Alabama Supercomputer Authority

HPC USER MANUAL
Ninth edition

Alabama Supercomputer Authority
686 Discovery Drive
Huntsville, AL 35806

The Alabama Supercomputer Authority

HPC User Manual

Ninth 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

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** or by contacting the helpdesk at 800.338.8320 or helpdesk@asc.edu

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.

The Alabama Supercomputer Authority

The Alabama Supercomputer Authority (ASA) provides high performance computing resources to state academic users, state government agencies, national 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. Technical services are provided through professional services and facilities management contractor CSC (formerly Computer Sciences Corporation). See Chapter 10 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. Network services provided include access to ASA high performance computing resources, Internet access, World Wide Web services, and training. ASA provides email and web hosting services for a number of customers. These include some large repositories, such as the Alabama Virtual Library (AVL). A number of customers also host disaster recovery equipment at the Alabama Supercomputer Center.

ASA's high performance computing resources include a SGI Altix 450 supercomputer and a Dense Memory Cluster (DMC). Usage of these systems is free for academic usage by faculty and students at public 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 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, like this

```
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.

Online Help

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. 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
```

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

Documentation from SGI is available online at
http://techpubs.sgi.com/library/tpl/cgi-bin/init.cgi

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 help desk is available 24 hours a day to assist with problem solving and to answer user questions about the status of the ASA systems and AREN.

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

Applications Analyst Email:	hpc@asc.edu
Help Desk Phone (in Huntsville):	(256) 971-7448
Help Desk Phone (outside Huntsville):	(800) 338-8320
Help Desk Email:	helpdesk@asc.edu

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 error log file
- The job number from the queue

If the output is rather large or multiple files, you can alternatively tell the support staff the directory name and file name to look at.

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.

Application Program Support: The analyst provides support for installation of programs, limited optimization of code, 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, benchmarking of codes, 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. 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 course work, thesis research, or work to be published in the peer reviewed literature. Work that will become the unpublished property of the funding agency 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 student 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 established, 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. The user can change the email address to which the renewal reminder is sent by editing the .forward file in their account. 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, the student's research adviser is contacted to see if they need the files, then the account is deleted.

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 CPU hour. The minimum initial purchase is $2000 with additional time purchased in increments of $1000. 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.

ASC Accounting System

The computer accounting system for the Alabama DMC cluster and SGI Altix supercomputers uses standard Linux accounting data with several enhancements. The enhancements provide the following functions:

- Tracking average and peak utilization
- Computing dollar values of the usage of each user
- Tracking total monthly usage for each campus, department, user, and application
- Tracking the work volume in each queue

The Alabama Supercomputer Authority reviews month-end accounting report files. Actual dollar amounts from these reports are used by the Authority's accounting system to compute the amount billed to commercial users and users with software royalty charges. Unsponsored academic research accounts are not billed directly.

The accounting system and related procedures handle all the administrative requirements for accounting. Procedures in place for adding new accounts, deleting unused or expired accounts, and modifying existing accounts include actions to handle such situations as changing rate schedules, changing expiration dates, and handling security in case of forgotten passwords. The helpdesk or applications analyst can help solve problems relating to such issues.

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 (10 GB). Users can request up to a 200 GB quota 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 200 GB can be sent to Donna Daniel, ASA Director of Client Services, at **ddaniel@asc.edu** or **334-242-0175**.

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 "**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 amount of disk usage shown by the quota command is updated nightly. Thus if you are over quota and have deleted files, you may have to wait until the next day for the system to show you as being under quota.

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>
```

3. Supercomputer Hardware

There are two high performance computing systems at the Alabama Supercomputer Center. One is a shared memory system, consisting of a cluster of SGI Altix 450 nodes. The other is a locally architected fat node cluster, called the Dense Memory Cluster (DMC). This section of the manual describes the 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 (**altix.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.

The home directories are backed up nightly. Those backups are saved only one to two weeks. Users who have accidentally deleted files should contact the supercomputer center staff immediately at **hpc@asc.edu** .

There are some differences between clusters also. There is a scratch file system mounted as /scratch in the directory tree. Each cluster has a completely different scratch file system, running on different hardware, and containing different files.

Users compiling their own software must keep in mind that software compiled on the Altix will not execute on the DMC and vice versa. Both clusters are capable of executing generic 32 bit x86 linux executables. However, the 32 bit executables run in an emulation mode on the Altix which runs much slower than on a 32 bit desktop computer. 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.

Both clusters share the same Torque/Moab queue system. This means that, for example, jobs submitted from the DMC login node may actually get run on the Altix. 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 Altix is a cluster of shared memory nodes. 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 Altix Shared Memory Supercomputer

The ASA SGI Altix system is a cluster of SGI Altix 450 nodes. SGI Altix 450 nodes can have up to 72 CPU cores and just under one terabyte of memory per node. The login node (altix.asc.edu) is an SGI Altix 350 node with 6 CPUs and 8 GB of memory.

The SGI Altix 450 series is physically constructed of vertical 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. At the time this manual was written, the Alabama Supercomputer Center had three Altix 450 compute nodes. Two of those nodes (altix7 and altix8) contain 72 CPU cores and 432 GB of memory. One node (altix9) contains 12 CPU cores and 464 GB of memory. These are 1.6 Ghz (altix7 & altix8) and 1.67 GHz (altix9) dual-core Intel Itanium2 processors with a maximum result rate of 6.68 GFLOPs. The web site **http://www.asc.edu/supercomputing/hardware.shtml** should be consulted for the number of CPU cores and amount of memory currently installed.

File Systems and Infrastructure Servers

There is a Panasas storage array for storage of data. This collection of storage nodes is broken into four volumes. Two of these are PanFS mounted as /apps and /home on both the Altix and DMC clusters. One volume is served via the PanFS file system as /scratch on the Altix nodes only. One is PanFS mounted as /scratch on the DMC nodes only. The Panasas arrays connect to the nodes via 10 Gigabit Ethernet, Infiniband, and Gigabit Ethernet.

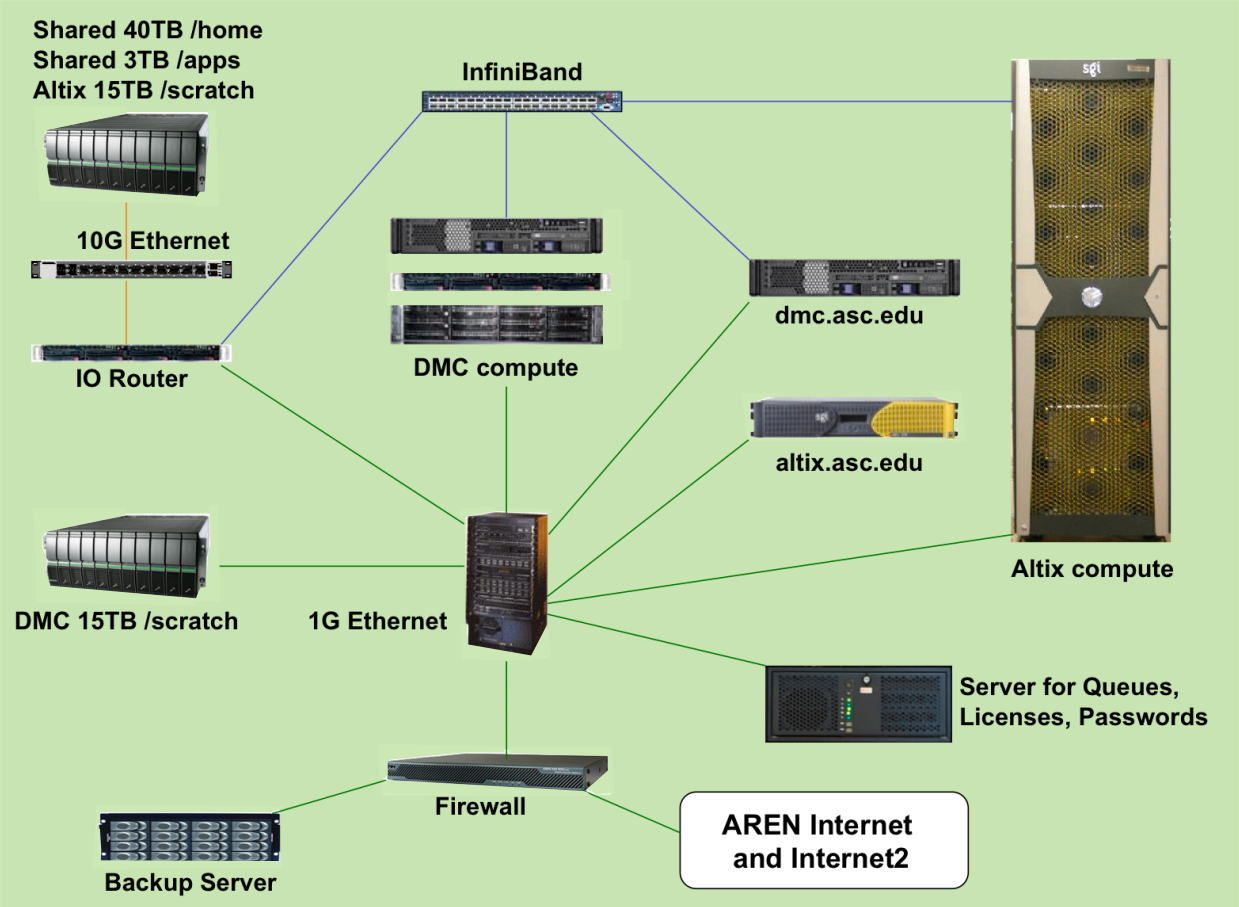
The infrastructure for the supercomputers also includes servers for passwords, group memberships, software licenses, operating system updates, and the queue system. Figure 3.1 shows a view of this infrastructure. The servers and network gear shown in the bottom third of this diagram are common to both the Altix and the DMC.

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. Components for the DMC were obtained from Microway, Penguin, Cisco, Voltaire, Linksys, Cluster Resources, Novell, SGI, Spectrum, Avocent, and Panasas. 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

Figure 3.1

The file system and network infrastructure.



the majority of the jobs at the Alabama Supercomputer Center at an optimal price point.

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 3.1 . The first 20 nodes have processors that do two floating point operations per clock tick, the rest do four floating point operations per clock tick. The newer DMC nodes (dmc21 and up) are physically configured as "twin" systems with two complete nodes in each 1U of rack space, as shown in Figure 3.2 . The newest nodes (dmc61 and up) have redundant power supplies.

Table 3.1 DMC Nodes

Nodes	Cores	Memory	Processors
dmc	8	32 GB	3.0 GHz AMD Opteron 8222 dual-core
dmc1 - dmc20	8	64 GB	3.0 GHz AMD Opteron 8222 dual-core
dmc21 - dmc60	8	64 GB	2.3 GHz AMD Opteron 2356 quad-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

At the time this manual was written, the DMC had a total of 1512 CPU cores and 8.2 Terabytes of memory. The web site **http://www.asc.edu/supercomputing/hardware.shtml** should be consulted for the number of CPU cores and amount of memory currently installed.

Figure 3.2

Two compute nodes fit in each 1U rack mount server in the DMC. Each twin node has two quad-core or 8-core processors.

The DMC can run shared memory parallelized applications up to the 16 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 Altix. The DMC has a separate /scratch file system. The scratch file system is a Panasas object-based storage solution.

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. 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

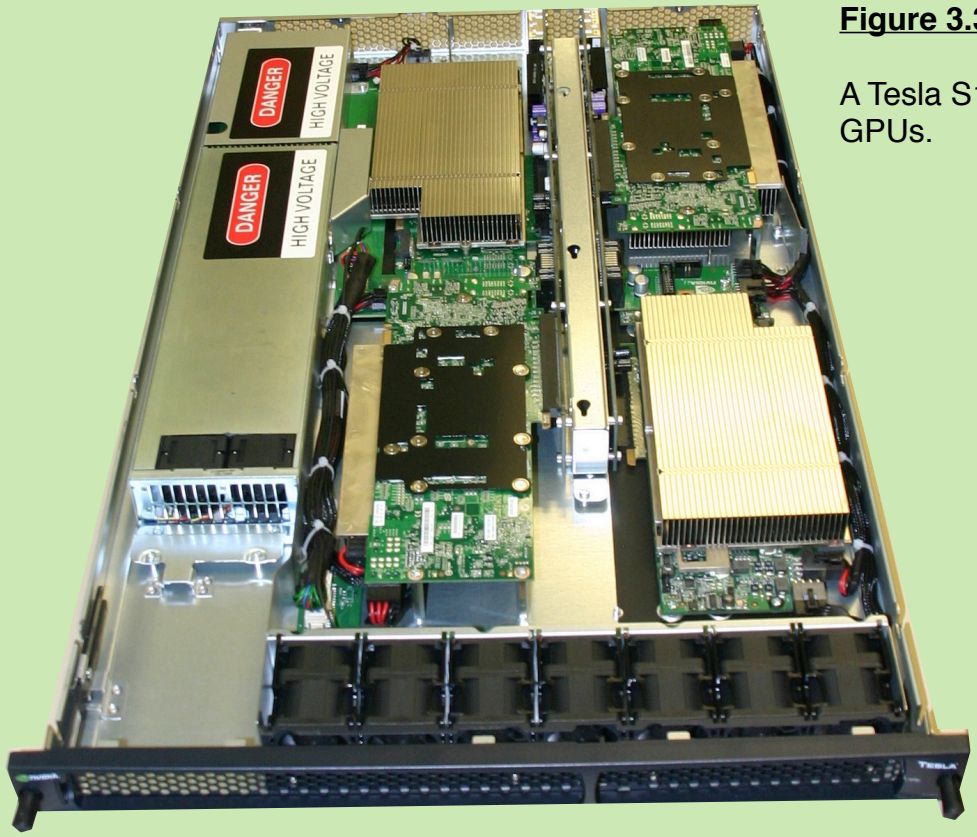


Figure 3.3

A Tesla S1070 with four GPUs.

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, as shown in Figure 3.3. At the time this manual was written, two NVIDIA Tesla S1070s installed. This amounts to a total of eight GPUs and 32 Gigabytes of dedicated GPU memory attached to four DMC compute nodes. The web site **http://www.asc.edu/supercomputing/hardware.shtml** should be consulted for the number of GPUs currently installed.

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**

4. Available Software

The Alabama Supercomputer Authority provides a selection of software to be used on the high performance computing systems. 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.txt files with instructions on how to use the software. The documentation and instructions on how to access each software package can be found on the system in the directory **/opt/asn/doc**

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. 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 see
http://www.asc.edu/supercomputing/software.shtml

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.

Table 4.1 gives is a listing of the software packages that were available at the time this manual was written. Those marked with an asterisk (*) require an additional fee for commercial usage.

Table 4.1 Available Software

Bioinformatics	Programming	Quantum Chemistry
AMOS	GNU Assembler	ACES II *
Bowtie	Blassic	CPMD
CAP3	C/C++	Dalton
ClustalW	Fortran (77, 90, 95)	GAMESS
Cufflinks	GASNet	GAMESSPLUS
GARLI	gprof	Gaussian
Genome Analysis Toolkit	LISP	GaussView
Karma	MPI	GAUSSRATE
Maq	Objective Caml	Jaguar *
Mothur	ompP	JuNoLo
mpiBLAST	OpenMP	LmtART
MrBayes	Perl, Python, awk, tk, tcl	MULTILEVEL *
MUMmer	Unified Parallel C	NWChem
NCBI Toolbox	Totalview	ORCA
NEXUS Class Library		POLYRATE
PHASE	**Molecular**	PSI3
PhyloBayes	**Mechanics / Dynamics**	Quantum-ESPRESSO
RAxML	Amber *	
RepeatMasker *	Autodock	**Other Simulations**
RNA2MAP	Desmond	M5sim
SHRiMP	GROMACS	NS-2
TopHat	GULP *	Sim-Alpha
TRF	LAMMPS	TauDEM
Twinscan	NAMD	
UMFPACK	Tinker	**Semiempirical**
Velvet		AMSOL *
WU-Blast *		MOPAC

Table 4.1 (Continued) Available Software

Mathematics	CFD	Structural Analysis
ACML	CFD-ACE+ *	ABAQUS *
deal.II	CFD-FASTRAN *	Hyperworks *
f2cblaslapack	INS2D	
GAP	INS3D	
IMSL	MM5	**Crystallography**
Lapack++	WRF	ABINIT
METIS		CCP4 *
MKL		CNS *
Octave		X-PLOR *
PETSc	**Visualization**	
R	gnuplot	**Operating Systems**
SCSL	Grace	SLES
SLATEC	NCAR Graphics	Ubuntu
Trilinos	VMD	CernVM

5. 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, 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.

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 (altix.asc.edu or dmc.asc.edu) and <user_id> is replaced by your account name.

EXAMPLE

Examples:

```
local>ssh asndcy@altix.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, enter "**`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 web site is named putty.exe . Usage of psftp.exe and pscp.exe from the same web site 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.

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 5.1 .

Enter the host name, either **dmc.asc.edu** or **altix.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.

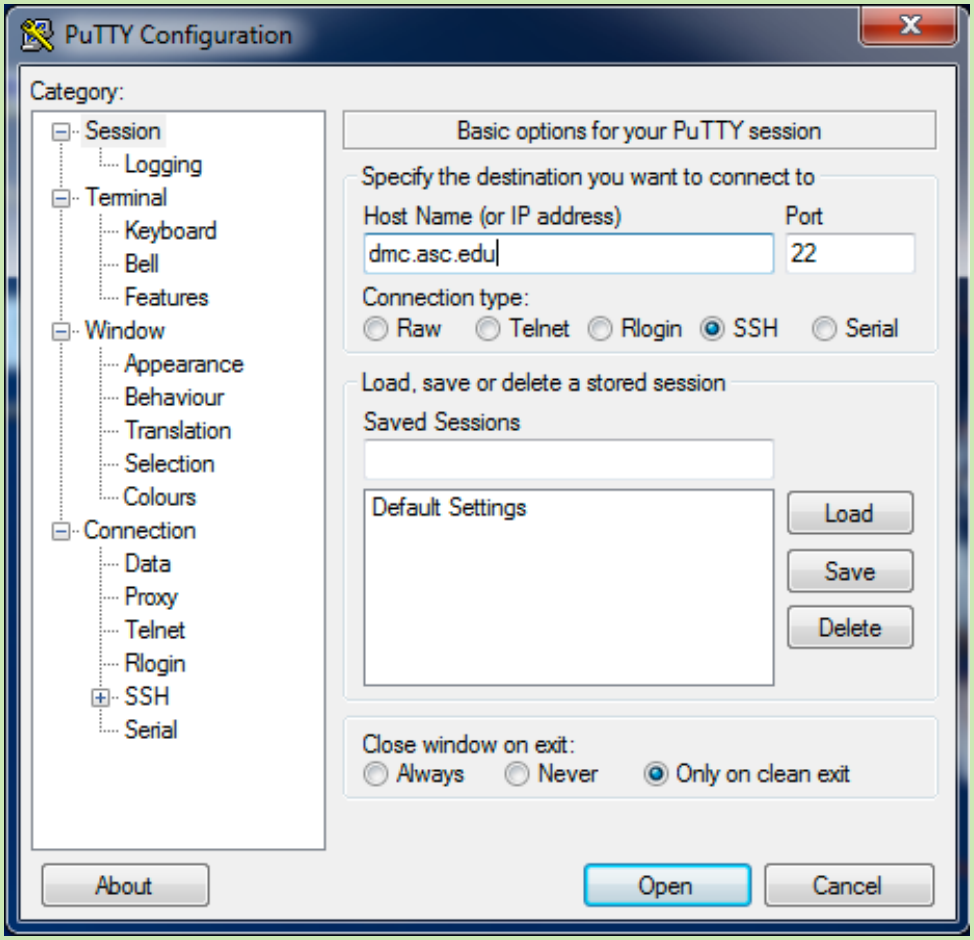


Figure 5.1

The PuTTY configuration window.

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 5.2

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.

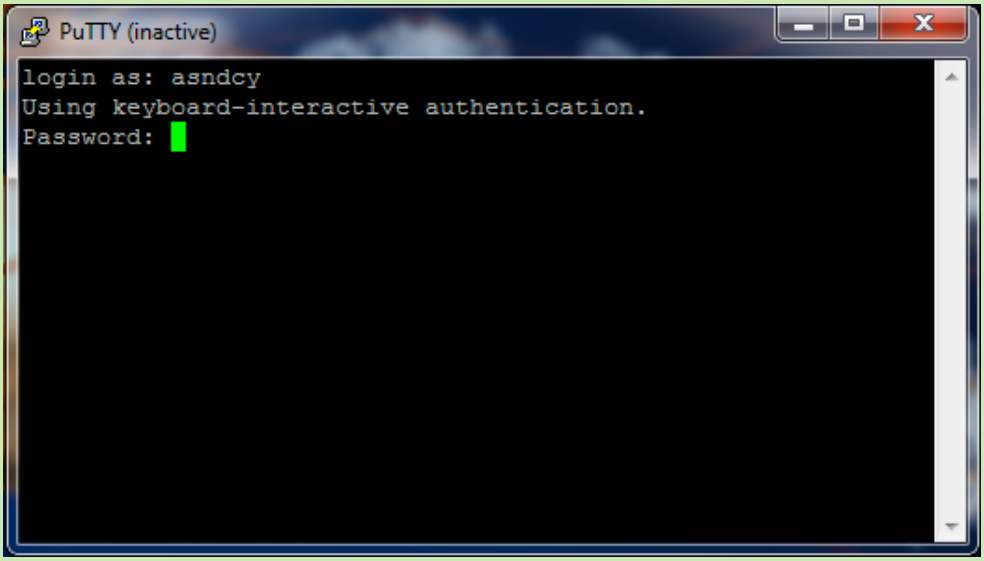


Figure 5.2

The PuTTY terminal window.

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, enter "**`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@altix.asc.edu
Connecting to altix.asc.edu...
asndcy@altix.asc.edu's password:
sftp>
```

Using psftp.exe: Download psftp.exe from the PuTTY web site 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 altix.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 /tmp>
```

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 web site 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.

Savvy users that move large amounts of data around tend to use scp because it requires less typing than sftp.

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 Xming, and Cygwin packages also include X-Windows servers.

Cygwin or Xming work with some Windows versions and not others. Check their web sites for information about using the latest version on your version of Windows. The Xming X-Windows client is easier to install than Cygwin.

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.

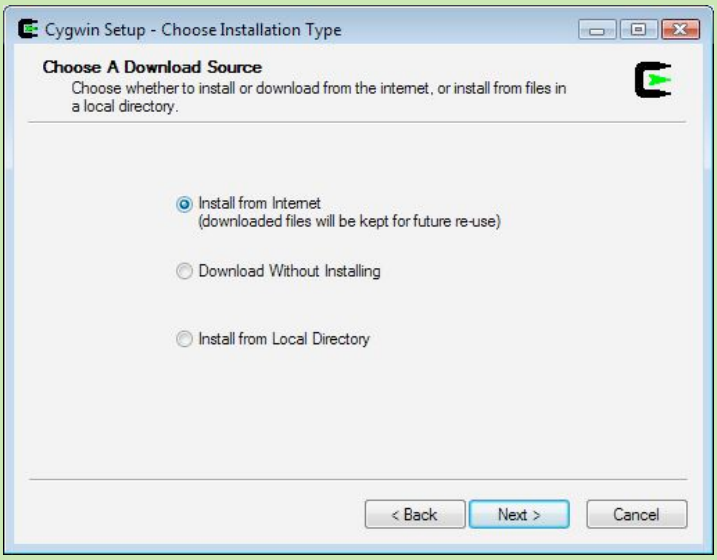
- The default Xming installation settings should work.

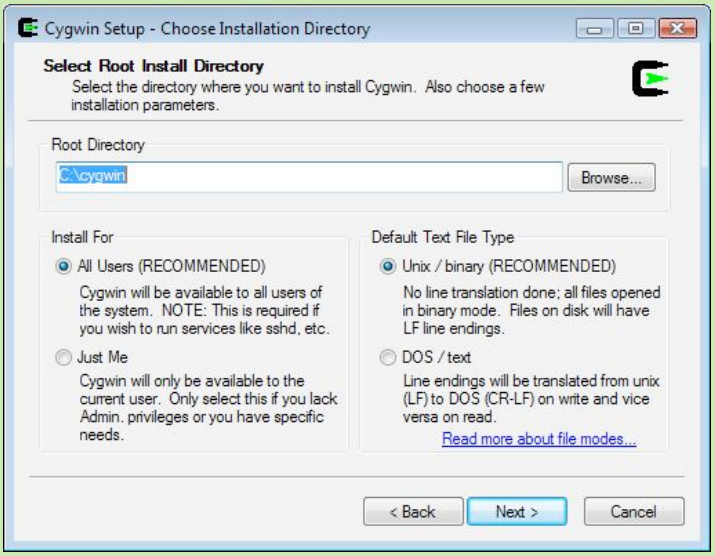
- To start Xming, select Start->All Programs->Xming->Xlaunch Use the default Xlaunch options with the exception of; Start a program, Using PuTTY, and fill in the machine name (i.e. dmc.asc.edu) your userid and password. This should open an xterm window with you logged in on the DMC.

- 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





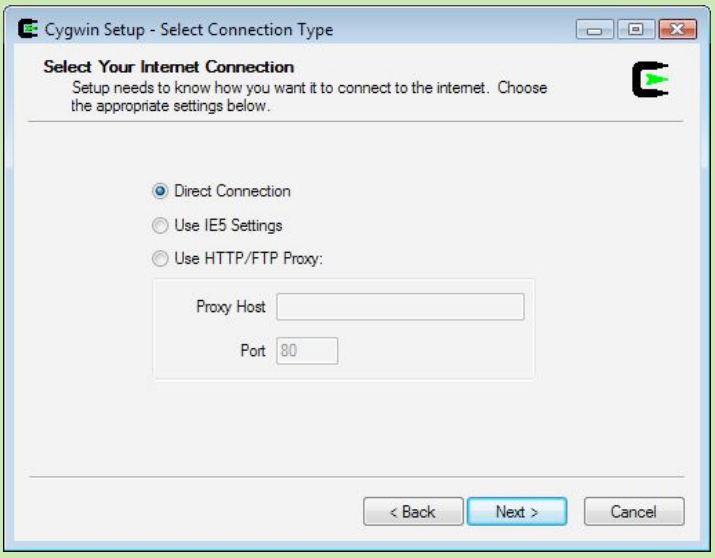


Figure 5.3

Recommended Cygwin installation options are;

Install from Internet
Default directory
Unix text file type
Direct Connection

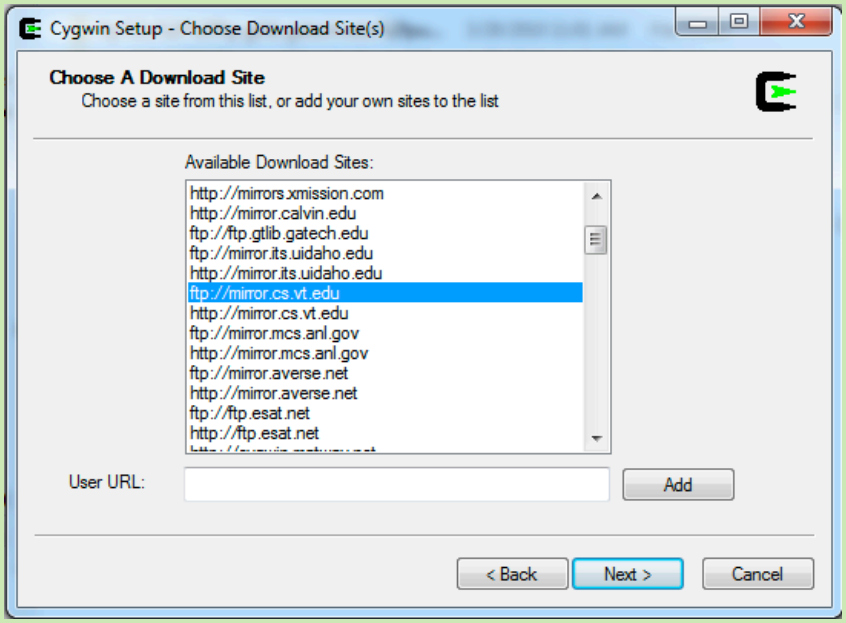


Figure 5.4

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

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 5.3 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 5.4.

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 5.5). 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.

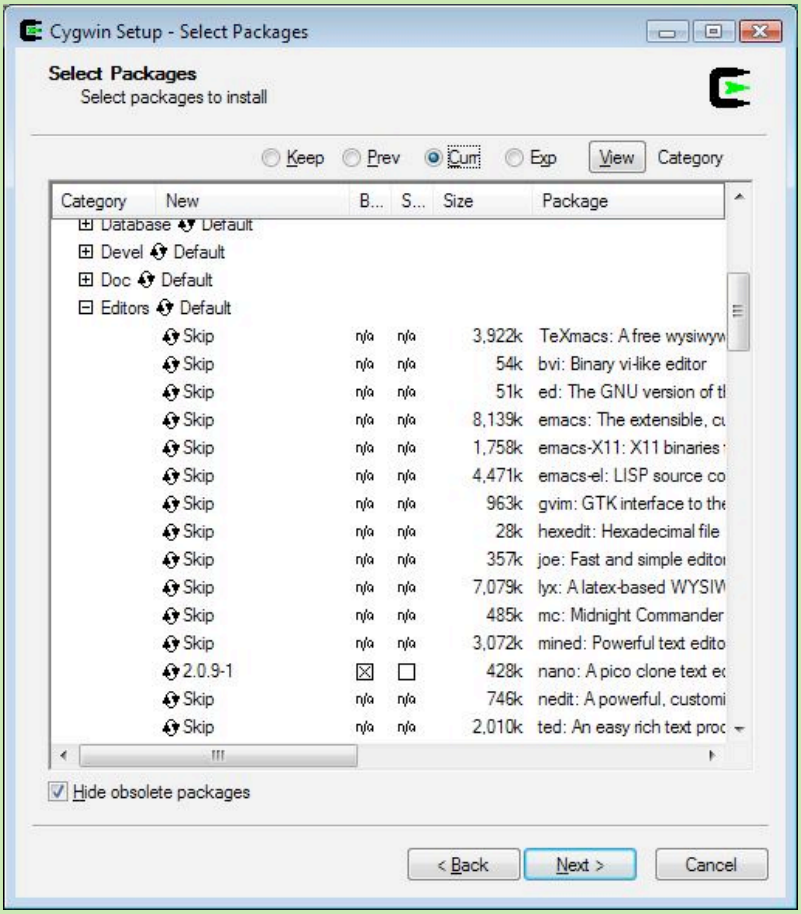


Figure 5.5

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

Adding the following packages to the default selections is recommended, as shown in Figure 5.5.

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)

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> altix.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. Contact your local information technology department to determine if there might be a firewall filtering network traffic between your computer and the AREN network.

6. 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 Altix are running versions of Linux. The Linux version on the Altix 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 Altix 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 (Moab) is a third party add-on to Linux systems. Moab 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 Moab 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 Moab queue system is designed to accept the same commands as for interactive use. You can prepare a file for submittal to the Moab 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.

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 potentially can create confusion.

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.

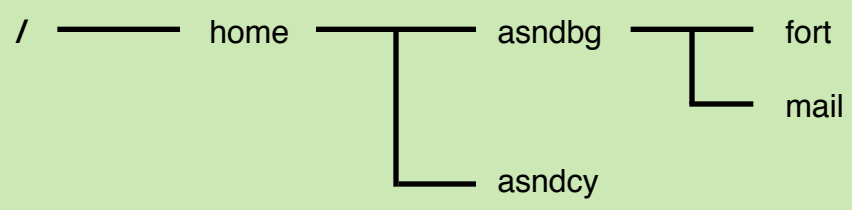
Subdirectories are indicated with a / preceded by the name of their parent directory. If there is a user subdirectory called asndbg in home, for example, the full designation for that subdirectory, starting from the root, would be **/home/asndbg** . The user asndbg might have organized FORTRAN programs 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 6.1 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 6.1

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 6.1, 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 6.2 for locally installed software and user accounts.

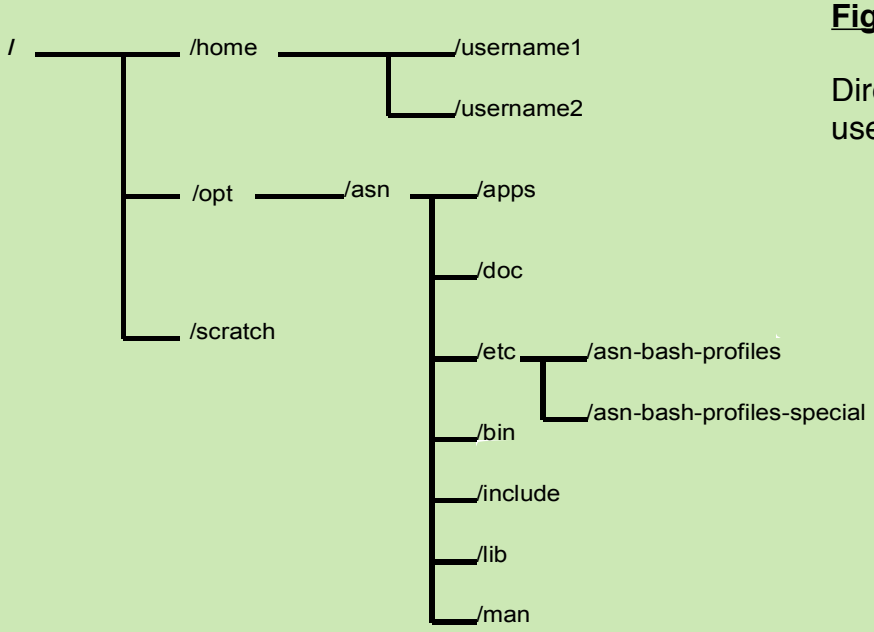


Figure 6.2

Directories used by many users.

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

/opt/asn/etc/asn-bash-profiles-special – Files users have the option of sourcing from their .bashrc.local.altix file or .bashrc.local.dmc file. Sourcing these files may not be needed or safe for all users. Generally if something here needs to be sourced for a given software package, it will be explained in the documentation in /opt/asn/doc .

/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 15 TB of scratch on the Altix and the DMC.

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.local         .flexlmrc
..        .bashrc.local.altix   .ldaprc
.alias    .bashrc.local.dmc     .nwchemrc
.bashrc   .poly_path9.3
```

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.

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. Because of this, 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.altix 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 .poly_path9.3 .flexlmrc .ldaprc and .nwchemrc configure your account to use certain software packages. These files should never be altered.

WARNING: The .bashrc .poly_path9.3 .flexlmrc .ldaprc and .nwchemrc 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 altix, the file command will show that it is an IA-64 architecture executable like this.

```
asndcy@dmc:ia64> file environ
environ: ELF 64-bit LSB executable, IA-64 (Intel 64 bit
architecture), version 1 (SYSV), for GNU/Linux 2.4.0,
statically linked, not stripped
```

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
```

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 the page. Typing "**q**" exits more. On most Linux systems "more" and "less" behave identically.

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.

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 systems 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>`**".

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
```

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 hpc@asc.edu
```

sends mail to hpc (user help on the supercomputers) at the address asc.edu. 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 hpc@asc.edu <errors.txt
```

would send the previously prepared letter to the supercomputer help email address.

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.

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 "qstat" 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>
```

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**".

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 "qstat" 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 jobs are running through the queue system by typing the following.

```
qstat
```

The "qstat" command show one line of text for every job submitted to run on the compute nodes through the queue system. Because of this, the output from qstat can be hundreds of lines of text. It is inconvenient to wade through all of that information to find out about your own jobs. However, Linux gives a convenient way to find just the desired information. For example, if user asndcy wants to see just their own jobs, they could type the following;

```
qstat | grep asndcy
```

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 "asndcy".

For an example of piping more than two commands together, we could count how many jobs user asndcy has running by typing the following;

```
qstat | grep asndcy | 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. For example, the export commands in the hidden account setup file .bashrc.local.altix can be displayed with the following command.

```
grep export .bashrc.local.altix
```

It often happens that some of the lines in the .bashrc.local.altix file are commented out by putting a pound sign (#) at the beginning of the line. It is possible to display just the commented out export commands like this

```
grep "#export" .bashrc.local.altix
```

The regular expression is put inside of double quotes so that Linux will know that the pound sign is part of the regular expression, not performing another function (commenting out a script line).

To display the export lines that are not commented, use a command like this;

```
grep export .bashrc.local.altix | grep -v "#export"
```

In this example, 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 aaaaaaaaaaaaabc 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.

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.

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>
```

Figure 6.3

The nano text editor.

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 6.3. 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 **CTRL** 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.

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.

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 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. The example session is all part of one on-going example throughout the narrative.

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.

EXAMPLE

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 myqstat

```
#!/bin/sh
#  This shows only my jobs from qstat
myname=`whoami`
qstat5 | grep $myname
```

Once the chmod command is used to make myqstat executable, you can type "**`myqstat`**" 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.

7. 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.

The queue system is the researchers 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.

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 must be run through the queue system.

The queue system at the Alabama Supercomputer Center is a Torque queue system with a Moab scheduler. This is often referred to as a Moab queue system in order to minimize confusion with other types of Torque installations, such as those with the Torque or Maui schedulers.

The Moab 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 one of the Altix compute nodes. 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 is called without any arguments. Calling qlimits gives an output like the following

```
Queue                  CPU     Mem # CPUs
-------------- ---------- ------ ------
large-serial    240:00:00   35gb      1
medium-serial    90:00:00    4gb      1
small-serial     40:00:00    1gb      1
medium-parallel 100:00:00   32gb   2-16
large-parallel  240:00:00  120gb   2-64
small-parallel   48:00:00    8gb    2-8
sysadm          168:00:00 1280gb 1-1000
class             2:00:00   30gb   1-16
express           01:00:00  500mb      1
special        1008:00:00  340gb   1-80
commercial     1008:00:00  360gb   1-16
daytime           4:00:00   16gb    1-4

Interactive limits:
                       CPU     Mem   File
---------------  ------- ------ ------
INTERACTIVE         600sec    4gb   40gb
```

The "CPU" column in this output gives the amount of time, per CPU 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 "# CPUs" column shows how many CPUs can be requested by a job in that queue

The express queue is accessible to everyone on the system. Express jobs are limited to 1 hour of CPU time, making it unusable for the majority of research calculations. However express jobs get into a run state almost immediately.

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 and large 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 "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** 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 CPU hour for access to the computing resources. For a quote on purchasing CPU 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 "qstat" command shows what jobs are running and pending in the queue system. The qstat command can be called with or without arguments. For example, the following are the most common ways of running qstat

```
qstat
qstat -an
qstat -f <job_number>
```

The first two of these commands show a list of jobs. The job number is in the left hand column. The numeric part of the job number can be used to get more information about the job, kill the job, or request help from the ASC staff. These commands also show a job status indicated by R or Q. If the status is R the job is running. If the status is Q the job is waiting to run. The flags "-an" cause qstat to also output which compute nodes the job is running on. The "-f" flag gives a full listing of a large amount of information available to the queue system about that job.

The staff at the Alabama Supercomputer Center have written a number of scripts to show what the queue system is doing in more convenient formats. These can be used by typing "**qstat4**", "**qstat5**", "**qstat6**", "**qstat7**" or "**qstat8**".

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 single sentence description of why the job isn't running. A much more verbose set of information about the job can be displayed with the command

```
checkjob -v <job_number>
```

The command "**mdiag -p**" can be used to display information about the priority ranking of pending jobs.

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, stderr output from the job, and a listing of resource utilization. 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.

The resource utilization section of the error log file looks like the following;

```
########################################################################
# Your job finished at : Mon Nov  8 13:10:39 CST 2010
# Your job requested :
cput=336:00:00,mem=2gb,neednodes=1:ppn=2,nodes=1:ppn=2,walltime=235:
00:00
# Your job used :
cput=00:00:01,mem=4372kb,vmem=35580kb,walltime=00:00:32
# Your job's parallel cpu utilization : 1%
# Your job's memory utilization (mem) :  0.21%
#         Alabama Supercomputer Center - PBS Epilogue
########################################################################
```

This entry 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 2012kb (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.

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 10% 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

```
qdel <job_number>
```

Use only the numeric part of the job number, not the .mds1 extension.

Running Existing Applications Software

Applications software installed by the ASC staff have both an associated queue script and a README.txt file with notes specific to running the software on the ASC systems. The README.txt 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.

```
rung03 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                  CPU    Mem # CPUs
-------------- ---------- ------ ------
commercial       20:00:00   80gb   1-16
small-parallel   48:00:00    8gb    2-8
sysadm          168:00:00 1280gb  1-160
large-serial    168:00:00   35gb      1
express          01:00:00  500mb      1
medium-serial    90:00:00    4gb      1
small-serial     40:00:00    1gb      1
medium-parallel 100:00:00   32gb   2-16
large-parallel  168:00:00   64gb   2-64
special        1008:00:00  340gb   1-80
class            01:00:00   30gb   1-16

Interactive limits:
                      CPU    Mem   File
---------------  ------- ------ ------
INTERACTIVE        600sec  100mb   1gb
```

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

Enter number of cpus (default <cr>: 2 ) 2

Enter Time Limit (default <cr>: 96:00:00 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: watercomG03)
water

============================================================
=====          Summary of your Gaussian job             =====
============================================================
  The input file is:  water.com
  The output file is:  water.com.log
  The time limit is  96:00:00  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:  200810131149.36
  Look for:  water  in queue:  small-parallel

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-serial
queue.
For other queues, set %mem and MaxDisk in your input file.

Job number 82641.mds1.asc.edu
```

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.

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 on a single node. There is a “run_script_mpi” command for mpi parallelized software. Consider the example of a user that has compiled a program on the altix. 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 altix
./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 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 altix, the user must enter "**altix**" at this prompt.

Now consider what could be done if the program foo has been compiled twice make both an altix executable and a dmc executable. In this case, the script must be able to identify whether it is running on the altix 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 an altix compute node or a dmc compute node. The "**hostname**" command returns the name of the node. The altix compute nodes have names like altix7 or altix9. 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 "altix" 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 altix or DMC
#

myhost=`hostname | tr -d '0123456789'`

# see if I'm on an altix node
if [ "$myhost" == "altix" ]
then
  foo_path=/home/asndcy/foo/bin_altix
fi

# see if I'm on a dmc node
if [ "$myhost" == "dmc" ]
then
  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 altix 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
```

Using the qsub Command

The application specific queue scripts and the run_script & run_script_mpi commands are specific to the Alabama Supercomputer Center. For most users, this is all they need. However, some users may be more familiar using "qsub" as is done at many other computing centers. The use of run_script and the application queue scripts is recommended, however the following discussion is provided for the benefit of those users that are interested in learning about qsub.

The application run scripts perform a number of tasks, such as setting up paths, environment variables and temporary working directories. These scripts then call the "qsub" command, which is the command that is specific to the Torque/Moab queue system. Other queue systems such as NQS, PBS, Sun Grid Engine, and LSF also have commands to submit a job to the queue system, and many of them call that command "qsub". However, all these different qsub commands do not use the same arguments or syntax. Sometimes a different syntax is needed even for running the same queue system software on two different systems that have been configured differently.

Detailed information on the options available to the qsub command are available by typing "**man qsub**". Additional information is available on the MOAB web page, which is at **http://www.clusterresources.com/products/mwm/docs/index.shtml**
The commands reference is in Appendix G of the manual at
http://www.clusterresources.com/products/mwm/docs/a.gcommandoverview.shtml
The Torque manual is at **http://www.clusterresources.com/torquedocs/**

The queue system has been configured with defaults so that very few arguments are absolutely necessary. In the simplest case, a job can be submitted with the command;

```
qsub <myscript>
```

This will result in submitting the job to the small-serial queue, available to any cluster, using 1 CPU.

The qsub command can be given a rich variety of options to control how the job is run. For example, the following submits the job to the express queue on the dmc cluster.

```
qsub -q express -l partition=dmc <myscript>
```

Here are some other qsub options that are useful.

-I	open an interactive session on a compute node
-joe	send stderr and stdout to the same file
-l cput=12:00:00	request 12 hours of CPU time
-l mem=8gb	request 8gb of memory
-l nodes=1:ppn=4	request 4 CPUs on one node
-r n	Mark the job as not rerunable.
-X	X-window forwarding

WARNING: It is highly recommended that the option "**-r n**" always be included. Failing to do so can result in having the job failing in an infinite loop.

Requesting multiple CPUs on the SGI Altix can be done with the syntax "**-l nodes=1:ppn=8**". This same syntax is used to request multiple CPUs on the same node of the DMC (up to 16 CPUs). To request multiple CPUs across multiple nodes of the DMC, use the syntax "**-l nodes=32**".

It is also possible to open an interactive session on a compute node. This is not generally recommended as it makes inefficient use of the compute node resources. Also, any work being run in this way will be killed if there is a network outage of some sort. It can also take a very long time to establish an interactive session on a compute node, as it may be necessary for the desired amount of memory or CPUs to become available. However, there are rare cases where the ASC staff will recommend this option. An interactive session can be opened by leaving out the script name and including the "**-I**" flag like this.

```
qsub -I -q express -l partition=dmc
```

Efficient Parallel Processing

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.

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.

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

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.

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

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%.

While the job is running, the parallel utilization can be viewed with the "**`qstat7`**" command. When the job is complete, the parallel utilization and memory utilization is put in the error log file.

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.

Note: If parallel utilization is below 75%, the job can be killed because of exceeding wall time limits before it has exceeded it's CPU time limit.

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) 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 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 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.

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 persons program would be killed when another persons 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.

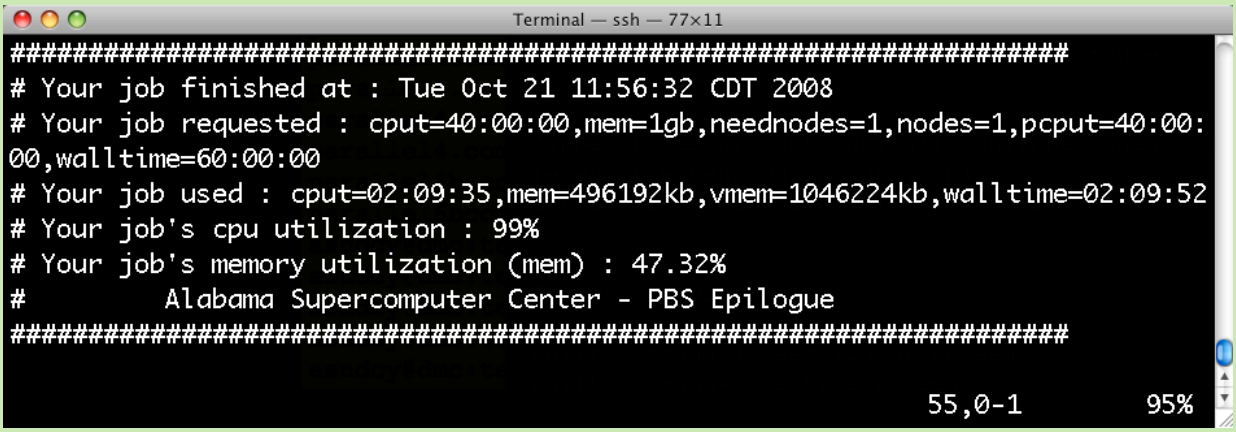
The memory and CPU time needs for many calculations are often not proportional to the size of the input. Consider the example of a software package that computes properties of molecules. This program might need to compute the distance between each pair of atoms. Thus if there are 20 atoms, the distances 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 $\mathbf{O(n^2)}$, 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 $\mathbf{O(n^3)}$ or $\mathbf{O(n^8)}$. 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. At the end of the error log file is an entry that looks like the one shown in Figure 7.1.

Figure 7.1

The resource utilization information at the end of the error log file.



If similar size calculations have already been run, simply looking at the error log file 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.

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.

$$T2 = T1 * (n2 / n1)^3$$

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**

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 NWChem 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**

8. Using Modules

The term "modules" is used in a number of different ways in the computer science field. In this context "modules" refer to Tcl Modules, 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.

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.txt (i.e. /opt/asn/doc/gaussian/README.txt)

In order to use modules, the following must appear in your .bashrc.local file.

```
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.altix & .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.

Modules can be unloaded with the command

```
module unload <module_name>
```

WARNING: Unloading modules doesn't always bring back the default account environment. This is because modules can override default paths. Unloading the module removes the overrides, but does not put back the settings that were overridden. Thus it is sometimes necessary to log off and log back in again after unloading a module.

Sometimes loading a module also loads the "standard" module. The standard module is a special module that strips many of the default paths and configuration settings out of the environment. This is done when some of the defaults could result in a conflict that prevents the software from working correctly.

9. 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 9.1 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 9.1 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 9.2 . Many other hidden files may appear in your account only if you are using certain software packages.

Table 9.2 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
.bashrc.local	settings that affect both altix and dmc
.bashrc.local.altix	settings that affect the altix only
.bashrc.local.dmc	settings that affect the dmc only
.bash_logout	commands run at logout, less often used
.cshrc, .login, .profile	primary configuration file on other shells, Linux, or Unix versions
.flexlmrc	license server configuration
.forward	holds email address where notices are sent
.rhosts	for passwordless rlogin between system
.ssh	directory with encrypted ssh keys & configuration
.vimrc	configuration for the vi editor

WARNING: The .rhosts file and 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 the other accounts as well.

The files .bashrc.local .bashrc.local.altix 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.txt**

The order in which commands are put in these files is sometimes important. For example, the module load commands for some software packages also load the "standard" module, which erases many of the default settings to give a clean SUSE

Linux environment. 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.

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 Tcl Modules 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.txt** (i.e. /opt/asn/doc/gaussian/README.txt).

The "source" command can be put in your .bashrc.local or .bashrc.local.dmc or .bashrc.local.altix 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 web sites such as http://www.linuxselfhelp.com/howtos/Bash-Prompt/Bash-Prompt-HOWTO-2.html

Creating an alias

Aliases are keystroke short cuts. For example, the user asndcy might frequently want to see what jobs he has running in the queue. He does this with a command like this.

```
qstat5 | grep asndcy
```

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 qs="qstat5 | grep asndcy"
```

After setting this, he must log off and log back in for the command to take effect. From then on, he can simply type "**qs**" to check the jobs he has in the queue. An alias can also be used to ensure that a given command line option is always added to a command.

If you have problems with an alias, try typing the command that the alias runs. List this command when contacting technical support.

Tips for Effectively Using the Supercomputers

The following are some suggestions for getting the most out of the altix and dmc.

- 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 less than optimal.
- Learn to utilize the queue system effectively.
- Look at the README.txt file for your program in /opt/asn/doc/PROGRAM
- Put your most frequently checked email address in the .forward file so that you will get notices about your supercomputer usage.

10. 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, 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 and several times since (not yet available). These revisions have added support for newer programming conventions such as pointers. 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 as 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 for the DMC only
- 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_altix** 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.altix and .bashrc.local.dmc files. Settings in .bashrc.local affect both clusters. Those options are documented in the README.txt 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)
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

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 output executable name isn't specified on the compile line, it will be named **a.out**

The following is an example of a console session in which a Fortran program is compiled and executed.

```
altix:~/hello $ ls
hello.f
altix:~/hello $ ifort hello.f
altix:~/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
altix:~/hello $ ./a.out
 Hello
altix:~/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>
```

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.

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. 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, -static, -fast, -fnsplit, -ip, -ipo, -prof_use, -tpp2.

PORTLAND COMPILERS: Other flags that can improve optimization are; -Bstatic, -fast, -O4, Mcache_align, -Mdalign, -Mllalign, -Mfunc32, -Munroll, -Minline, -Mscalarsse.

GNU COMPILERS: Other flags that can improve optimization are; -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.

Using a static compilation flag can improve execution speed performance and make the executable easier to run on a different computer. However, static linking makes the executable take more disk space. In theory, static linking can result in the software using more memory at run time, but in practice this effect is usually very small.

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_altix** and **/opt/asn/doc/compilers_dmc** on both supercomputers.

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.

```
altix:~/source/mandy $ g++ mandy.cc -O3 -pg -o mandy
altix:~/source/mandy $ mandy test5_big
altix:~/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.

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 CVS and Subversion as well as web portals and commercial software packages.

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.

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.

Appendix: ASA Policies

This appendix contains the Alabama Supercomputer Authority policies. Any new revisions to these policies will be posted on the ASA web site at

http://www.asc.edu/usermanual/policies/policymenu.shtml

Privacy and security statements are distributed on the Alabama Department of Finance Information Services Division website at

http://isd.alabama.gov/isd/statements.aspx

Policy: ASAP01 - R4

Title: **ASA Policies**
Date: 01/01/94:Revised 3/8/2005
Distribution: All ASA Users

A. PURPOSE.

This policy defines the ASA policy document as a vehicle for documenting and communicating policies relating to the Alabama Supercomputer Authority (ASA) and its user community.

B. DESCRIPTION.

ASA policies are numbered and controlled documents that define current policies of the Alabama Supercomputer Authority (ASA). Policies defined in these documents will be followed in operating the George C. Wallace Supercomputer Center (GCWSC), the Alabama Research and Education Network (AREN) and other ASA activities. Policies will be numbered ASAP01-R0, ASAP02-R0, etc. Revisions will be indicated by incrementing the revision number: ASAP01-R1, ASAP01-R2, etc.

C. APPROVAL.

All ASA policies will be approved by the Alabama Supercomputer Authority (ASA) and concurred to by the current professional services contractor.

D. FREQUENCY.

ASA policies will be issued and updated on an as-needed basis.

E. DISTRIBUTION.

ASA policies are applicable to clients of GCWSC, AREN, High Performance Computing (HPC) and Alabama Supercomputer Authority (ASA) and its professional services contractor. Copies of policies are available electronically through ASA's web site http://www.asc.edu/usermanual/policies/policy02.shtml

F. SUBJECT MATTER.

ASA policies will address issues which affect users of GCWSC, AREN, and HPC and generally relate to the operational and administrative activities of ASA. While procedures may be defined in the "ASA User Guide", the ASA policy will be the official document for overall guidance.

Policy: ASAP02 - R8

Title: **List of Current Policies**
Date: 08/16/95:Revised 9/1/2009
Distribution: ASA Staff, ASA Clients

A. OVERVIEW.

This policy exists to provide a list of all current ASA policies. This list will be updated on a periodic basis as needed to remain current.

B. CONTENTS.

Policy Number	Revision Number	Revision Date	Policy Title
ASAP01	R4	Mar 8, 2005	ASA Policies
ASAP02	R8	Sep 1, 2009	List of Current Policies
ASAP03	R5	Mar 8, 2005	ASA- Hours of Operation
ASAP04	R7	Sep 1, 2009	HPC Charges for Commercial Clients
ASAP05	R8	Sep 1, 2009	HPC Client File Storage
ASAP06	R5	Mar 24, 2005	HPC Proprietary Program Charges
ASAP08	R6	Apr 21, 2005	ASA User Accounts
ASAP13	R2	Apr 18, 2005	Configuration Change Request
ASAP14	R5	Mar 24, 2005	Access to GCWSC
ASAP17	R4	Sep 1, 2009	Requesting Dedicated Time on ASA HPC Platforms
ASAP20	R4	Mar 15, 2005	ASA Acceptable Use Policy
ASAP21	R1	Mar 9, 2005	CIPA Content Filtering Policy
ASAP22	R0	In Process	ASA Disaster Recovery

Policy: ASAP03 - R5

Title: **ASA Hours of Operation**
Date: 01/01/94:Revised 3/8/05
Distribution: All ASA Clients

A. OVERVIEW

This policy defines the scheduled hours of operation of the George C. Wallace Supercomputer Center (GCWSC), the Alabama Research and Education Network Operations Center, the ASA Network Office, and the ASA Business Office.

B. THE ASA BUSINESS OFFICE

Alabama Supercomputer Authority
Center for Commerce
401 Adams Avenue
Suite 758
Montgomery, Alabama 36130

(334) 242-0100
(334) 242-0637 (fax)

The ASA office is opened Monday - Friday, 0800-1700 (closed on all State holidays).

C. THE AREN NETWORK OPERATIONS CENTER

AREN Network Operation Center
George C. Wallace Supercomputer Center
686 Discovery Drive
Huntsville, Alabama 35806
(256) 971-7448
1-800-276-0670
helpdesk@asc.edu

The Network Operations Center (NOC) for the Alabama Research and Education Network is manned, 24 hours per day, 7 days per week, 365 days per year.

Planned network outages will be posted on the ASA web site www.asc.edu at least four (4) days in advance.

Network outages required by unexpected hardware or software problems will be posted as soon as possible to give users as much advance notice as possible.

The George C. Wallace Supercomputer Center is open to users (and visitors by appointment) Monday through Friday, 0800 to 1700. Off-hour access to users is available, by prior arrangement.

These schedules relate to the functioning of all equipment and services that provide network connectivity to ASA clients.

D. THE ASA NETWORK OFFICE GEORGE C. WALLACE SUPERCOMPUTER CENTER

686 Discovery Drive
Huntsville, Alabama 35806
(256) 971-7404
(256) 971-7473 (fax)
asainfo@asc.edu

The ASA Network office is opened Monday - Friday, 0800-1700 (closed on all State holidays).

Policy: ASAP04 - R7

Title: **HPC Charges for Commercial Clients**
Date: 01/01/94:Revised 9/1/2009
Distribution: All ASA Commercial Clients

A. OVERVIEW.

This policy establishes guidelines for charging for processing on ASA's High performance Computing (HPC) platforms. Currently, ASA maintains two (2) HPC platforms, an SGI 350 and a Dense Memory Cluster system. The method is based upon a cost recovery charge per CPU hour.

B. HPC SYSTEM CHARGES.

- There is a minimum purchase of 2000 CPU hours upon which an account is established.
- Current CPU/hour rates are on the ASA web site www.asc.edu under HPC services.
- CPU hours on each platform are assigned a value. ASA reserves the right to change the dollar value per CPU hour at any time (will not effect current contracts).
- Jobs that are submitted without a sufficient CPU hour balance will be automatically suspended until additional CPU hours are purchased.
- Special services will be addressed on a case-by-case basis.
- AREN access charges may or may not apply
- All HPC contracts expire within one year unless changed by mutual agreement

C. SOFTWARE CHARGES

Clients that request use of software provided by ASA must abide by the software license agreements for those software packages. Most of the software packages have the license terms included in the online documentation.

Most of the software provided by ASA is licensed for academic use. Commercial users or university researchers doing work under contract to commercial firms must contact the Alabama Supercomputer Authority to see if this usage is allowed by the current license.

D. ADDITIONAL CHARGES

Additional charges may be incurred if additional disk space is required above the assigned initial space.

Charges for additional consultation may be deemed appropriate when time is expended above and beyond the normal is experienced.

All users are responsible for the safe disposition (backup) of their own program and data files.

Policy: ASAP05 - R8

Title: **HPC Client File Storage Capacities**
Date: 08/16/95:Revised 9/1/2009
Distribution: All ASA Users

A. OVERVIEW.

The policy relates to file storage capacities for clients using ASA High Performance Computing (HPC) facilities.

The client is responsible for providing his/her own backup by what ever method is appropriate (e.g. ftp to local node, etc.).

File backup is performed at the central site only for unscheduled outages and disaster recovery.

B. CLIENT PERMANENT FILE STORAGE.

Each client account is assigned a /home directory in a file system for a particular HPC platform. Client files stored within this directory are termed permanent client storage. Files stored here remain until removed by the client or by the PERMANENT STORAGE PURGE (see C, below).

File storage limits in the **/home** directory are set at 1 GB (1 gigabyte) for the particular HPC platform and a total of 2 GB (2 Gigabytes) across all systems **excluding** scratch disk.

If a client requires more file space (up to 40 GB) the client must request additional space from ASA's HPC Computational Specialist at **hpc@asc.edu** .

Clients requiring file space over and above 20 GB must make a request directly through ASA's Chief Fiscal Officer, there will be a charge involved.

C. PERMANENT STORAGE PURGE.

When the **/home** limit is reached, clients will get a warning message indicating that disk usage must be reduced below their assigned limit within in 7 days. After 7 days without reducing usage, any process attempting to create additional files will be killed.

D. CLIENT TEMPORARY DISK STORAGE.

To accommodate jobs that require large amounts of temporary scratch space, clients will have access to the **/scratch** directory. This directory provides 1.1 TB (1.1 Terabytes) of temporary storage. **Data left in this directory after calculation completes will be purged after seven (7) days with no backup.**

E. CLIENT DISK BACKUP.

To provide recovery in the event of a storage hardware failure, files on HPC file storage facilities will be backed up periodically to a RAID server. These backups are not available for recovery of individual files deleted by a client. No backup is maintained for files on **/scratch** .

In the event that client **/home** directory files are destroyed by hardware or software system failure, **/home**, directory files will be restored from the latest backup available. Files on **/scratch** cannot be restored. A news notification will be posted.

F. ACCOUNT EXPIRATION.

Accounts are established on an annual basis. **Home directory files are maintained up to one year after account expiration.**

Policy: ASAP06 - R5

Title: **HPC Proprietary Program Charges**
Date: 01/01/94:Revised 3/24/2005
Distribution: All ASA HPC Clients

A. OVERVIEW.

This policy discusses charges to ASA High Performance Computation (HPC) clients for proprietary application programs available on ASA HPC platforms.

B. CHARGING.

ASA Policy ASAP04 specifies the charging policy for ASA HPC platforms. This policy documents only the ADDITIONAL charges each user is to pay for using certain proprietary programs. These charges are specified by the vendor of each particular program, and are based on program usage.

- Proprietary charges are in ADDITION to the normal processing charges for HPC time.
- Proprietary charges are always billed by the Alabama Supercomputer Authority, unless the user has received an exemption.

Policy: ASAP08 - R6

Title: **Establishing an ASA HPC User Account**
Date: 05/01/94: Revised 04/21/2005
Distribution: ASA HPC Clients

A. OVERVIEW.

This policy describes how accounts are established for High Performance Computation clients.

Parties interested in establishing accounts should contact the ASA HPC Computational Specialist by email hpc@asc.edu or telephone (256-971-7434).

B. ACCOUNT CATEGORIES.

The Alabama Supercomputer Authority has four (4) categories of accounts:

- Academic/Education - Accounts allocated to state academic education without charge

- Academic/Fee-based - A state academic researcher working off a grant which pays for HPC use (sponsored research), state academic researcher working on an Industrial/Government sponsored project, or a private institution researcher are charged a fee for using ASA HPC resources.

- Collaborative Client - A state researcher who has entered into an agreement with ASA to provide and utilize resources to be located at ASC.

- Commercial - Paid use from Industry or Government Agency. Charges for these account categories are outlined in policy ASAP04 and current rates are displayed on ASA's website www.asc.edu.

C. ACADEMIC/COMMERCIAL ACCOUNTS

Potential clients should contact the ASA HPC Computational Specialist hpc@asc.edu or ASA directly information@asc.edu in order to initiate a "Request for Services Contract".

D. ACCOUNT EXPIRATION.

Accounts will expire one year from when the account was established or last renewed. Accounts will be reactivated as soon as new account requests are received and approved by ASA. Users with queued jobs that will run past this deadline can

complete the ASA HPC Grant Request form up to one week prior to the account expiration date. Other users will be required to complete this form after the expiration date. The form can be completed online at www.asc.edu/cgi-bin/account_request.cgi.

Accounts may be closed at anytime based on a request from an academic or a commercial account representative prior to expiration. When an account expires, the account is deleted from all platforms. Files are held for 30 days and then deleted unless a special request is made in writing prior to the expiration date.

E. ACCOUNT PASSWORDS.

Initial passwords are assigned by the Account Administrator at ASC. Users may set their own password.

Clients who have lost passwords, or suspected security breeches, should contact the ASA HPC Computational Specialist hpc@asc.edu or 256-971-7434 as soon as possible.

Policy: ASAP13 - R2

Title: **ASA Configuration Change Request**
Date: 01/01/94:Revised 4/18/2005
Distribution: ASA Staff

A. GENERAL

The purpose of this policy is to establish guidelines for the ASA Configuration Change Request (CCR). The purpose of the CCR is to provide a documented procedure to effect changes and enhancements to the services provided by the Alabama Supercomputer Authority by ASA's Professional Services Contractor not covered in the current contract. The requested changes should be submitted and labeled "Configuration Change Request" (CCR).

B. SUBMISSION.

A CCR can be submitted by ASA or ASA's professional services contractor to the ASA COO. Copies (in electronic form or paper) will be passed around for discussion as necessary.

C. APPROVAL.

The ASA COO will then make any additional comments and send it on to the ASA CEO for approval or disapproval. **Since the request may mean the expenditure of funds, the responsible party (originator) is not to take any action until given direction/approval by the ASA CEO.**

POLICY: ASAP14 - R6

Title: **Access to the George C. Wallace Supercomputer Center**
DATE: 05/01/94:Revised 3/24/05
Distribution: ASA Staff, Contractor Personnel, and ASA Clients

A. OVERVIEW

This policy defines the procedure governing physical access to the George C. Wallace Supercomputer Center (GCWSC).

B. ACCESS CONTROL PROCEDURE

Access to the GCWSC is electronically controlled 7 days per week, 24 hours per day. Electronic "keys" are be assigned to necessary and essential personnel based on a category system (see below). Each key is electronically encoded for facility access and tracking purposes. The ASA facilities manager is responsible for the issuance and accounting for all keys. The ASA facilities manager is also responsible for maintaining the electronic access control and camera system.

C. GCWSC ACCESS LIST

The access list for GCWSC is structured into three sections or categories:

1. Category 1 - Permanent Staff

Category 1 is defined as Permanent staff: ASA staff personnel and ASA's contractor personnel permanently assigned to the GCWSC. Assignments to this list will be directed by the ASA CEO and COO and controlled by the ASA facilities manager.

Permanent staff will have unrestricted access to the facility Monday through Friday during business hours. Access after close of business on weekdays and on weekends will require signing in and out on the visitor log located in the front lobby desk. Signing the visitor log is not required between the hours of 0700 to 1700 on weekdays.

Internal access to individual offices will be controlled through mechanical key assignments determined by ASA and ASA contractor management. The ASA facilities manager will assign and account for mechanical keys.

Category 1 personnel are authorized to escort guests.

Category 1 personnel should make every effort to inform their management and operations helpdesk personnel in advance of guest arrivals.

2. Category 2 - Approved on-site Users

Category 2 is defined as on-site or contract users who may be assigned space at GCWSC. Category 2 users must be approved by ASA and assigned space at GCWSC. Application for office or computer room space consists of correspondence to ASA management indicating purpose and duration of requirement. Category 2 users will be required to sign the visitors log at all times. Personal identification may be requested. Movement within the facility is limited to the assigned office and public areas on the first floor.

These users are not authorized to escort guests.

3. Category 3 - Other building visitors

a. Emergency Personnel

Emergency personnel shall have all rights of access necessary to perform their duties. If required, network operations personnel or the ASA facilities manager will escort or arrange escort in compliance with emergency procedures.

b. Tours and Training Sessions

Tours of the facility and training sessions shall be scheduled in advance with ASA. ASA will be advised of time, group identity, and expected size in advance. These visitors shall not have access to contractor offices or the ASA offices. Tours of the computer room must be scheduled in advance and approved by ASA and ASA's contractor management. Groups must be escorted by Category 1 staff at all times in areas other than the public areas in the front of the first floor.

c. Vendors and Maintenance Personnel

All vendor support personnel, building support personnel and maintenance personnel, must sign the visitor log and be escorted or confirmed by Category 1 personnel.

All other visitors to the GCWSC will be required to sign the visitor log. They must be announced to their point of contact who will then give operations personnel verbal acceptance and instructions.

Only Category 1 management may grant exceptions to this procedure; such exceptions are to be documented in the visitor log and acknowledged by the member of management granting the exception and initialing the entry.

D. OFFICE ASSIGNMENTS

Office assignment will be the responsibility of the ASA CEO. A copy of the approved space application indicating room number and length of stay will be provided to ASA. The ASA facilities manager is responsible for providing access to the assigned office space and whatever communications they might need.

POLICY: ASAP17-R4

TITLE:**Requesting Dedicated Time on ASA HPC Platforms**
DATE: 01/01/94:Revised 9/1/2009
Distribution: All ASA Clients

A. OVERVIEW.

This policy defines the process for requesting dedicated time on the Alabama Supercomputer Authority (ASA) HPC platforms, the associated charges for use, and the time available for dedicated machine use.

Dedicated time means that one client has the sole use of the complete platform for a specific period of time and is established for commercial clients.

B. REQUESTING DEDICATED MACHINE TIME.

Requests for dedicated machine time should be directed to the HPC Computational Specialist at **HPC@asc.edu**. The reason for dedicated time must be specified. A description of what activities will be performed and what assistance is needed.

C. CHARGES FOR DEDICATED TIME.

Charges for dedicated time will be negotiated with ASA. Dedicated time will include one hour of preparation time to configure the platform for dedicated use. If a client signs up for dedicated time and fails to use it, the user will be charged for one hour of use at the rate negotiated.

D. AVAILABILITY OF DEDICATED MACHINE TIME.

The DMC Cluster and the SGI ALTIX 350 are available for dedicated time as negotiated.

Policy: ASAP20 - R4

Title: **ASA Acceptable Use Policy**
Date: 01/01/94:Revised 3/15/2005
Distribution: All ASA Clients

1. OVERVIEW

The Alabama Supercomputer Authority, a state non-profit corporation (1975 Alabama Code §§ 41-10-390 to 41-10-406), administers the Alabama Research and Education Network (AREN), a statewide education network, and operates the George C. Wallace Supercomputer Center. The purpose of this policy is to provide a definition for acceptable use by authorized clients of ASA services and to indicate recommended action if the policy is violated. In those cases when information is transmitted across regional networks or Internet, ASA clients are advised that acceptable use policies of those networks apply and may limit access.

2. ASA ACCEPTABLE USE POLICY

- ASA services are for the use of individuals legitimately affiliated with ASA clients, to facilitate the exchange of information consistent with the academic, educational, and research purposes of its member organizations.

- It is not acceptable to use ASA services for illegal purposes.

- It is not acceptable to use ASA services to transmit threatening, obscene, or harassing materials.

- Access to the INTERNET is provided through agreements with INTERNET Service Providers. These agreements allow ASA to grant access to the INTERNET to government, education, and industrial clients. Charges may be assessed by ASA to facilitate network and Internet connectivity.

- The reselling of the ASA services is prohibited unless approved in writing by ASA.

- It is not acceptable for ASA clients to interfere with or disrupt network users, services or equipment (intentionally and unintentionally) through the use of ASA services. Disruptions include, but are not limited to, unsolicited advertising, propagation of computer worms or viruses, and using AREN to make unauthorized entries to any other computers accessible via the network. ASA clients are responsible for maintaining an acceptable security status on all assets connected to AREN.

- ASA clients must respect the legal protection applied to programs, data, photographs, music, text documents and other material as provided by copyright, trademark, patent, licensure and other proprietary rights mechanisms.

- Authorized ASA clients are required to protect their attached computers, servers, and networks from computer viruses or worms that cause a systemic disruption to ASA and its INTERNET services.

- Authorized ASA clients are required to provide current and accurate client contact information to enable ASA representatives to have ready access for resolution of problems.

- Information and resources accessible through ASA services are private to the individuals and organizations which own or hold rights to those resources and information unless specifically stated otherwise by the owners or holders of rights. It is therefore not acceptable for an individual to use ASA services to access information or resources unless permission is granted by the owners or holders of rights to those resources or information.

The intent of this policy is to identify certain types of uses that are not appropriate, but this policy does not necessarily enumerate all possible inappropriate uses. Using the guidelines given above, ASA may at any time make a determination that a particular use is not appropriate.

C. VIOLATION OF POLICY

All organizations authorized to access ASA services are responsible for informing their users of this acceptable use policy. All users of ASA services are required to follow the acceptable use guidelines, both in letter and spirit.

ASA reserves the right to monitor and review all traffic and data on ASA provided services for potential violations of this policy. Violations of policy that are not promptly remedied by individuals or ASA clients may result in termination of access to ASA services. ASA will only release sensitive, confidential or personally identifiable information to third parties when required by law, or when in ASA's judgment, release is required to prevent serious injury or harm that could result from violation of this policy.

Final authority for the determination of violation of the ASA Acceptable Use Policy and subsequent penalty rests with the ASA Board of Directors.

It is the responsibility of ASA clients to contact ASA, in writing, regarding questions of interpretation. Until such issues are resolved, questionable use should be considered "not acceptable".

Policy: ASAP21-R1

Title: **CIPA Content Filtering Policy**
Date: 5/8/2002 Revised:3/9/2005
Distribution: All ASA Clients

A. OVERVIEW

The Alabama Research and Education Network (AREN) is a statewide education network administered by the Alabama Supercomputer Authority (ASA), a state non-profit corporation. ASA provides connectivity and Internet access to many K-12 public school systems and public libraries within the state of Alabama. By July 1, 2002 all public school systems and public libraries that receive E-rate funds were required to conform to the **Children's Internet Protection Act (CIPA)**, an act signed into law by Congress on December 21, 2001.

"No public school or public library may receive discounts unless it certifies that it is enforcing a policy of Internet safety that includes the use of filtering or blocking technology". - (CIPA 12/21/2001)

ASA utilizes a vendor-supplied content filtering service that is available to "all" AREN clients. This service may be used as the "Technology Protection Measure" referenced in CIPA.

"A Technology Protection Measure is a specific technology that blocks or filters Internet access. It must protect against access by adults and minors to visual depictions that are obscene, child pornography, or with respect to use of computers with Internet access by minors harmful to minors. It may be disabled for adults engaged in bona fide research or other lawful purposes." - (CIPA 12/21/2001)

The use of the ASA content filter does not make a public school system or a public library fully "CIPA compliant". Additional steps (i.e. design and implement an "Internet Safety Policy", provide public notice and hold a public hearing etc.) must be taken by the public school or the public library as outlined within CIPA.

B. ASA CONTENT FILTERING, TECHNOLOGY PROTECTION MEASURE

1. The ASA's current content filtering solution refers to the Technology Protection Measure in use by ASA for AREN Internet web access at any point in time.

2. ASA must have in its possession at the time of activation a valid "ASA Agreement for Activation of CIPA Content Filtering" and a SLD Form 479 from the respective public school system or public library.

3. The ASA content filtering service provides for filtering according to various profiles available to clients (i.e. public libraries, elementary schools, high schools, etc.). The default profile will block Internet web sites classified under the current CIPA requirements:

"block or filter Internet access for both minors and adults to certain visual depictions. These include visual depictions that are (1) obscene, or (2) child pornography, or (3) with respect to use of computers with Internet access by minors, material that is harmful to minors" - (CIPA 12/21/2001)

C. FILTERING / BLOCKING FLEXIBILITY

1. The filtering software vendor includes provisions for reviewing and classifying all web sites that have not been classified. If a filtered client reaches an obviously inappropriate site, the vendor will automatically update to block the site within 72 hours. This automatic update occurs because lists of all unclassified sites are sent to the vendor for classification and review on a daily basis.

2. In order to expedite filtering of unclassified sites, a manual request for filtering change can be requested.

3. Requests for filtering changes must come directly from the client's designated filtering coordinator. The filtering coordinator would normally be the school system's technology coordinator or the library's network manager but may be any person designated by the school superintendent or library administrator on the ASA Agreement for Activation of CIPA Content Filtering.

4. No action will be taken by ASA personnel to change filtering profiles, unblock sites, or block sites until a client's filtering coordinator has requested such action via the ASA Content Management System (CMS).

5. The ASA CMS can be accessed by designated filtering coordinators via the following URL: https://intranet.asc.edu/client_graphing/client_login.php

6. Requests for site re-classification (blocking open sites or unblocking filtered sites) made through ASA CMS will be automatically forwarded to our filtering vendor for review. The review and reclassification process may still take up to 72 hours but should occur faster than the automatic classification process.

7. Immediate blocking or unblocking of filtered content may be requested, but such requests are discouraged. If immediate action is a normal requirement of a client, that client should use local resources to accomplish immediate blocks. These

local resources could include proxy server or router based content filters that are managed locally by the client.

8. Requests for immediate blocking or unblocking should be made through the ASA CMS and will be serviced on a first-come-first-served basis. All requests will be serviced within two school hours. If ASA personnel have taken no action within two hours, the filtering coordinator may contact the ASA helpdesk by phone with their ticket number and ask that the issue be expedited. Immediate blocking or unblocking requests made during the weekend will be addressed before noon on the following workday.

D. ADDITIONAL INFORMATION

1. ***ASA CIPA Content Filtering FAQ*** will provide additional information about how content filtering will be implemented.

2. The AREN client who wants to utilize the provided content filtering must sign an ***ASA Agreement for Activation of CIPA Content Filtering*** and a signed copy ***SLD Form 479***.

3. The ***ASA Agreement for Activation of CIPA Content Filtering*** and signed ***SLD Form 479*** should then be mailed or faxed to:

Alabama Supercomputer Authority
Center for Commerce
401 Adams Avenue
Suite 758
Montgomery, AL 36130
(334) 242-0100
(fax) 334-242-0637

E. DISCLAIMERS AND LIABILITY

1. ASA's content filtering solution is designed to block inappropriate web sites as defined by current CIPA guidelines. This solution **does not** filter inappropriate content contained in email or chat rooms.

2. ASA will strive to provide a content filtering service that is useful and current and will attempt to address problems where the filtering service is found to be deficient.

3. ASA assumes no liability in the event that the content filtering service is not 100% effective. ASA also assumes no responsibility for the currency of the filter or the content provided through it.

4. All ASA clients are responsible to abide by the ASA Acceptable Use Policy (ASAP20).

5. ASA clients may add additional content filtering to their own networks in addition to the one provided by ASA.

6. ASA clients may discontinue their use of ASA's content filtering service with written notification and with proof of use of an alternative content filtering technology protection measure.

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 accessing the directory **/opt/asn/doc**

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.

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.