, any text typed is put into the file being edited. The **esc** key is pressed in order to exit text input mode and resume command mode.

The following narrative intersperses an example vi session with a listing of vi commands.

EXAMPLE

Sample vi session

Edit a file in the directory where the file is located or should be located (in the case of a new file). Begin editing a new file by typing the following:

```
vi file.txt
```

Insertion of text can be done after issuing the i command by typing the following;

```
i
Augy had a little lamb,
its fleece was white as snow (etc)
<esc>
```

Move about in the file with the **h j k** and **l** keys when in command mode (the <**esc**> returns one to command mode).

Cursor Positioning Keys--in the Command Mode

`h`	Moves cursor one character to the left.
`j`	Moves cursor down one line anywhere in text.
`k`	Moves cursor up one line anywhere in text.
`l`	Moves cursor one character to the right.

Move with these keys to the "**f**" in "**fleece**" in the second line, and type **`cw`**. The word "**fleece**" disappears, replaced by a $. Type "**`fur`**" in substitution then **esc**.

To save the file, with changes, type **`:wq`** This exits to the shell.

Entering text input mode--End this mode with an <esc>

`a` Append text after the cursor. Enter as many lines and<return>'s as needed.

`i` Insert text before the cursor. Enter as many lines of text and <return>'s as needed.

`o` Open a new line below cursor. Ready for the text input.

`O` Open a new line above cursor. Ready for the text input.

`R` Replace characters on the screen, starting at the cursor, with any characters typed.

These commands, after execution, return the editor to the command mode

`r` Replace a single character under the cursor with a single character that is typed.

`/foo` Search sequence; looks for next occurrence of pattern following / (in this case, the word "foo").

`?foo` Search sequence; like /, but searches backwards from the cursor.

`n` Used after / or ? to advance to the next occurrence in the buffer of the pattern.

`u` Undo the last command.

`U` Undo all the changes to the current line.

`x` Delete character highlighted or underlined by the cursor.

<**`del`**> or **`#`** or **`CTRL-H`** This backspace feature of the shell also works in the editor. These commands move the cursor character by character, left within a line, erasing each character from the buffer.

`CTRL-F` Scroll or page the screen forward one page at a time.

`CTRL-B` Scroll or page the screen backward one page at a time.

`CTRL-D` Scroll or page the screen down one-half page at time.

`CTRL-U` Scroll or page the screen up one-half page at time

`CTRL-G` Identify the line where the cursor is located by line number.

`nG` Position the cursor at line n in the file.

`:%s/text1/text2/g` Replace all instances of text1 with text2 throughout the entire document.

Operators in the Command Mode

`d` Delete indicated text starting at the cursor. For example, use dw to delete a word and dd to delete a line; 3dd deletes 3 lines. Deleted text is stored temporarily in a buffer whose contents can be printed out with the p command. Also, d can be used with named buffers in the manner described for y command.

`c` Delete indicated text starting at the cursor and enters Text Input Mode. Thus, cw deletes from the cursor to the end of the word, allowing users to add text between those positions.

`y` Copy indicated text, starting at the cursor, and stores it in a buffer. There are nine unnamed buffers (1-9) that store the last nine delete or yank operations, and 26 named buffers (a-z) that can be used for storage. The double quote mark (") is used to tell the editor the name of the buffer. Thus, "cy$ will store text from the cursor to the end of the line in a buffer named c.

`p` Inserts "delete" and "yank" buffer contents after the cursor or on the next line. Command p puts the last item yanked or deleted back into the file just after the cursor, and "cp will put the contents of buffer c after the cursor.

Scopes for Use with Operators

e The scope from the cursor to the end of the current word; e. g., if the cursor is on the "**u**" in "**current**", and the user types **`de`**, then "**urrent**" is deleted.

w The scope is from the cursor to the beginning of the next word, including the space.

b The scope is from the letter before the cursor, backwards, to the beginning of the word.

$ The scope is from the cursor to the end of the line.

O The scope is from just before the cursor to the beginning of the line.

) The scope is from the cursor to the beginning of the next sentence. A sentence is ended by "**.**", "**!**", or "**?**", followed by 2 spaces or by an "end of line" (provided by the <return> key).

(The scope is from just before the cursor back to the beginning of the sentence containing the cursor.

} The scope is from the cursor to the end of a paragraph. A paragraph begins after an empty line.

{ The scope is from just before the cursor back to the beginning of a paragraph.

Leaving the Editor

`<esc>:w` Write the contents of the buffer into the current file of the same name. Can write to a new filename. Also, can send partial buffer contents using line numbers, such as A:3,10w popcorn.

`<esc>:q` Quit the buffer after a :w command.

`<esc>:wq` Write and quit, placing buffer contents in file.

`<esc>:q!` Quit editor without making changes in file. Dangerous.

`<esc>ZZ` Write and quit, placing buffer contents in file.

Using the ex editor while in vi

`:` Generate a colon (:) prompt at the bottom of the screen and let users make one ex command. Users are returned to the vi mode when the command finishes execution.

`Q` Quit vi and place users in the ex editor, giving users a command mode prompt, the colon (**:**) at the bottom of the screen. Users can get back to vi while in the command mode.

When in Doubt

`esc` Put users in the command mode.

There are many good books with tutorials on the use of the vi editor. One such is published by O'Reilly & Associates, Inc. and is titled "Learning the vi and Vim Editors" by Arnold Robbins, Elbert Hannah, and Linda Lamb.

Shell Scripts

This manual has already discussed a number of features that make Linux a very powerful command line environment. One of the best features of all is that anything that can be typed on the command line can be automated by putting the same commands in a shell script.

Users that only run software already installed on the supercomputers may not need to read this section of the manual. Users who write their own software, want to do system administration, or find themselves doing a large amount of repetitive typing should read on.

A shell script, or just script, is different from a binary program in that the script is not compiled into an executable, machine language file. A script is a text file which contains commands in some scripting language. The script language interpreter executes the commands as it reads that text file. The advantage of shell scripts is that they are easy to write and automate tasks. The disadvantage is that they run much slower than compiled programs and are thus not an appropriate way to write the main program that uses thousands of hours of CPU time. Shell scripts automate many tasks within the operating system, and for the convenience of the users. Often scripts wrap the computationally intensive program, meaning that the shell script stages input data to the working directory, sets up environment variables, creates a nodes list, calls the computationally intensive program, then copies the results back to the users home directory.

There are thousands of scripting languages. Here are just a few to consider. Bourne shell scripts can be used to automate tasks on both Linux and Unix computers. Bash shell scripts have some enhanced features, but are specific to Linux and not on most Unix systems by default. Perl is a scripting language specifically for generating reports of data with a tabular format. Python is a scripting language powerful enough to approach the level of capability of a compiled programming language, thus making it sometimes ideal for rather large scripting projects that aren't quite big enough to justify going to a compiled language. The rest of this section of the manual will discuss Bourne shell scripting only.

A shell is the process which interacts with the commands issued by a logged-on user. Each logged-on user has his own shell. A new shell (and even multiple new shells) can be created by a user with the command "**`sh`**". The command can also execute shell scripts.

For example If the file named "check" contained the lines

```
#
date
who
```

then the command

```
sh check
```

would create a separate shell from the one the user is operating under, execute the commands (printing the date and the current users on the screen), and return to the user's original shell. The new shell would disappear as soon as the commands were finished executing.

To create a check file that can be executed directly, without typing sh every time, write it like this.

```
#!/bin/sh
date
who
```

Then make it be an executable file with the command

```
chmod +x check
```

Now you can just type "**check**". This shell script works as though it is any other program, even though it is not a compiled program.

Another useful scripting feature is to use the accent character (backwards single quote at the top left of the keyboard) to feed the output from a command into a variable in the script, like this.

```
myname=$(whoami)
```

This can be combined with the pipe discussed previously to make the following script, that we named myscratch

```
#!/bin/sh
#  This shows only my data in /scratch
myname=`whoami`
ls -l /scratch | grep $myname
```

Once the chmod command is used to make myscratch executable, you can type "**myscratch**" and see a listing of only your own jobs in the queue system. This is much more convenient than wading through thousands of lines of output to find the relevant information.

Scripts can contain many of the constructs found in other languages such as variables, loops, arrays, conditional statements ("if" statements), and subroutines. Some scripting languages have very rich feature sets, such as Python having more string processing functions than most compiled languages. Many books on scripting with titles containing phrases like "Unix Shell Programming", "Learning BASH", "Python", or "Linux Shell Scripting" can be found at most book stores and libraries. A selection of these books are listed in the Bibliography at the end of this manual.

9. Working with the Queue System

Nearly all supercomputing facilities use a job queue system. A job queue system is similar to a printer queue in that a pile of work can be submitted, then the queue system software will start each job when the necessary resources become available. In the case of a job queue system, the resources being managed are computer processors, memory, and sometimes software licenses.

Tip!

The queue system is the researcher's friend. If you want to get a large amount of work done, the best thing you can do is learn how to utilize the queue system effectively.

A queue system is a valuable tool for users. A pile of jobs can be submitted on Friday night to be run when resources become available. Before the invention of queue systems, supercomputer users would frequently find themselves having to log in at 2:00 a.m. on a Sunday morning because that was when the resources were available. The queue system also guarantees that the users job will get the number of CPUs and amount of memory that they requested when the job was submitted.

Reminder

Programs run interactively (without using the queue system) on the login nodes are limited to 10 minutes of CPU time. After 10 minutes, interactive jobs are automatically killed. Any job larger than this, or using more than 2 cores, must be run through the queue system.

The queue system at the Alabama Supercomputer Center is a SLURM queue system. SLURM is a more recently created queue system, which is designed for improved security and to scale up to manage very large computing systems.

WARNING: If you see error messages when you login via ssh, those same errors will prevent your jobs from running correctly through the queue system.

The SLURM queue system readily facilitates integrating multiple clusters under one queue system. Thus a job submitted from the dmc.asc.edu login node may actually end up running on the UV compute node. The queue scripts provided for running individual applications have been written to allow the software to run on any node where the specific application can be run. The queue scheduler will run the calculation on the node that gives the best turn around time. Users compiling their own applications must either specify the cluster where the application is compiled, or compile for both clusters and create a run script to select the correct version of the software. Submitting user written software to the queue system is discussed later in this manual.

Selecting a Queue

There are a number of queues available. A list of the queues available can be displayed with the "**`qlimits`**" command. The qlimits command can be called without any arguments, or with a "-a" flag. Calling qlimits gives an output like the following

```
Queue                     Wall Time       Mem  # Cores
------------------------ ---------- -------- --------
express                     4:00:00      16gb      1-4
small                      60:00:00       4gb      1-8
medium                    150:00:00      16gb     1-16
large                     360:00:00     120gb     1-64
huge                      360:00:00     256gb   64-128
class                      16:00:00      64gb     1-64

Interactive limits:
                          Wall Time       Mem  # Cores
-----------------------  ---------- -------- --------
INTERACTIVE                00:10:00       4gb        2
```

The "Wall Time" column in this output gives the amount of time that can be requested in the format HH:MM:SS. The "Mem" column shows the maximum memory that can be requested, which is a total for all CPUs. The "# Cores" column shows how many CPU cores can be requested by a job in that queue

The express queue is accessible to everyone on the system. Express jobs are limited to 4 hours of wall clock time. Also, only one job per person can be in a run state in the express queue. This makes it unusable for the majority of research calculations. However express jobs get into a run state almost immediately.

Tip!

An old supercomputer users trick is to submit a test job to the express queue, let it run a few minutes, then kill it. This is done as a check on whether the input file is constructed correctly, as incorrect inputs typically cause the calculation to fail within the first few minutes. Once this check on correct inputs is made the job can be submitted to the appropriate queue to allow it to run to completion.

The small, medium, large, and huge queues are available to all users of the system. These queues are used to run the majority of the work on the supercomputers.

The “class” queue is available for working on course homework assignments. It is typically only accessible to class accounts. These accounts only exist for the duration of the semester, and contain the letters “cls” in the account name. Instructors wishing to get class accounts for their student should contact the staff at the Alabama Supercomputer Center by emailing **hpc@asc.edu**

The “huge” queue will run only one job at a time per user. It can queue to the DMC, but not the UV. Work submitted to this queue must be MPI parallelized.

The “special” queue is available for academic research that requires resources beyond those available through the large queues. Access to the special queues is turned on for a six month period of time, after having been granted access to this queue. In order to get access to the special queue, the user must first do a time complexity calculation to estimate the amount of CPU time and memory required by their job. They must then have their research adviser send a request for special queue access, including the resource requirements and a description of the work to **hpc@asc.edu** The special queue allows only one job to run at a time per user. This is done because the system has capacity to allow a few people to be running exceptionally large jobs, but can’t support allowing all users to run jobs of this magnitude.

A second mechanism for running exceptionally large jobs is to request dedicated machine time. Dedicated time means having the entire resources of the Alabama Supercomputer Center (or one of the clusters) reserved for the use of just one person. This is done for a once in a lifetime type of opportunity. For example, the last use of dedicated machine time was by a UAH professor who had their experiment flying on the space shuttle, and thus had to get data from the shuttle, use that data to run a simulation, and use the simulation results to call back up to the mission specialist on the shuttle to alter the experimental settings. Getting dedicated time requires months of prior planning, proposals and arrangements.

A third way of getting larger than normal resources is do a collaborative computing hardware purchase. A researcher with computing needs beyond what the existing facilities can accommodate can work with the Alabama Supercomputer Authority to contribute money towards the purchase of additional computing resources. There would then be queues that are only accessible to the members of that research group, which has additional CPUs reserved for their usage. For example, there were once queues named dixon-serial and dixon-parallel, which were for the use of researchers working for Dr. David Dixon at the University of Alabama. One advantage of doing this is that the staff at the Alabama Supercomputer Center can take care of system administration, hardware maintenance and software installation. Another advantage is that it is possible to run work that utilizes both the resources purchased by the faculty members and the existing resources to run calculations larger than could be run if the same grant money were used to simply put a new system on campus. This can result in getting access to a very large amount of computing resources, as a few hundred thousand dollars buys a large amount of computing power at today's prices.

The "commercial" queue is available to industry customers. These customers are paying by the dedicated hour for access to the computing resources. For a quote on purchasing processing time, contact the HPC staff at **hpc@asc.edu**

The "sysadm" queue is used by the staff at the Alabama Supercomputer Center. It is used for testing new queue settings, reproducing problems users are having, testing hardware, and other administrative functions.

Monitoring Jobs

The "**`squeue`**" command shows what jobs are running and pending in the queue system. The squeue command supports a wide range of display options. The default options are set with a global alias, as seen with the "**`whence squeue`**" command. Typing "**`squeue`**" gives an output like this

```
JOBID        NAME         USER          TIME ST         QOS
30008        VASP       asnabc   11-11:36:28  R       large
30286 Ka1942shSC       asnabc          8:32 CG        huge
30341 scr8SCRIPT       asnabc   12-12:56:40  R       large
36601 pmemdAMBER       asnabc          0:00 PD       large
```

The first two columns of this output shows a list of jobs. The job number is in the left hand column. The job number can be used to get more information about the job, kill the job, or request help from the ASC staff. This output also show a job status indicated by R or PD or CG. If the status is R the job is running. If the status is PD the job is waiting to run, or pending. The status CG indicates that the job is completing. squeue shows only your own jobs. QOS stands for "quality of service", which is simply what SLURM calls its queues.

Other squeue options can be seen with the command "**`man squeue`**". To use these options, it may be necessary to turn off the default behavior with the command "**`unalias squeue`**".

Tip!

The staff at the Alabama Supercomputer Center have written a number of scripts to show what your jobs are doing in more convenient formats. Try running "**`squeue3`**", "**`squeue4`**", or "**`squeue5`**" to see other information.

Jobs may be pending for a number of reasons. The job could be requesting more memory, CPUs, or software licenses than are presently available. It is sometimes the case that the requested resources aren't within the capabilities of the cluster. The easiest way to find out why a job isn't running is with the command;

```
checkjob_asn <job_number>
```

The checkjob_asn command analyzes various information and attempts to output a description of why the job isn't running. Additional information can be obtained with the commands "**checkjob_asn2**", "**checkjob_asn3**", "**checkjob_asn4**", and "**checkjob_asn5**".

Once a queued job completes, an additional file will be created in the directory with the job inputs. This is referred to as an error log file. The file name consists of the job name from the queue and the queue job number. This file contains information about how the job was submitted to the queue, stdout output from the job, and stderr output from the job. If a job fails to start or fails to run to completion, this file is one of the primary places to find out what is wrong. If you contact the ASC staff for help, they will want to see this file. Simply giving the technical staff the directory you are in and the job number is sufficient. The technical staff members can go into user directories, but do so only when asked for help.

The command "**jobinfo -j JOBNUMBER**" gives information about the job. If this command is called before the job is complete, it gives information about how the job was submitted to the queue. Calling the jobinfo command after the job is complete shows information about the jobs resource utilization, like this;

```
asndcy@dmc:tests> jobinfo -j 41697
###################################################################
#         Alabama Supercomputer Center - SLURM Epilog
# Your job ID was:                              41697
# Your job name was:                            parallel2comG09
# Your job started at:                          2016-07-22T10:19:38
# Your job ended at:                            2016-07-22T10:20:15
# Your elapsed time was:                        00:00:37
# Your username for this job was:               asndcy
# Your account for this job was:                users
# Your group for this job was:                  analyst
# Your partition was:                           dmc
# Your architecture for this job was:           nehalem
# Your max memory used was:                     128756K
# Your CPU time was:                            00:01:14
# Your number of processors used was:           2
# Your job state is:                            COMPLETED
# Your exit code is:                            0:0
# Your CPU utilization was:                     100%
# Your job ran on nodes:                        dmc112
# Your job submit QOS was:                      small
###################################################################
```

This information is useful for determining why the job died, and what resources it needs. For example, if the job requested 2gb of memory and it used 2234mb (>2gb) of memory, the queue system may have killed the job due to exceeding its memory allocation. The results from a job that ran correctly can be used to determine how much memory should be requested next time.

Tip!

Jobs that request more CPUs and memory can take longer to get into a run state. Thus it is to the users advantage to request a reasonable amount of memory, usually about 20% more than the job is expected to need. Choosing the optimal number of CPUs is discussed in the parallel processing section of this manual.

Deleting Queued Jobs

It is sometimes necessary to delete jobs from the queue. This can be because the job was submitted with the wrong inputs, isn't running correctly, or is stuck in a pending state due to invalid queue settings. Running or pending jobs can be deleted with the command

```
scancel <job_number>
```

Running Existing Applications Software

Applications software installed by the ASC staff have both an associated queue script and a README.md file with notes specific to running the software on the ASC systems. The README.md files are in subdirectories of /opt/asn/doc For example, the notes on how to use the Gaussian software are in the directory
/opt/asn/doc/gaussian In the example of the Gaussian software, a calculation using the input file water.com (which would have to be in the current directory) can be run with the following commands.

```
rung09 water.com
This runs Gaussian in the current directory via the queue system
Report problems and post questions to the HPC staff (hpc@asc.edu)

Choose a batch job queue:

Queue                      Wall Time       Mem  # Cores
------------------------ ---------- -------- --------
express                      4:00:00      16gb      1-4
small                       60:00:00       4gb      1-8
medium                     150:00:00      16gb     1-16
large                      360:00:00     120gb     1-64
huge                       360:00:00     256gb   64-128
class                       16:00:00      64gb     1-64
sysadm                     168:00:00       4tb   1-1000
```

```
Enter Queue Name (default <cr>: small) small

Enter number of processor cores (default <cr>: 2 ) 2

Enter Time Limit (default <cr>: 60:00:00 HH:MM:SS)

Enter memory limit (default <cr>: 1gb ) 2gb

Choose your job starting date and time (<cr> for now):
If not running right now, enter time and date as
[[CC]YY]MMDDhhmm[.ss]


Enter a name for your job (default: watercomG09)
parallel2

============================================================
=====         Summary of your Gaussian job            =====
============================================================
  The input file is: parallel2.com
  The output file is:  parallel2.com.log
  The time limit is 60:00:00 HH:MM:SS.
  The target directory is: /home/asndcy/calc/gaussian/tests
  The memory limit is: 2gb
  The number of CPUs is:  2
  The job will start running after: 2016-07-22T10:19:35
  Job Name: parallel2comG09
  Virtual queue: small
  QOS: -p dmc,uv --qos=small
  Constraints: --constraint=dmc|uv

The output file will not be placed in your directory,
until the job is complete.

Gaussian has default memory and disk use set for the small queue.
For other queues, set %mem and MaxDisk in your input file.

  Queue submit command:
sbatch -p dmc,uv --qos=small  -J parallel2comG09
--begin=2016-07-22T10:19:35 --requeue --mail-user=dyoung@asc.edu -o
parallel2comG09.o%A --mail-type=FAIL,END,TIME_LIMIT -t 60:00:00 -N
1-1 -n 2 --mem-per-cpu=1000mb --constraint=dmc|uv

Submitted batch job 41697
```

Press "Return" to take the default.

All of the queue scripts on the system use this same set of prompts for the queue, memory, etc. If it is not possible for a given program to run in parallel, the script will not ask for a number of CPUs. With many programs, it is necessary to construct the inputs appropriately for parallel execution, as well as requesting multiple CPUs from the queue script. It is possible to respond to all of the interactive prompts by pressing "Return" to take the defaults. Once the queue has been entered, the default prompts will reflect reasonable values for that queue.

Tip!

If you use the same queue settings every time, you can set preferences so it knows what you want and doesn't give the prompt. This is done by editing the **.asc_queue** file in your home directory. The comments in that file make it self explanatory, but additional information is in the file **/opt/asn/doc/slurm/preferences.txt**

Running User Written Software

When the user has written their own software, there will not be a queue script already available tailored to that software. There is however, a queue script named "run_script" which is configured to run any script that does not require arguments. run_script forces the job to use one or more processors, all on a single node. There is a "run_script_mpi" command for software capable of using processors on different nodes. Consider the example of a user that has compiled a program on the uv. For example, the program may be named foo and require an argument with the name of an input file such as bar.inp In this example, the program can't be submitted directly to run_script because it requires an argument. However, the user can create a script, say named myscript, that contains the following text.

```
#!/bin/sh
# script to run the foo program on uv
source /opt/asn/etc/asn-bash-profiles-special/modules.sh
module load intel/15.0.0
./foo bar.inp
```

The user can put this text in the myscript file using a text editor such as nano. The characters "`./`" in front of the program name indicate that the script expects to find the foo program in the same directory as the bar.inp and myscript files. The source and module lines load tell your program how to find the libraries that came with version 15.0.0 of the Intel Compilers. If you had to load modules to compile the software, you often must load the same modules to run that software. See the chapter on Using Modules for more details.

The myscript file must be made executable with a command like this

```
chmod +x myscript
```

This calculation can now be submitted to the queue system with the following command;

```
run_script myscript
```

The run_script program will give the same prompts as the other queue scripts with one additional prompt. The final prompt asks which cluster it should be allowed to run on. Since the program in this example was compiled on the uv, the user must enter "**`uv`**" at this prompt.

Now consider what could be done if the program foo has been compiled twice make both an uv executable and a dmc executable. In this case, the script must be able to identify whether it is running on the uv or the dmc and use the correct version of foo. A script like this is shown on the following page.

The following script must determine whether it is on the uv compute node or a dmc compute node. The "**`hostname`**" command returns the name of the node. The uv compute node is named uv1. The dmc compute nodes have names like dmc2 or dmc34. The output from hostname is piped into the command "**`tr -d '0123456789'`**". The tr command can translate one character into another, but the -d flag tells it to delete any of the specified characters. The small back ticks before the hostname command and after the tr command tell the script to execute those commands and place the result in the $myhost variable. This results in having the $myhost variable set to either "uv" or "dmc" with no numbers appended.

For more information on writing scripts like this, see the section of this manual on Shell Scripts.

```
#!/bin/sh
#  script to run foo on either the uv or DMC
source /opt/asn/etc/asn-bash-profiles-special/modules.sh
myhost=$(hostname | tr -d '0123456789')

# see if I'm on a uv node
if [ "$myhost" == "uv" ]
then
  module load intel/13.0
  foo_path=/home/asndcy/foo/bin_uv
fi

# see if I'm on a dmc node
if [ "$myhost" == "dmc" ]
then
  module load pgi/12.1
  foo_path=/home/asndcy/foo/bin_dmc
fi

# see if foo_path has been set
if [ $foo_path ]
then
  # run the program
  ${foo_path}/foo bar.inp
else
  echo "ERROR: this is not uv or dmc"
fi
```

This new script can be made executable, then submitted to the queue with run_script. This time, when run_script asks which cluster to use, the user can press "**Return**" to signify that it can run on either cluster.

Another function typically found in scripts submitted to the queue is copying the work over to be done on the /scratch drive. The user home directories are visible to all compute nodes, but these directories are on a lower performance file system. If the calculation does a large amount of accessing data on the disk drive, it will run faster on the high performance /scratch file system. Here is an example of a script that utilizes /scratch

```
#!/bin/sh
#
#  Sample script for submitting jobs to the queue system
#  This script shows how to run the calculation
#  in the /scratch directory
#
#  Replace the USER name in this script "asndcy"
#  with your own user name.
#  Replace the program name "mandy_gcc" with your own program name.
#  Replace the input file name "test1.in" with your own input name.
#
#  This script must be made executable like this
#    chmod +x my_script
#
#  Submit this script to the queue with a command like this
#    run_script my_script

#  put in my user name
USER=asndcy

# create a directory on /scratch
mkdir /scratch/$USER

#  copy the program and input files to scratch
cp mandy_gcc /scratch/$USER
cp test1.in /scratch/$USER

# run the program
cd /scratch/$USER
./mandy_gcc test1

# copy the results back to my home directory
cp test1.bmp /home/$USER
```

The application specific queue scripts and the run_script & run_script_mpi commands are specific to the Alabama Supercomputer Center. Everything that you can do with the queue system is accessible through these interfaces. However, some users may be more familiar using "sbatch" as is done at many other computing centers. The use of run_script and the application queue scripts is recommended. The use of sbatch directly is strongly discouraged, and not supported by the HPC technical staff. The run scripts wrap sbatch, and thus allow improvements and adaptations to changing needs on a regular basis. Thus users who want to use sbatch directly will find that they have to do so without documentation or technical support, and that they will have to change the options they give to sbatch as frequently as every six months.

User written MPI software should be launched with the **srun** command when run through the queues. Short tests (no more than 2 processors and 10 minutes) can be run on the login nodes with the **mpirun** command. See the MPI documentation accessible via the "`ascdocs`" command for additional information.

Queue System Fairness

Job queue systems are highly configurable. Even another facility using SLURM may have it configured to operate much differently from the SLURM installation at the Alabama Supercomputer Center. An organization that does weather prediction may have the queue system configured with reservations to ensure processors are available to run todays weather predictions at the same time every day. Many organizations have a system in place to charge users for the cost of processing time. Some organizations allocate computer resources based on the relative priority of various projects. The administration of the HPC systems at the Alabama Supercomputer Center is based on the following management principles;

- Select hardware to minimize unplanned outages, meet users' needs, and get a good amount of processing power for the cost.
- Provide some resources not available on most campuses (i.e. very large memory systems)
- Provide access to cutting edge technologies, as budget allows.
- Put significant effort into good documentation, in order to operate with a minimal support staff.
- Proactively innovate for ease of use.
- Proactively correct problems.
- ASA does not render judgment on the merit of client work.
- No in-house research is being done at the Alabama Supercomputer Center.
- Make the systems useful to as many clients (and disciplines) as practical.
- Assign resources based on policies that are applied equally to all users in the same category (research, classroom instruction, ASA's collaborators, or commercial).
- Encode policies in queue software, as much as possible. We do not manage based on "gentleman's agreements" or daily/weekly monitoring of users' behavior.
- Have a quick turnaround time on user help requests, with a necessarily longer turnaround time on software install requests or purchase requests.
- Find a balance of three, sometimes conflicting, goals;

1. Maximize allocation of resources (processors in a run state).
2. Attempt to give users submitting large job loads an approximately equal number of processor cores as other big users.
3. Set priorities to give shorter average queue wait times for users submitting a light job load than for users submitting a heavy job load.

In order to implement these ideals, a quantified fairness metric is used. Each month, the following ratio is computed for all users. This gives a range of values, typically arranged as a gaussian distribution with some users over- or under-advantaged and many clustered around the mean. Queue system scheduling configuration settings are then adjusted to minimize the standard deviation of this distribution.

$$\frac{\textbf{(monthly dedicated hours for person X)}}{\textbf{(average wait time in seconds for person X)}} = \textbf{user fairness for X}$$

We will note that there are some implications of this fairness metric. All users are considered equal. Thus job scheduling does not take into account the research group, project, or academic department. Jobs requesting many processors should and do wait in a pending state longer than single processor jobs, as the multi-processor jobs will get more processing time once they get into a run state. Some jobs may wait a shorter or longer than average amount of time, as fairness is defined on average but not on a per-job basis.

The queue system is configured to view the hardware as a large pool of available resources. As such, there are not X many processors assigned to the small queue, and Y many assigned to the large queue. There are, however, some processors reserved for the exclusive use of the class queue to ensure that homework assignments do not wait behind week-long research calculations.

10. Efficient System Utilization

Parallel computing is a very elegant idea. The naive, partially correct idea is that a computational task that takes 8 hours to complete using one CPU could be done in 4 hours on two CPUs and similarly scale to large numbers of CPUs. In actuality there are many algorithms or steps of algorithms that can only be executed serially (single CPU/thread execution). A second reason that this ideal case is not achieved, and seldom close is that the program must do additional work to coordinate the efforts of the multiple processors working on the calculation. In a few cases, each processor is actually recomputing values that would have been computed only once if it were run on a single CPU. Here are three examples to consider.

Example 1

Number of CPUs	1	2	4	8
Execution time (hrs)	8	4	2	1

Example 1 is called “linear speedup”. In reality there are very few algorithms that approach this ideal case. A couple of examples of algorithms that can come close to linear speedup are fractal geometry and testing encryption solutions. These algorithms work so well because each CPU can be given a different piece of data to work on, and will not need to access data or results from the other CPUs.

Example 2

Number of CPUs	1	2	4	8	16	32
Execution time (hrs)	8	4.5	3.2	3.5	5.3	12.7

Example 2 illustrates the most common case. Many algorithms can be parallelized to some extent. Thus adding CPUs improves performance up to some optimal number of CPUs. Going beyond the optimal number of CPUs often degrades performance, since the code is doing more work to parallelize the problem than actually solving the problem. The optimal number of CPUs depends on the algorithm being used, how heavily it has been parallelized, and the size of the problem being solved. Examples of problems that show this behavior are quantum chemistry and engineering simulations based on finite element algorithms.

Example 3

Number of CPUs	1	2	4	8
Execution time (hrs)	8	8.2	10	23

Example 3 illustrates an algorithm that parallelizes so poorly that attempts to do so harm the overall performance. Some of the highly correlated quantum chemistry algorithms show this behavior. This behavior can also be seen when the problem being simulated is so small that it really only took a small amount of time to run on a single CPU.

In theory the underlying algorithm determines how well a program can be parallelized. In practice the extent to which the software developers have completely rewritten the code for parallel execution is often the limiting factor. Even within a single program, there might be a large difference depending upon which input options were selected.

NOTE: At the time this user manual was last updated (July 2016) work was still under way to implement a Parallel Efficiency calculation in SLURM.

There are several ways to get a quantitative measurement of how well a program is parallelized. The software developers typically report a parallel efficiency as a percentage. Parallel efficiency is the ideal time divided by the wall clock time. The ideal time is the time to run the same calculation as an un-parallelized, single CPU application divided by the number of processes being used in the parallel test.

$$\textbf{Parallel Efficiency (\%)} = \frac{\textbf{(time to run single processor) * 100}}{\textbf{(number of processors) * (parallel wall clock time)}}$$

Thus if a calculation took 8 hours to run on a single processor and 2.5 hours to run on four processors, it's parallel efficiency would be 80%. As a general rule of thumb, if the parallel efficiency is below 75%, it is better to use fewer processors for each job and run more jobs at the same time. Parallel efficiency is one of the best measures of how well a program has been parallelized. However, parallel efficiency isn't convenient for the users of the application to work with because it requires doing both the single processor calculation and the multiple processor calculation.

It is often more convenient for users to look at the parallel utilization. The parallel utilization is the CPU time divided by the dedicated time. The CPU time reported at the end of the error log file is the sum total of the time that each processor was actually running, not sitting idle. The dedicated time is the wall clock time times the number of processors used.

$$\textbf{Parallel Utilization (\%)} = \frac{\textbf{(CPU time) * 100}}{\textbf{(number of processors) * (parallel wall clock time)}}$$

If the parallel utilization is 100% it means that all of the processors were working all of the time. If the parallel utilization is less than 100% it usually means that there were times during the calculation that just the master processor was running and the other processors were idle. This happens because there are sections of the code that are mathematically impossible to run in parallel, and because there are sections of the code that haven't yet been rewritten to run in parallel. For example, if four processors are requested, the master process runs 100% of the time, and each of the three slave processes run 5% of the time, the parallel utilization is 28.75%.

As a general rule of thumb, if the parallel utilization is below 75%, it is better to use fewer processors (CPUs) for each job and run more jobs at the same time. Jobs also tend to get through the queue system faster if the requested memory results in a memory utilization of 80% or higher.

Parallel utilization is not quite as good a measure of program performance as parallel efficiency. This is because parallel utilization fails to take into account that the CPU time is usually somewhat larger than the time to run the single processor job. This is because there is additional work involved in coordinating the efforts of multiple processors and passing data between those processors. There are a few software packages that show extremely large discrepancies between CPU time and single processor execution time because each processor is recomputing values that would have been computed only once in single processor execution.

Running Parallel Applications

Two things must be done to run a parallel application. First, the desired number of CPUs must be requested from the queue system when the job is submitted. Second, the application must be told how many CPUs to use, and sometimes requires a list of which CPUs. Some of the application software queue submit scripts take care of both of these.

Users writing their own codes will need to handle both tasks. The request to the queue system is made with options to the "**run_script**" command or the "**run_script_mpi**" command. The run_script command tells the queue system that all CPUs must be on the same node. The run_script_mpi command allows CPUs to be utilized on different nodes of the cluster. The way that the application is given a CPU count depends upon the mechanism that was used to parallelize it.

Writing Parallel Software

Parallel computers do not automatically run applications in parallel. Each piece of software must be written and compiled to run in parallel. This adds another whole dimension to programming projects. Thus, it is advisable to read books and take classes on parallel programming before embarking on a parallel programming project.

The following paragraphs discuss parallelization mechanism available on the computers at the Alabama Supercomputer Center. At present, MPI and OpenMP constitute about 95% of the parallelized software for high performance computing applications.

MPI (Message Passing Interface) is a message passing library standard. At the time this manual was written, MPI was the most widely used parallelization mechanism. MPI programs can run on both shared memory computers and distributed memory clusters. Most of the programs that scale up to the use of hundreds of CPUs or more are parallelized with MPI. The downside of using MPI is that converting a program from serial to parallel usually requires rewriting a major percentage of the code.

OpenMP is an API for shared memory multiprocessing. OpenMP parallelized programs can be run on shared memory computers, but not on distributed memory systems. OpenMP is often desirable because a program can be parallelized in sections with a fairly modest amount of effort. OpenMP generally works best for fine grained parallelization (at the loop level), when parallel programs will be executed on a small number of CPUs (2-16), and when all CPUs will be frequently accessing shared data.

P-Threads is an API for handling Linux threads. The p-threads library is used by the Unix operating system for managing multiple execution threads. It can also be used by an application for writing a shared memory parallel program. Parallelization with p-threads requires a large amount of low-level programming. Thus pthreads are generally only used when there is a technical reason for needing this low-level control over the shared memory threading.

Java threads The Java virtual machine has the ability to execute multi-threaded Java programs. The support for manipulating threads is included in the Java language and is augmented by open source libraries, such as the “spin” library.

PVM (Parallel Virtual Machine) PVM is the predecessor of MPI. Today most PVM codes have been converted to MPI.

Coarray Fortran (CAF) is a parallel programming extension of the Fortran syntax that allows a programmer to view a single shared partitioned address space.

High Performance Fortran is an extension of Fortran 90, which allows the programmer to put in directives for parallel execution.

Parallel Math Libraries Some math libraries have versions that have been compiled with support for parallel execution of the functions in that library.

Parallelizing Compilers There are compilers that attempt to parallelize a serial program automatically. At present this technology is still in its infancy, so other parallelization methods almost always give a better parallelization.

Note: Parallelizing compilers are considered the holy grail of parallel programming. However, this infancy stage technology does not yet perform as well as other parallelization mechanisms.

Global Arrays The Global Arrays toolkit allows software to be written as though it were on a shared memory computer, even though it may actually be running on a distributed memory system. Global Arrays are often used in conjunction with other tools, such as MPI or TCGMSG. The Global Arrays tools are implemented on top of ARMCI (Aggregate Remote Memory Copy Interface). The TCGMSG message passing library is also distributed with the Global Arrays toolkit.

Estimating CPU Time and Memory Needs

The HPC systems at the Alabama Supercomputer Center have multiple queues, which allow jobs to run for various lengths of time and use different amounts of memory. Most users of desktop computers aren't used to thinking about how long it will take the computer to do the work, or how much memory is required. This isn't often a problem on desktop computers because the computer has been designed with capacity to run typical business applications, and uses a small percentage of that capacity most of the time, except when it bogs down running a video game.

Processing time, memory and disk space become an issue with the type of applications typically run on supercomputers. This is because these applications can take days or weeks to run using hundreds of gigabytes of memory and terabytes of disk space. Furthermore, there are hundreds of people running work on a supercomputer. If resources like memory weren't managed, one person's program would be killed when another person's program used all of the memory. This doesn't happen because the queue system assigns memory and processors to each job, thus guaranteeing access to the necessary resources. In order to make this system work, the user must specify how much memory their job needs. If that estimate is unreasonably high, it will result in waiting much longer for the job to get access to those resources, and fewer jobs can be run at once. Thus it is to the advantage of the user to know how to give a reasonable estimate of the resource needs for their calculations.

The memory and CPU time needs for many calculations are often not proportional to the size of the input. Consider the example of a software package that computes properties of molecules. This program might need to compute the distance between each pair of atoms. Thus if there are 20 atoms, the distances between each pair could be stored in a 20x20 element array, which would have 400 elements in the array. Thus the amount of memory needed by this step of the program would be proportional to the square of the number of atoms. Likewise the amount of CPU time required to compute the distances between each pair of atoms would be proportional to the square of the number of atoms. The size of the problem, in this example the number of atoms, is typically called "**n**". The time complexity or memory complexity is then represented as **O(n^2)**, a format known as Big O Notation. The rest of this section of the book uses the term "time complexity" but the same type of analysis could be used for estimating memory or disk space needs.

Computer programs can have many different time complexities, and each function within a given computer program can be assigned it's own time complexity. Some functions require the same amount of time, no matter how much data is involved, thus giving a constant time complexity or **O(1)**. Some things scale linearly with the size of the problem, denoted as **O(n)**. Some processes are slightly better than linear, such as **O(n log n)**. Many algorithms scale as some power of the size of the problem, such as **O(n^3)** or **O(n^8)**. There are a few algorithms that scale very badly, such as factorially scaling problems with a time complexity of **O(n!)**.

This slightly theoretical discussion of time complexity notation shows that different programs and functions within programs scale differently. We must now find a way to practically apply this knowledge to making an estimate of how long it will take to run a given program. The first step is to look at some calculations that have already been completed. The jobinfo command provides and output that looks like the one shown in Figure 9.1.

If similar size calculations have already been run, simply runing jobinfo with the job number from those calculations may be all that is needed to find out how much CPU time and and memory to expect a calculation to require. Time complexity calculations refer to CPU time, but wall clock time can be estimated as CPU time divided by the number of processors being used, or a bit more than that if the algorithm doesn't parallelize efficiently.

Many researchers will do several test calculations leading up to a large calculation. They will do a small calculation to estimate the resource requirements of a medium size calculation. Then use those results to estimate the requirements of the large calculation. This is done using the time complexity of the algorithm. For example, if the algorithm has an $O(n^3)$ complexity and the time for a smaller calculation (T1) is known along with it's size (n1), then the time for the larger calculation (T2) can be computed from it's size (n2) as follows.

$$\mathbf{T2 = T1 * (n2 / n1)^3}$$

In these examples, the size might be the number of atoms, number of elements in a finite element calculation, number of basis functions, or some other aspect of the calculation. Often a moderate size calculation is better than a small calculation for doing complexity estimates. This is because the smallest calculations have resource utilization dictated more by overhead than by the critical portion of the code.

If you don't know the time complexity of a software algorithm, there is a formula for determining it. There is an example of how to do this in the file
/opt/asn/doc/gaussian/Estimating_CPU_time.pdf

If you logged in via X-Windows, this can be viewed with the command
"`firefox /opt/asn/doc/gaussian/Estimating_CPU_time.pdf`"
Otherwise, download it to your local computer to view the file.

A very few software packages have a "check" mode which performs an estimation of the resource utilization, but doesn't actually run the calculation. These should be used when available, but they are unfortunately rather rare.

There is an experimental piece of software written at the Alabama Supercomputer Center called "**`swami`**". The name swami is a reference to an old Johnny Carson skit where he played a mystical swami that predicted the answers to questions before reading the question. Swami predicts the CPU time and memory required for Gaussian and NAMD calculations. Swami uses an artificial intelligence learning algorithm. Results of completed calculations can be loaded into Swami with the "**`swami-learn`**" program. As more results are entered, the predictions made by swami become more accurate. More information about swami can be found in the directory **/opt/asn/doc/swami**

11. Using Modules

The term "modules" is used in a number of different ways in the computer science field. In this context "modules" refer to LMOD, which is a way of controlling your account environment settings. Modules are used to configure your account to access some of the software packages installed on the supercomputers.

Tip!

If it is necessary to use a module for one of the software packages, the appropriate module command will be described in the file /opt/asn/doc/PROGRAM/README.md (i.e. /opt/asn/doc/gaussian/README.md)

The environment necessary to run modules is loaded automatically when you login. However, this environment is not automatically visible to scripts that run through the queue system. Thus if you are compiling your own software, you may have to include the following in your run scripts, in order to use modules.

```
source /opt/asn/etc/asn-bash-profiles-special/modules.sh
```

In many cases, modules are working behind the scenes but you are not aware of them. For example, jobs using the Gaussian software are submitted to the queue with the command "**rung09 <filename>**". The rung09 script calls the appropriate module to load the Gaussian environment settings. These environment settings tell the operating system where to find the Gaussian executables, the associated libraries, and other data files. Modules can also remove conflicting settings, thus allowing multiple version of the same software package to be installed without problems from getting the wrong one accidentally.

The list of modules available on the system can be shown with the command

```
    module avail
```

Note that many software package have more than one version that can be accessed through loading the appropriate module. One of those versions may be labeled as the default version.

Information about a given module can be displayed with the command

```
    module help <module_name>
```

The module name can be of the form "**`name`**" or "**`name/version`**". Specifying a name without the version will give information for the default version. For example, try typing "**`module help gaussian`**".

The list of modules currently loaded in your session can be shown with the command

```
module list
```

Unless you have configured your account to automatically load modules on login, this command will probably indicate that no modules are currently loaded.

A module can be loaded with a command like this

```
module load <module_name>
```

For example, typing "**`module load gaussian`**" will load a default version of Gaussian, as shown by typing "**`module list`**". If your account has not been given permission to access the Gaussian program, attempting to load it will give an error. To load a different version of Gaussian, you could type "**`module load gaussian/g09a02`**". Module load commands can be put in your .bashrc.local.uv & .bashrc.local.dmc & .bashrc.local files in order to automatically load modules when you log in.

Loading a module may result in loading several other modules automatically. This may be done if a piece of software needs access to the libraries associated with a particular compiler, math library, or MPI version.

TIP: Specify the version when loading modules inside of scripts. The default version is the most recent available, which changes every time a new version is installed. This can cause your scripts to break if the version is not specified.

WARNING: It is possible to bypass the module system by setting the paths in your script. However, loading the module also logs the fact that someone is using that software. If the software use is not logged, software is flagged as unused and removed from the system. As such, you should always load the module to access software.

Modules can be unloaded with the command

```
module unload <module_name>
```

Modules can fix problems with multiple versions of the same software. However, they can also create problems. Consider what happens when you load the module for software package A. Loading the module for that software loads all of the modules that it needs, such as version 10 of the Intel compilers. Now what happens when you load the module for software package B, which needs version 12 of the Intel compilers? Version 10 is unloaded and replaced with version 12. In this example, loading software B may break software A.

WARNING: Loading multiple modules can sometimes break software packages. Thus it is best to load only the modules for the work you are doing right now.

There is a way to unload all of the modules currently loaded with a single command. This is done with the command.

```
module purge
```

If you think loading too many modules may have broken something, type "module purge" then load only the modules that are absolute necessary for what you are doing.

TIP: A shortcut for the module command is "**`ml`**". Typing "**`ml`**" is equivalent to "**`module list`**". Typing "**`ml av`**" is equivalent to "**`module avail`**". Typing "**`ml NAME`**" loads the named module.

12. Account Configuration

Casual users of Linux and the Alabama Supercomputer Center can skip this chapter and still get all of their work done. Individuals that expect to spend a large amount of their time on Linux will find a number of items in this chapter to make their work more efficient. An understanding of account configuration is also useful to understand why things aren't working correctly, which is important if you want to administer Linux systems.

WARNING: The parts of Linux described in this chapter fill important roles in the operating system, and can be valuable tools. However, changing them incorrectly can cause many things to break, at least for your own account. It's best to tread carefully, and ask questions if you feel you don't understand how to use them correctly.

The following sections discuss various ways to customize the behavior of your Linux account. Many of these changes take effect the next time you login on the system.

Environment variables

Environment variables are values that are visible to all of the software running on a computer. These are often used to tell the operating system how it should behave, and to tell software packages which directories to find important files in.

Login to your Linux account and type the command "**`env`**". This shows a list of all the environment variables that your account currently sees. Some are found in all Linux systems, some are specific to the bash shell, and some are specific to a given computer program. Table 10.1 list a selection of the environment variables that are amongst the more important to account configuration.

Some environment variables are redundant. For example, the variables LIBRARY_PATH, LD_LIBRARY_PATH, LIBPATH, and SHLIB_PATH all tell the operating system where to look for static libraries and shared object files (.so files are dynamic linked libraries, equivalent to .dll files in Windows). There are multiple environment variables doing the same job because some are used by different shells, or linux distributions. Since different programs look at different ones, a redundant configuration keeps all of the programs finding the paths to the libraries.

Table 12.1 Useful environment variables

Variable	What it Does
CLASSPATH	Java programs use this to find their libraries
DISPLAY	tells X-Windows where to display graphics
HISTSIZE	number of commands displayed by the "history" command
HOME	your home directory
HOST, HOSTNAME	the computer (or cluster node) you are logged in on
HOSTTYPE, CPU	the computer processor architecture
INCLUDEDIR, INCLUDE	paths to header files
INFOPATH, INFODIR	paths to data displayed by the "info" command
LD_LIBRARY_PATH, LIBRARY_PATH, LIBPATH, SHLIB_PATH	paths to static linked and dynamic linked libraries
LS_COLORS	allows customizing colors used by the "ls" command
LS_OPTIONS	default options for the "ls" command
MANPATH	paths to data for the "man" command
PATH	paths to find executable files
PS1	changes the bash command prompt
PWD	the current directory
TERM	terminal display settings
USER, LOGNAME	your user name
_	(underscore) the command currently being executed

The value of an environment variable can be displayed with the "echo" command. For example, the PATH environment variable tells the operating system where to look for programs to run. You can see where the run_script program resides by typing "**`which run_script`**".

To see all of the directories that the operating system is looking in to find run_script, type the following;

```
echo $PATH
```

You can set new environment variables, or add data to existing environment variables. For example, you may want to create some of your own programs and scripts. In order for those programs to be found when you run them, you can put them in a new subdirectory, typically named /home/MYNAME/bin . In order to tell the operating system to look for your programs in this directory, you would add a line like this into your .bashrc.local file.

```
export PATH="$PATH:/home/MYNAME/bin"
```

Note that by including $PATH: in the new value of PATH, you are appending a new directory onto the existing path list. The directory names are separated by colons. If you left out this $PATH: part of the line you would be taking away all of the paths to the operating system commands, thus breaking most of the functionality of your account.

Environment variables can be used in shell scripts. For example, if you wanted a shell script to create a directory with your user name, you could use a line like this "**`mkdir /scratch/$USER`**". Environment variables are accessible within most compiled computer languages also, although the mechanism for accessing them varies from one language to the next.

Hidden files

Any file or directory that has a name starting with a period will be hidden by default when you use the "**`ls`**" command. To see the hidden files in your home directory type "**`ls -a`**".

Hidden files and directories are typically used to store configuration settings that control how your account behaves. These settings tell your account how to find operating system commands, access various applications software, and set default behavior of programs. A list of some of the common hidden files and directories is shown in Table 10.2 . Many other hidden files may appear in your account only if you are using certain software packages.

Table 12.2 Useful hidden files & directories

File or Directory	What it Does
.alias	location for alias commands
.bashrc	primary account configuration file for most Linux systems. WARNING: Do not modify .bashrc on the ASC systems
.asc_queue	preferences for the queue system
.bashrc.local	settings that affect both uv and dmc
.bashrc.local.uv	settings that affect the uv only
.bashrc.local.dmc	settings that affect the dmc only
.bash_logout	commands run at logout, less often used
.cshrc, .login, .profile	primary account configuration file for most Linux systems.
.flexlmrc	license server configuration
.forward	holds email address where notices are sent
.ssh	directory with encrypted ssh keys & configuration
.vimrc	configuration for the vi editor

WARNING: Files in the .ssh directory can be configured to allow you to move between systems without typing a password. This is convenient, but it can also be dangerous. If you do this then a criminal gets into one of your accounts, they can instantly get into all of your other accounts as well.

The files .bashrc.local .bashrc.local.uv and .bashrc.local.dmc are the ones that must most frequently be modified in order to configure your account to run a given software package, or to change the default behavior. The .alias file is used for setting aliases, described later in this chapter. The .forward file contains an email address where any notifications generated on the system will be sent. On SUSE Linux systems, the .profile file is edited only in rare cases, such as setting the LS_COLORS environment variable. Modifications needed for a specific program will be listed in the file **/opt/asn/doc/PROGRAM/README.md**

The order in which commands are put in these files is sometimes important. For example, the PATH environment variable is searched in the order that directories appear, which determines what executable to run if there are two with the same name.

It is also advised that you only load modules for software you are using right now, since modules can conflict thus breaking the software. This will remove many of the settings that may come before it in the .bashrc.local files.

The source and module commands

Sometimes the account configuration needed by a program is more complex than is convenient to ask users to type into their .bashrc.local file. In this case a whole list of settings can be loaded with a single command using the "source" or "module" commands.

Reminder

> Either the "source" or "module" command is needed to configure your account to use many software packages interactively. These are often not needed to run programs through the job queue system, as the queue scripts provided on the system handle this for you.

The "source" command is available in all distributions of Linux (in some shells it is a period instead of the word "source"). The "module" command is part of the LMOD package, which is an add-on to the operating system described in the previous chapter. The Alabama Supercomputer Center is slowly shifting software packages from using the "source" command to using the "module" command.

The "source" or "module" command syntax needed to run a given software package will be documented in the file **/opt/asn/doc/PROGRAM/README.md** (i.e. /opt/asn/doc/gaussian/README.md).

The "source" command can be put in your .bashrc.local or .bashrc.local.dmc or .bashrc.local.uv file. Here are examples of typical source and module load commands. As shown here, comments (beginning with #) can be added to remind yourself when each should be used.

```
# The following sets the X-Windows $DISPLAY variable.
# This is needed for some X-Windows clients
# Do not source this for cygwin with -Y flag in ssh
source /opt/asn/etc/asn-bash-profiles-special/display.sh

# this is needed to run Gaussview
module load gaussian/g09b01
```

The Command Prompt

The default bash shell command prompt includes the full path to the current directory. This can be inconvenient when working in a directory deep within the directory tree and the command prompt is taking up most of the screen space, like this;

```
asntest@dmc:/opt/asn/doc/gaussian/sample_inputs_g09_A01>
```

This can be changed by setting the PS1 environment variable in the .bashrc.local file. For example, try using the setting;

```
    export PS1='\h:\W> '
```

This will result in seeing a command prompt like this.

```
dmc:sample_inputs_g09_A01>
```

Command prompts can have the machine name, path, time, date and other information. Details of how to set all of these options can be seen on the bash manual page that you get by typing "**`man bash`**" and on websites such as http://www.linuxselfhelp.com/howtos/Bash-Prompt/Bash-Prompt-HOWTO-2.html

Creating an alias

Aliases are keystroke short cuts. For example, a user who organizes calculations into separate directories might want to see only a list of subdirectories of the current director. He can this with a command like this.

```
    ls -l | grep ^d
```

While this works, it may be more than he wants to type many times each day. He can make a faster way of doing this by putting the following line in his .alias file.

```
    alias ls2="ls -l | grep ^d"
```

After setting this, he must log off and log back in for the command to take effect. From then on, he can simply type "**`ls2`**" to get a listing of subdirectories. An alias can also be used to ensure that a given command line option is always added to a command.

Tip!

If you have problems with an alias, try typing the command that the alias runs. List this command when contacting technical support.

You can temporarily remove an alias with the command “**`unalias name`**”.

Tips for Effectively Using the Supercomputers

The following are some suggestions for getting the most out of the uv and dmc.

Tip!

- Set aliases for frequently typed, long commands.
- Learn to use multiple Linux commands like head, tail, cat, and grep in a single command line with data piped from one to the next.
- Learn to write scripts to automate tasks.
- Create a bin directory for your scripts, and add it to the PATH variable.
- Organize your files into directories so you can find them easily.
- Have a plan for what files to keep, which you get rid of, where you store them on campus, and when they get deleted.
- Run tests to find out how many processors your software will use efficiently. Too many will hurt your productivity more than using fewer than optimal.
- Learn to utilize the queue system effectively.
- Look at the README.md file for your program in /opt/asn/doc/PROGRAM

13. Compiling Software

The majority of the applications on supercomputers are written in compiled languages such as C++, C, and Fortran. Compiled programs tend to run faster and use less memory than interpretive languages, such as Perl, R, or Matlab. Languages that compile to byte codes, such as Java or Python, are intermediate in performance.

The Fortran programming language was originally created in the 1950s as a language for mathematical applications. Major revisions to the Fortran specification were released in 1966, 1977, 1990, 1995, 2003, and 2008. These revisions have added support for newer programming conventions such as pointers. Fortran 2003 added object oriented programming constructs to Fortran. Fortran 2008 added concurrent programming. However, Fortran remains a procedural, line oriented language, making it archaic by the standards of the computer science field. Fortran is a very small percentage of all software development in the world. In spite of this, Fortran code remains rather common in the high performance computing field due to the number of software packages that utilize code that was written in Fortran decades ago.

The C language is the programming language of choice for writing operating systems and hardware device drivers. It is a procedural language that is powerful enough to do things that could only be done in processor specific assembly language before the invention of C. However, C is rather unforgiving as a language for applications development because it is weakly typed and contains no intrinsic error checking. As such, C is rarely used for writing mathematical simulation software.

The C++ language is an object-oriented, strongly typed derivative of C. With a few notable exceptions, C programs will compile as C++ code. However, the majority of C++ code is object oriented code, which would not compile as C code. C++ has been used for a large percentage of applications software written in the past 20 years.

Prior to the invention of the Java language, most graphic interface based programs could only be used under one operating system. Java changed that situation by creating a programming language that could be used to write graphic interfaces that would run on many different platforms without any change to the code, or even being recompiled. Thus Java is very popular for graphic interface development. However, Java compiles to byte codes which results in it not executing as quickly as natively compiled programs for complex mathematical operations. Many software packages use a Java graphic interface on top of mathematical executables written in C, C++ or Fortran.

There are several different compiler suites available on the ASC systems. These include the following.

- GNU C, C++, and Fortran
- Intel C, C++, and Fortran
- Portland Group C, C++, and Fortran
- Special purpose languages such as CUDA, Objective Caml, Unified Parallel C, and LISP

Selecting the best compiler is not necessarily a trivial task. One rule of thumb is to follow the software makers recommendations, if a recommendation is given. Many public domain software packages are developed using GNU compilers, and in rare cases may only compile with the GNU compilers. Some legacy codes, such as those originally developed for Vax Fortran, compile best with the Portland Group compilers. Those exceptions aside, the Intel compilers most frequently give the best optimized executables, which thus run the fastest.

Additional information about the compilers is available by using the "**`man <command>`**" command. There are also documentation and README files in the directories **/opt/asn/doc/compilers_uv** and **/opt/asn/doc/compilers_dmc**

In some cases, ASC keeps old versions of compilers. A users account can be configured to use these older versions by adding commands to the .bashrc.local and .bashrc.local.uv and .bashrc.local.dmc files. Settings in .bashrc.local affect both clusters. Those options are documented in the README.md files in the directories listed in the previous paragraph.

The compile C, C++, and Fortran commands are;

gcc	GNU C/C++
g++	GNU C++, loads C++ libs
gfortran	GNU Fortran 95 (includes Fortran 77 & 90)
pgcc	Portland Group C compiler
pgCC	Portland Group C++ compiler
pgf77	Portland Group Fortran 77
pgf90	Portland Group Fortran 90
pgf95	Portland Group Fortran 95
pghpf	Portland Group High Performance Fortran
icc	Intel C
icpc	Intel C++
ifort	Intel Fortran 66, 77, 90, 95

Reminder

Compile commands, except a very old version of gcc, are not available until you load the appropriate module.

A Fortran Program Example

The basic sequence for compiling a Fortran program is as follows:

```
ifort program.f -o program
```

The source code must be in a file named with the extensions .f or .f90

The program can be executed interactively by simply typing the following.

```
./program
```

If the executable name isn't specified on the compile line, it will be named **a.out**

EXAMPLE

The following is an example of a console session in which a Fortran program is compiled and executed.

```
uv:~/hello $ ls
hello.f
uv:~/hello $ ifort hello.f
uv:~/hello $ ls -l
total 1028
-rwxr-xr-x    1 asndcy analyst 800106 May 19 11:58 a.out
-rw-r--r--    1 asndcy analyst     65 May 18 13:49 hello.f
uv:~/hello $ ./a.out
 Hello
uv:~/hello $
```

A C Program Example

The GNU C compiler is called with the gcc command:

```
gcc source.c -o myprogram
```

The command above compiles the source code in the file "source.c" and creates an executable named "myprogram". If the executable file name is not specified, it will be named a.out

The following line executes the program

```
./myprogram
```

EXAMPLE

The following is an example of a console session in which a C program is displayed, compiled and executed.

```
asndcy@dmc:~/hello> ls
hello.c
asndcy@dmc:~/hello> ls -l
total 4
-rw-r--r--  1 asndcy analyst    81 May 19 12:06 hello.c
asndcy@dmc:~/hello> cat hello.c
#include <stdio.h>
main()
{
  printf ("Hello World .....from C\n");
  return;
}

asndcy@dmc:~/hello> gcc hello.c
asndcy@dmc:~/hello> ls -l
total 20
-rwxr-xr-x  1 asndcy analyst 13351 May 19 12:07 a.out
-rw-r--r--  1 asndcy analyst    81 May 19 12:06 hello.c
asndcy@dmc:~/hello> ./a.out
Hello World .....from C
asndcy@dmc:~/hello>
```

Compilation and Runtime Errors

An important part of the programmers skill set is recognizing common compile and runtime errors. If understood, these tell you what is wrong and suggest how to fix the problem. The following are some common errors and what to do about them.

The first error

As a general rule of computer science, the first error generated is the most important one, at least most of the time. It is quite common that thirty error messages may be generated from a single problem. The first one is the real problem and the subsequent ones are side effects of that problem, called cascading errors.

Cannot execute binary file

This usually means that you are trying to run a program compiled for an architecture of processor that is not compatible with the processor in this computer or compute node.

The line number

Compilers often tell you exactly what line of the program has a problem. There is a quirk with some compilers and some types of errors in which the problem may actually be on the line of code before the line specified by the error message. Look at both.

No such file or directory

This error can be generated when a file can't be found by the compiler. There may be several ways to fix this. Many compilers recognize the environment variables LD_LIBRARY_DIR (sometimes replaced by LIBRARY_PATH, SHLIB_PATH, or LIBPATH) and INCLUDEDIR. Sometimes you need to specify a path in a Makefile or compile line. Some software packages have installers that recognize environment variables. Sometimes paths are specified on the command line to a configure program. Sometimes an absolute path is specified in the code, such as #include commands with double quotes, in which case it may be necessary to copy the necessary files into the source tree. With all of these possibilities, your best bet is to read the installation documentation. If that doesn't help, start trying each of the possibilities.

Not declared in this scope

This error means that the code is trying to call a function that has not been defined. Functions are usually defined in header files with an extension of .h The header file is then referenced in the code with a command like `#include <myheader.h>` and the .h may be optional in the command. Most commonly you see this error if the include command is in the code, but the compiler can't find that file. A flag like this can be added to the compile line `-I/path/to/include` or this may be accomplished by giving some argument to a configure script. This error can also occur if you are including the header file from one version of the library then linking against a different version.

Permission denied or Failed to execute

This typically happens when you wrote a script, but did not set the execute permissions on the file with chmod command. Another reason for getting this error can be writing a script file on a Microsoft Windows computer, which uses different end of line characters in text files. A text file can be converted from windows format to linux format with the command `dos2unix filename`

Segmentation fault or Segmentation violation

These are generally runtime errors, generally called segmentation errors. Segmentation faults are related to memory. This can happen if you didn't request enough memory from the queue system. It can also occur if the computer itself does not have enough memory installed. Another cause can be if you created an array 100 elements long, then tried to access element 101.

Syntax error

This is an error in correctly specifying the command or punctuation in a programming language.

Unable to locate a modulefile

The error occurs if you have a `module load module_name` command and no such module is found. This most often occurs if the job is running on the UV and you are trying to load a module that is only available on the DMC.

Undefined references

An undefined reference means that the compiler can't find a function that your code is calling. This is usually because a required library isn't being linked in on the compile line. Sometimes the order in which libraries are specified is important, so the seemingly missing library has to put at the end of the compile line.

If you add the necessary library (with a **-lname** flag) and still can't find it, you may have to specify the path for the library like this **-L/path/to/lib -lname**

Warnings

99.9% of the time warnings can be ignored. That one in a thousand where a warning message identified the root problem will be one where you tried many other options before looking closely at the warning messages.

Static Versus Dynamic Linking

Most programs use libraries. These are files that contain functions that can be called from your program. For example, trigonometric functions are often not part of the core programming language, thus requiring you to compile with a **`-lm`** flag to load the math library if your program must call a cosine function. There are two ways in which your program can access these libraries of functions, called static and dynamic linking.

Static linking means that all of the necessary functions are physically integrated into the executable file. This means that the executable will always be able to find those functions and will always use the same versions of those functions. Static linking is a good idea if robustness is important, particularly if you want to use an executable on a computer other than the one where it was compiled. Static linked programs also execute slightly faster. The disadvantage of static linking is that the executable files take up more disk space.

Dynamic linking means that the program knows where to find those functions in library files when it needs them to execute. This means that the executable files will be smaller. As such, your program can break if the library files are changed, or erased, or if the environment variables specifying their location are changed. Your program will also execute slightly slower. You can check what libraries are dynamic linked executable will use with the command

```
ldd executable_name
```

Most compilers use dynamic linking by default, and have flags to specify static linking. The library files must also be compiled for dynamic or static linking. Library files have names ending with .a are static linked. Library files having names ending with .so are dynamic linked.

Optimization

This section provides information about techniques you can use to optimize Fortran, C, or C++ code on the ASC supercomputers.

Optimization is the process of changing a program or the environment in which it runs to improve its performance. Performance gains generally fall into one of two categories of measured time:

- User CPU time. Time accumulated by a user process when it is attached to a CPU and executing. When running on a single CPU, CPU time is a fraction of elapsed time. When multitasked, CPU time is a multiple of elapsed time.

- Elapsed (wall-clock) time. The amount of time that passes between the start and termination of a user process. Elapsed time includes the following:

  - User CPU time
  - Linux system CPU time
  - I/O wait time
  - Sleep or idle time

Optimization begins with code that has been debugged and is running correctly on the system. Before beginning optimization work, some sample inputs should be prepared along with validated correct outputs. This allows testing to verify the work done did not change the results given by the software.

Manual parallelization with libraries like MPI, PVM, ARMCI and LINDA is very labor intensive. It is advisable to try all other optimization options and analyze the size of future runs carefully before embarking on this path. Furthermore, some codes will parallelize well while others get only marginal improvement. For an algorithm that does parallelize well, this can allow you to decrease the elapsed execution time in proportion to the number of CPUs used.

The first step is to compile using the best compiler options and data types for your program. It is recommended that you read the compiler documentation, as some flags may improve execution speed at the expense of losing some mathematical accuracy. The following paragraphs give recommendations for the compilers available at ASC.

ALL COMPILERS: The first optimization step is to try -O1 -O2 and -O3 compiler flags. -O3 is usually the best, but in rare cases a code won't compile correctly with -O3. GNU compilers use -O in place of -O1.

INTEL COMPILERS: Other flags that can improve optimization are; -Bstatic, -fast, -fnsplit, -ansi-alias, -qopt-prefetch, -unroll-aggressive, -ip, -ipo, -prof_use, -tpp2.

PORTLAND COMPILERS: Other flags that can improve optimization are; -Bstatic, -fast, -O4, -Mipa=fast,inline, Mcache_align, -Mdalign, -Mllalign, -Mfunc32, -Munroll, -Minline,
-Mscalarsse.

GNU COMPILERS: Other flags that can improve optimization are; -Og, -Ofast, -fcaller-saves,-fcse-follow-jumps, -fcse-skip-blocks, -fdelayed-branch, -felide-constructors, -fexpensive-optimizations, -ffast-math, -ffloat-store, -fforce-addr, -fforce-mem, -fmemoize-lookups, -fno-defer-pop, -fno-function-cse, -fno-peephole, -fomit-frame-pointer, -frerun-cse-after-loop, -fschedule-insns, -fschedule-insns2, -fstrength-reduce, -fthread-jumps, -static, -funroll-all-loops, -funroll-loops. The -fexpensive-optimizations can be a useful catch-all to try first. Beware of -ffast-math if your code is sensitive to numerical precision.

More information on compiler flags can be found in the man pages for each compiler command, and in files in the directories **/opt/asn/doc/compilers_uv** and **/opt/asn/doc/compilers_dmc** on both supercomputers. Up to a 4X speedup can be achieved by using the compiler flags to utilize vector instructions.

One option is to use optimization flags that will utilize the instructions on the more recently made processors. This can give improved execution time at the expense of only being able to run the executable on certain nodes in the cluster. Sometimes limiting where jobs can run will increase queue wait times to the point that this does more harm than good. A more sophisticated approach is to compile several versions optimized for various processors, then write a queue script to select which one to run based on the processor architecture where the job is assigned.

Tip!

Before continuing any further with optimization, it is highly advisable to prepare a set of test calculations and correct results to compare them to. Although it is usually possible to make a program get the same answer much faster, there is definitely a risk that the changes you make might break the program, causing it to give incorrect results. Throughout the course of optimization work, save the last copy of the source code that ran correctly, and rerun the test set frequently.

Optimizing code is an iterative process requiring the following steps.

1. evaluate the code
2. determine possible areas where optimization techniques can be applied
3. apply the techniques
4. check code performance
5. is code sufficiently optimized?
    - if not, return to step 1
    - if yes, the code is optimized for single CPU performance

The evaluation step is most often done using a program called a profiler. In order to use a profiler, first recompile the program with a compiler flag that turns on profiling. Then run a test calculation, which generates an extra output file with profiling data. Then run the profiler, which analyzes the data and outputs information about how the program ran.

Usually, there are two crucial pieces of data in the profiler output; how much time was spent executing each subroutine, and how many times each subroutine is called. The most productive optimization is usually making the functions that spend the most time executing run more efficiently. The second most productive optimization is usually cutting down on the number of calls to a function that is called millions of times.

On the following page is an example of how to compile a C++ program with the g++ compiler, run a calculation, and profile the code with the gprof profiler.

```
uv:~/source/mandy $ g++ mandy.cc -O3 -pg -o mandy
uv:~/source/mandy $ mandy test5_big
uv:~/source/mandy $ gprof mandy
Flat profile:

Each sample counts as 0.000976562 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00    263.00   263.00        1   263.00   263.00  mandelbrot(_IO_FILE*)
  0.00    263.00     0.00        1     0.00     0.00  _GLOBAL__I_x_initial
  0.00    263.00     0.00        1     0.00     0.00
write_header(_IO_FILE*)
  0.00    263.00     0.00        1     0.00     0.00
__static_initialization_and_destruction_0(int, int)

 %         the percentage of the total running time of the
time       program used by this function.

(more explanation of columns listed here)

                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.00% of 263.00 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00  263.00                 main [1]
              263.00    0.00       1/1           mandelbrot(_IO_FILE*) [2]
                0.00    0.00       1/1           write_header(_IO_FILE*) [9]
-----------------------------------------------
              263.00    0.00       1/1           main [1]
[2]    100.0  263.00    0.00       1         mandelbrot(_IO_FILE*) [2]
-----------------------------------------------
                0.00    0.00       1/1           __do_global_ctors_aux [14]
[8]      0.0    0.00    0.00       1         _GLOBAL__I_x_initial [8]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[9]      0.0    0.00    0.00       1         write_header(_IO_FILE*) [9]
-----------------------------------------------
                0.00    0.00       1/1           __do_global_ctors_aux [14]
[10]     0.0    0.00    0.00       1
__static_initialization_and_destruction_0(int, int) [10]
-----------------------------------------------

 (more explanation of the above table printed here)
```

In the first table generated by gprof, the third column indicates how much time was spent executing each function. Since essentially all of the CPU time was spent on the mandelbrot function, that is the only one that the programmer needs to worry about optimizing. It is very typical that the majority of the CPU time executing even a complex program is spent in executing just a handful of the functions.

In the example above each function was called only once (column 4 of the first table). However, if there were a function called millions of times, it would show up here. The second table shows where each function was called from.

Because the program was compiled with the "–pg" option, it generated an additional file, named gmon.out when it was executed. The profiler is analyzing both the executable and the gmon.out file to give an analysis of how the program performed for this one specific calculation. Ideally the test calculation should run anywhere from 5 minutes to 30 minutes. If the running time is too short, there won't be enough data to give useful results. If the running time is too large, it will be inconvenient to do the optimization work.

Note that profilers give information only to a certain accuracy. This is because the operating system kernel tracks the time threads are running only to a certain accuracy. As a general rule of thumb, times of a second or more are meaningful for a five minute calculation, but fractions of a second are not. More understanding of the source of these errors can be gained by investigating kernel accounting based on a "jiffy counter" (currently used in all versions of Linux) and the alternative, more accurate, scheme called "microstate accounting" that is currently available only in the Solaris version of Unix available from Sun Microsystems.

Once the sections of code needing to be optimized have been identified, the programmer must find a way to get the exact same result more efficiently. Here are some of the most common ways to do this. These start with number 1 being the most likely to help, working down to ones that give less significant improvements.

1. See if there are known algorithms for doing the same task that have a better time complexity than the current algorithm.
2. Look at nested loop structures.
    1. Add a test to determine whether an inner loop needs to be executed on a given iteration.
    2. See if a function can be moved from an inner loop to an outer loop.
    3. See if the loops can be reordered.
3. Look for values that are being computed more than once. Compute them once and store them in a variable, array, or even a file.
4. Are values being computed, which could be replaced by constants? For example, replace sqrt(2.0) with 1.414213562
5. Cut down on the number of times a given function is called.
6. See if sorts can be replaced by a data structure that maintains the data in order.
7. Look for large data moves that could be replaced with pointers.
8. Minimize the use of transcendental functions. For example X*X*X is faster to evaluate than pwr(X, 3.0).
9. Replace single character file I/O with block reads and writes.
10. Programs that do very large amounts of file I/O can be limited by I/O. It is sometimes productive to store data in memory instead of files, or to recompute values rather than storing them for reuse later.

Programming Best Practices

Well written software is a pleasure to work with, and to write. Badly written software can be a nightmare for everyone involved. Degree programs tend to focus on learning programming languages, data structures, algorithms, and architecture. Once you have graduated (regardless of having a degree in computer science, chemistry or some other field) you will find that any professionally run organization has a strong emphasis on good programming practices. Many organizations have code reviews where your boss and senior coworkers review the source code you have written and critique how you could have done a better job. The details of programming best practices vary from one organization to the next, but the same topics are addressed. The following paragraphs discuss the most important aspects of programming best practices. References to additional information are in the Bibliography.

Code basics

Variable naming is important to remind yourself, and programmers that come after you what the variable does. Consider the following examples

`d`	this tells you nothing
`distance`	a bit better
`distanceToSun`	now we know what distance
`ldDistanceToSun`	prefix indicates locally scoped, double precision
`ldDistanceToSunKM`	excellent, we know what it is, and the units

Many companies have variable naming standards. Probably the most comprehensive is the Hungarian Naming scheme created at Microsoft, and used by many other organizations.

Indents in code show which lines are inside of loops, “if” statements, etc. Even archaic Fortran allows indents past the seventh character.

Comments are sections of code that are not compiled or executed. These are where you put notes about what the function does, what it assumes about input data, longer descriptions of variables, why you inserted specific lines of code, etc. Many people have regretted not putting in enough comments. We have never heard of a problem with too many comments.

Use **dynamic allocation.** Dynamic allocation is setting the size of arrays at run time, not at compile time. This prevents problems from arrays being sized too small, or the program using too much memory to run small problems.

Do not hard code paths or input values. Program input should be in a file that is read by the program, or provided interactively through a graphic interface. The path to the location of needed files can be set in an input file, on the command line, or in an environment variable.

Enumerate flags. It is often necessary to tell a function how to select from optional behaviors. Doing so with integers makes the code difficult to read. Enumerate the flags so that you can use English words as flags.

Create plenty of **error traps.** It is immensely aggravating to debug software that passes garbage data from one function, to another, to another, to another. Error traps are "if" statements that check that the data sent to a function is valid, and if not print out a usable message with the function name, problem, and data value. As a matter of self discipline many organizations require error traps to also halt program execution. If error traps slowing down code execution is a concern, have a way to turn them off through the use of a command line flag, or compile time `#ifdef` statement.

Expect user input to be wrong. A user's first impression of a program is based how well it behaves when they are initially learning to use it. Users can, will, and do leave out portions of the input, put in extra input, put text in numeric fields, and put in combinations of options that make no logical sense to an expert in the field. As such, any user input should be heavily error trapped. Error message should give a clear English description of what is wrong, and if possible how to fix it. Even the most experienced users will occasionally make mistakes and appreciate good checking of input data. Good input error traps and descriptive error messages will also cut down on the number of technical support calls you get.

Error trap files and directories. Always check for the existence of a file before opening it, or the existence of a directory before using it. In many languages the file open command returns an error code to make it easier to write error traps.

Software architecture

Writing code is not the first step in software development. The first step is typically to define who will use the software and their use cases. **Use cases** are examples of the types of tasks that must be performed using the software. Each task can then be broken down into a set of steps, input values, buttons, command line flags, etc.

Software can be **prototyped**. If the core of the software is dependent upon creating a new algorithm for solving some scientific problem, it might be best to test that algorithm before writing the whole program. This is sometimes done with a lighter duty scripting language. Graphical software can have paper prototypes showing how it will look, and even used for initial usability testing.

Design the software's high level architecture. There are many discussions of this in books on **design patterns**. Design patterns are high level architectural constructs such as a client-server architecture, wrappers, interpreters, etc.

Give careful consideration to the use of **third party function libraries.** Libraries can save large amounts of software development work. However, if the library is poorly maintained it might introduce errors into the code, incompatibilities with certain operating systems, etc. Every library used increases the difficulty of getting the software to install and compile on another computer.

Avoid machine dependent code. If the software uses functions specific to one operating system or hardware platform, it can be difficult or impossible to get it to run on other computers.

Manage shared object libraries. Shared objects (dll's in Windows) are libraries of functions that are loaded into memory only once, even if being used by multiple programs. This saves memory and disk space. However, your software can break if a new version of the needed library is installed on the system. Some developers static link all executables to avoid these programs, and improve run time performance. Some manage paths with wrapper scripts or the Modules functions described elsewhere in this manual. Others include all needed libraries with the code and force the paths to use that copy, even if it is duplicating something that is part of the operating system.

Organization

Have an **automated build process.** This can be done with the Linux "make" command, the "ant" software, or shell scripts. In any case, there should be a single command to compile the whole software package.

Use a **code repository.** These are systems that store all of (and all previous versions of) the source code, often including documentation, tests, and example input data. The code repository database will have a mechanism to undo changes that broke the software, and to see what the code looked like in any previous version of the software. There are public domain version control systems such as Mercurial and Subversion as well as web portals and commercial software packages. There is an overview and comparison of version control systems on the supercomputers in the file **/opt/asn/doc/subversion/version_control_systems.txt**

Have a **database of all known bugs** in the software. This can be as simple as a ring binder, but is most often a relational database with a web front end. Most bug tracking systems also have mechanism to enter feature requests, assign bugs to developers, and track what version the bug was fixed or identified in.

Have **code reviews.** A code review is a meeting in which source code is examine by technical management and senior coworkers. Even the most diligent programmer will become lax if no one ever looks at their code. Knowing that lax programming will be publicly discussed keeps people adhering to the standards.

Establish appropriate **project management practices**. There are many software project management methodologies with names like waterfall, spiral, agile, and extreme. Which is best depends upon whether the software is dependent upon experimental algorithms, is highly cost driven, or will be developed for many years to come. There is an overview of software project management methodologies on the HPC systems in the file **/opt/asn/doc/mercurial/sw_devel_methods.txt**

Validation and refinement

Functional tests are tests of the entire program. These tests and various subsets of them are usually automated. Functional tests verify that correct results are being generated and that functions are working together correctly. There is often an automated mechanism that runs a good selection of functional tests every night.

Unit tests are tests of individual functions within the software. Unlike functional tests, unit tests can tell which subroutine is broken. Unit tests are used when routines are written and modified. If the final program is expected to be more than a few thousand lines of code, start writing unit tests from day one and require programmers to check in the unit tests when they check in the code. Failure to do so will result in not knowing which functions to trust and having to constantly hand check function after function as part of debugging.

Profile the software, and **optimize** key sections for performance. Especially for modeling and simulation software, a few weeks of performance optimization can save years of run time.

There are **static code analysis programs**, and **run time analysis programs**. These identify problems such as memory leaks, and many types of programming errors. Often turning on all compiler warnings can find a large percentage of these problems.

If **security issues** are a concern, use testing techniques specifically designed to identify those problems.

Documentation

Don't expect the documentation person to do your job for you. Writing documentation is part of programming, even if there is a documentation expert to pretty it up.

User manuals are as important as the software itself. User manuals must describe the functions of the buttons and input values, but that is not sufficient. It is necessary to have documentation discuss reasonable values, and provide tutorials. Really good documentation describes when and why you would use a particular setting.

Self documenting code is a myth. Write a **developers guide**, even if it is just a text file. Describe the code architecture, object classes, build processes, and other development specifics. If the new intern is asking piles of questions or can't get anything done, you need a better developers guide.

Include **example files** with the software distribution, so users can see a working input and the corresponding output.

Bibliography

This manual undoubtedly will not cover everything you need to know. This section is provided to suggest some other useful references. There are several important points to note in addition. First, with the rise of Linux there is now a large amount of useful information freely accessible on the internet (and an even larger amount of useless information). Second, many, many books published by O'Reilly have proven to provide consistently high quality information on topics related to computing. Documentation for specific software packages in available by logging in on the HPC systems and typing the command **`ascdocs`**

System specific information

Documentation from SGI is available online at
http://techpubs.sgi.com/library/tpl/cgi-bin/init.cgi

Tutorials for beginning users of Linux

Kiddle, Oliver , Jerry Peek, Peter Stephenson. From Bash to Z Shell. Berkeley, CA, Apress L.P., 2004.

Newham, Cameron. Learning the Bash Shell 3rd Edition. Sebastopol, CA, O'Reilly & Associates, Inc., 2005.

Blum, Richard. Linux For Dummies, 9th Edition. Hoboken, NJ, Wiley Publishing, 2009.

Printed compilations of Linux commands

Barrett, Daniel. Linux Pocket Guide. Sebastopol, CA, O'Reilly & Associates, Inc., 2004.

Siever, Ellen, Robert Love, Arnold Robbins, Stephen Figgins. Linux in a Nutshell 6th Edition. Sebastopol, CA, O'Reilly & Associates, Inc., 2009.

Volkerding, Patrick, Kevin Reichard. Linux System Commands. New York, NY, John Wiley & Sons, 2000.

Hughes, Phil. Linux for Dummies Quick Reference 3rd edition. Hoboken, NJ, Wiley Publishing, 2000.

Books about using specific Linux commands

Robbins, Arnold, Elbert Hannah, Linda Lamb. Learning the vi and Vim Editors. Sebastopol, CA, O'Reilly & Associates, Inc., 2008.

Information on writing shell scripts

Burtch, Ken. Linux Shell Scripting with Bash. Indianapolis, IN, Sams Publishing. 2004.

Robbins, Arnold, Nelson Beebe. Classic Shell Scripting. Sebastopol, CA, O'Reilly & Associates, Inc., 2005.

Cooper, Mendel. Advanced Bash-Scripting Guide. http://www.tldp.org/LDP/abs/html/
Kochan, Stephen, Patrick Wood. Unix Shell Programming. Carmel, IN, Hayden Books, 2003.

Information on parallel programming

Chandra, Rohit, Leo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, Ramesh Menon. Parallel Programming in OpenMP, San Francisco, CA, Morgan Kaufmann, 2000.

Dongarra, Jack, Ian Foster, Geoffrey Fox, Ken Kennedy, Andy White, Linda Torczon, William Gropp. Sourcebook of Parallel Computing. San Francisco, CA, Morgan Kaufmann, 2003.

Gropp, William, Ewing Lusk, Anthony Skjellum. Using MPI. Cambridge, MA, MIT Press., 1999.

Sloan, Joseph. High Performance Linux Clusters. Sebastopol, CA, O'Reilly & Associates, Inc., 2004.

Wadleigh, Kevin, Isom Crawford. Software Optimization for High Performance Computing: Creating Faster Applications (HP Professional Series) Prentice Hall, 2000.

Sanders, J., E. Kandrot. CUDA BY EXAMPLE. Upper Saddle River, NJ, Addison Wesley, 2011.

Sources of information on time complexity of algorithms

Big O Notation, Wikipedia
http://en.wikipedia.org/wiki/Big_O_notation

Young, David. Computational Chemistry; A Practical Guide for Applying Techniques to Real World Problems. New York, NY, John Wiley & Sons, 2001.

Most text books on algorithms and data structures discuss time complexity.

Programming best practices

McConnell, Steve. Code Complete: A Practical Handbook of Software Construction 2nd Edition. Redmond, WA, Microsoft Press, 2004.

McConnell, Steve. Rapid Development: Taming Wild Software Schedules. Redmond, WA, Microsoft Press, 1996.

Sutter, Herb, Andrei Alexandrescu. C++ Coding Standards; 101 Rules, Guildelines, and Best Practices. Boston, MA, Addison-Wesley, 2005.

Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Boston, MA, Addison-Wesley, 1994.

Hass, Anne Mette Jonassen. Guide to Advanced Software Testing. Boston, MA, Artech House, 2008.

Secure coding standards
https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards