Altair®

PBS Professional™ 7

User Guide

UNIX®, LINUX® and Windows®

Portable Batch System™ User Guide

PBS-3BA01: Altair® PBS Professional™ 7, Updated: April 23, 2005
Edited by Anne Urban

Altair Grid Technologies is a subsidiary of Altair Engineering, Inc. For more information, and for product sales and technical support, contact Altair at:

URL: www.altair.com www.pbspro.com
Email: sales@pbspro.com, support@pbspro.com

Location	Telephone	e-mail
North America	+1 248 614 2425	pbssupport@altair.com
China	+86 (0)21 5393 0011	support@altair.com.cn
France	+33 (0)1 4133 0990	francesupport@altair.com
Germany	+49 (0)7031 6208 22	support@altair.de
India	+91 80 658 8540 +91 80 658 8542	pbs-support@india.altair.com
Italy	+39 832 315573 +39 800 905595	support@altairtorino.it
Japan	+81 3 5396 1341	support@altairjp.co.jp
Korea	+82 31 728 8600	support@altair.co.kr
Scandinavia	+46 (0)46 286 2050	support@altair.se
UK	+44 (0)1327 810 700	support@uk.altair.com

For online documentation purchases, visit: store.pbspro.com

Table of Contents

Preface

Intended Audience

PBS Professional is the professional workload management system from Altair that provides a unified queuing and job management interface to a set of computing resources. This document provides the user with the information required to use the Portable Batch System (PBS), including creating, submitting, and manipulating batch jobs; querying status of jobs, queues, and systems; and otherwise making effective use of the computer resources under the control of PBS.

Related Documents

The following publications contain information that may also be useful to the user of PBS:

PBS-3BQ01 **PBS Professional Quick Start Guide**: offers a short overview of the installation and use of PBS Professional.

PBS-3BA01 **PBS Professional Administrator Guide**: provides the system administrator with information required to install, configure, and manage PBS, as well as a thorough discussion of how the various components of PBS interoperate.

PBS-3BE01 **PBS Professional External Reference Specification**: discusses in detail the PBS application programming interface (API), security within PBS, and inter-daemon/service communication.

Ordering Software and Publications

To order additional copies of this and other PBS publications, or to purchase additional software licenses, contact an authorized reseller, or the PBS Sales Department. Contact information is included on the copyright page of this document.

Document Conventions

PBS documentation uses the following typographic conventions.

abbreviation	If a PBS command can be abbreviated (such as sub-commands to qmgr) the shortest acceptable abbreviation is underlined.
`command`	This fixed width font is used to denote literal commands, filenames, error messages, and program output.
`input`	Literal user input is shown in this bold fixed-width font.
`manpage`(x)	Following UNIX tradition, manual page references include the corresponding section number in parentheses appended to the man page name.
terms	Words or terms being defined, as well as variable names, are in italics.

Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community is most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of *Pittsburgh Supercomputing Center*; and Dirk Grunwald of *University of Colorado, Boulder*. The ports of PBS to the Cray T3e and the IBM SP SMP were funded by *DoD USAERDC*; the port of PBS to the Cray SV1 was funded by *DoD MSIC*.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the *Space Sciences Center* of the *University of New Hampshire* was the first beta tester. Wendy Lin of *Purdue University* was the second beta tester and holds the honor of submitting more problem reports than anyone else outside of NASA.

Chapter 1
Introduction

This book, the **User Guide** to the Portable Batch System, Professional Edition (PBS Professional) is intended as your knowledgeable companion to the PBS Professional software. The information herein pertains to PBS in general, with specific information for PBS Professional 7.

1.1 Book organization

This book is organized into 10 chapters, plus two appendices. Depending on your intended use of PBS, some chapters will be critical to you, and others may be safely skipped.

Chapter 1 gives an overview of this book, PBS, and the PBS team.

Chapter 2 discusses the various components of PBS and how they interact, followed by definitions of terms used in PBS and in distributed workload management.

Chapter 3 introduces PBS, describing both user interfaces and suggested settings to the user's environment.

Chapter 4 describes the structure and components of a PBS job, and explains how to create and submit a PBS job.

Chapter 5	introduces the `xpbs` graphical user interface, and shows how to submit a PBS job using `xpbs`.
Chapter 6	describes how to check status of a job, and request status of queues, nodes, systems, or PBS Servers.
Chapter 7	discusses commonly used commands and features of PBS, and explains how to use each one.
Chapter 8	describes and explains how to use the more advanced features of PBS.
Chapter 9	describes and explains the job array features in PBS.
Chapter 10	explains how PBS interacts with multi-node and parallel applications, and illustrates how to run such applications under PBS.
Appendix A	provides a quick reference summary of PBS environment variables.
Appendix B	includes information for converting from NQS/NQE to PBS.

1.2 What is PBS Professional?

PBS Professional is the professional version of the Portable Batch System (PBS), a flexible workload management system, originally developed to manage aerospace computing resources at NASA. PBS has since become the leader in supercomputer workload management and the *de facto* standard on Linux clusters.

Today, growing enterprises often support hundreds of users running thousands of jobs across different types of machines in different geographical locations. In this distributed heterogeneous environment, it can be extremely difficult for administrators to collect detailed, accurate usage data, or to set system-wide resource priorities. As a result, many computing resources are left under-utilized, while other are over-utilized. At the same time, users are confronted with an ever expanding array of operating systems and platforms. Each year, scientists, engineers, designers, and analysts must waste countless hours learning the nuances of different computing environments, rather than being able to focus on their core priorities. PBS Professional addresses these problems for computing-intensive industries such as science, engineering, finance, and entertainment.

Now you can use the power of PBS Professional to better control your computing resources. This allows you to unlock the potential in the valuable assets you already have, while at the same time, reducing dependency on system administrators and operators, freeing them to focus on other actives. PBS Professional can also help you effectively manage growth by tracking real usage levels across your systems and enhancing utilization of future purchases.

1.3 History of PBS

In the past, UNIX systems were used in a completely interactive manner. Background jobs were just processes with their input disconnected from the terminal. However, as UNIX moved onto larger and larger machines, the need to be able to schedule tasks based on available resources increased in importance. The advent of networked compute servers, smaller general systems, and workstations led to the requirement of a networked batch scheduling capability. The first such UNIX-based system was the Network Queueing System (NQS) funded by NASA Ames Research Center in 1986. NQS quickly became the *de facto* standard for batch queueing.

Over time, distributed parallel systems began to emerge, and NQS was inadequate to handle the complex scheduling requirements presented by such systems. In addition, computer system managers wanted greater control over their compute resources, and users wanted a single interface to the systems. In the early 1990's NASA needed a solution to this problem, but found nothing on the market that adequately addressed their needs. So NASA led an international effort to gather requirements for a next-generation resource management system. The requirements and functional specification were later adopted as an IEEE POSIX standard (1003.2d). Next, NASA funded the development of a new resource management system compliant with the standard. Thus the Portable Batch System (PBS) was born.

PBS was quickly adopted on distributed parallel systems and replaced NQS on traditional supercomputers and server systems. Eventually the entire industry evolved toward distributed parallel systems, taking the form of both special purpose and commodity clusters. Managers of such systems found that the capabilities of PBS mapped well onto cluster systems. (For information on converting from NQS to PBS, see Appendix B.)

The PBS story continued when MRJ-Veridian (the R&D contractor that developed PBS for NASA) released the Portable Batch System Professional Edition (PBS Pro), a commercial, enterprise-ready, workload management solution. Three years later, the MRJ-Ver-

idian PBS Products business unit was acquired by Altair Engineering, Inc. Altair set up the PBS Products unit as a subsidiary company named Altair Grid Technologies focused on PBS Professional and related Grid software.

1.4 About the PBS Team

The PBS Professional product is developed by the same team that originally designed PBS for NASA. In addition to the core engineering team, Altair Grid Technologies includes individuals who have supported PBS on computers around the world, including some of the largest supercomputers in existence. The staff includes internationally-recognized experts in resource-management and job-scheduling, supercomputer optimization, message-passing programming, parallel computation, and distributed high-performance computing. In addition, the PBS team includes co-architects of the NASA Metacenter (the first full-production geographically distributed meta-computing grid), co-architects of the Department of Defense MetaQueueing (prototype Grid) Project, co-architects of the NASA Information Power Grid, and co-chair of the Global Grid Forum's Scheduling Group.

1.5 About Altair Engineering

Through engineering, consulting and high performance computing technologies, Altair Engineering increases innovation for more than 1,500 clients around the globe. Founded in 1985, Altair's unparalleled knowledge and expertise in product development and manufacturing extend throughout North America, Europe and Asia. Altair specializes in the development of high-end, open CAE software solutions for modeling, visualization, optimization and process automation.

1.6 Why Use PBS?

PBS Professional provides many features and benefits to both the computer system user and to companies as a whole. A few of the more important features are listed below to give the reader both an indication of the power of PBS, and an overview of the material that will be covered in later chapters in this book.

Enterprise-wide Resource Sharing provides transparent job scheduling on any PBS system by any authorized user. Jobs can be submitted from any client system both local and remote, crossing domains where needed.

Multiple User Interfaces provides a graphical user interface for submitting batch and interactive jobs; querying job, queue, and system status; and monitoring job progress. PBS also provides a traditional command line interface.

Security and Access Control Lists permit the administrator to allow or deny access to PBS systems on the basis of username, group, host, and/or network domain.
Job Accounting offers detailed logs of system activities for charge-back or usage analysis per user, per group, per project, and per compute host.

Automatic File Staging provides users with the ability to specify any files that need to be copied onto the execution host before the job runs, and any that need to be copied off after the job completes. The job will be scheduled to run only after the required files have been successfully transferred.

Parallel Job Support works with parallel programming libraries such as MPI, PVM and HPF. Applications can be scheduled to run within a single multi-processor computer or across multiple systems.

System Monitoring includes a graphical user interface for system monitoring. Displays node status, job placement, and resource utilization information for both stand-alone systems and clusters.

Job-Interdependency enables the user to define a wide range of inter-dependencies between jobs. Such dependencies include execution order, synchronization, and execution conditioned on the success or failure of another specific job (or set of jobs).

Computational Grid Support provides an enabling technology for meta-computing and computational grids, including support for the Globus Grid Toolkit.

Comprehensive API includes a complete Application Programming Interface (API) for sites who desire to integrate PBS with other applications, or who wish to support unique job scheduling requirements.

Automatic Load-Leveling provides numerous ways to distribute the workload across a cluster of machines, based on hardware configuration, resource availability, keyboard activity, and local scheduling policy.

Distributed Clustering allows customers to utilize physically distributed systems and clusters, even across wide-area networks.

Common User Environment offers users a common view of the job submission, job querying, system status, and job tracking over all systems.

Cross-System Scheduling ensures that jobs do not have to be targeted to a specific computer system. Users may submit their job, and have it run on the first available system that meets their resource requirements.

Job Priority allows users the ability to specify the priority of their jobs; defaults can be provided at both the queue and system level.

Username Mapping provides support for mapping user account names on one system to the appropriate name on remote server systems. This allows PBS to fully function in environments where users do not have a consistent username across all hosts.

Fully Configurable. PBS was designed to be easily tailored to meet the needs of different sites. Much of this flexibility is due to the unique design of the scheduler module which permits significant customization.

Broad Platform Availability is achieved through support of Windows 2000 and XP, and every major version of UNIX and Linux, from workstations and servers to supercomputers. New platforms are being supported with each new release.

System Integration allows PBS to take advantage of vendor-specific enhancements on different systems (such as supporting cpusets on SGI systems).

Job Arrays are a mechanism for containerizing related work, making it possible to submit, query, modify and display a set of jobs as a single unit.

Chapter 2
Concepts and Terms

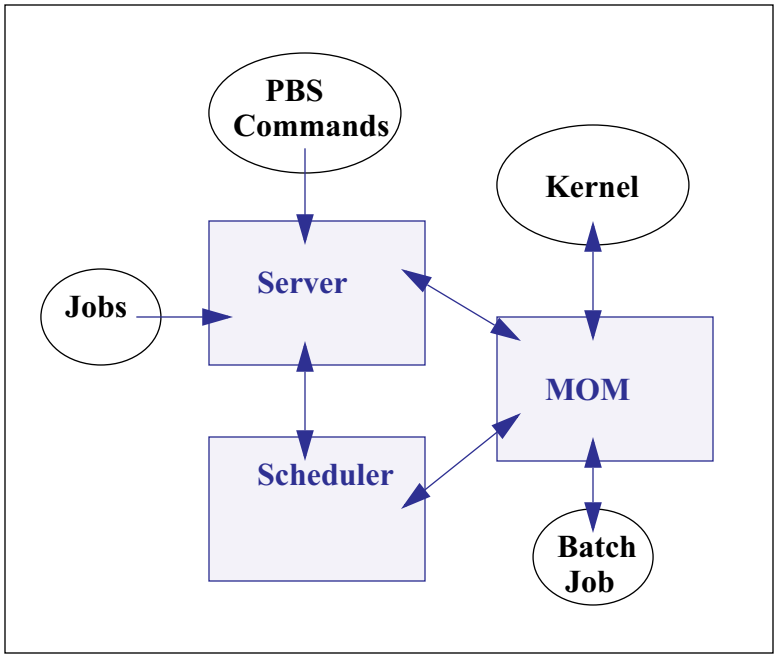
PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload management solutions like PBS Professional include the features of traditional batch queueing but offer greater flexibility and control than first generation batch systems (such as NQS).

Workload management systems have three primary roles:

Queuing	The collecting together of work or tasks to be run on a computer. Users submit tasks or "jobs" to the resource management system where they are queued up until the system is ready to run them.
Scheduling	The process of selecting which jobs to run, when, and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize efficient use of resources (both computer time and people time).
Monitoring	The act of tracking and reserving system resources and enforcing usage policy. This includes both software enforcement of usage limits and user or administrator monitoring of scheduling policies to see how well they are meeting stated goals.

2.1 PBS Components

PBS consist of two major component types: user-level commands and system daemons/services. A brief description of each is given here to help you understand how the pieces fit together, and how they affect you.



Commands PBS supplies both command line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These *client commands* can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

There are three command classifications: user commands, which any authorized user can use, operator commands, and manager (or administrator) commands. Operator and manager commands which require specific access privileges are discussed in the **PBS Professional Administrator Guide**.

Server The *Job Server* daemon/service is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server*. All commands and the other

daemons/services communicate with the Server via an *Internet Protocol* (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, and running the job. Normally, there is one Server managing a given set of resources. However if the Server Failover feature is enabled, there will be two Servers.

Job Executor (MOM)

The *Job Executor* or *MOM* is the daemon/service which actually places the job into execution. This process, *pbs_mom,* is informally called *MOM* as it is the mother of all executing jobs. (MOM is a reverse-engineered acronym that stands for Machine Oriented Mini-server.) MOM places a job into execution when it receives a copy of the job from a Server. MOM creates a new session that is as identical to a user login session as is possible. (For example under UNIX, if the user's login shell is `csh`, then MOM creates a session in which `.login` is run as well as `.cshrc`.) MOM also has the responsibility for returning the job's output to the user when directed to do so by the Server. One MOM runs on each computer which will execute PBS jobs.

A special version of MOM, called the *Globus MOM*, is available if it is enabled during the installation of PBS. It handles submission of jobs to the Globus environment. Globus is a software infrastructure that integrates geographically distributed computational and information resources. Globus is discussed in more detail in the **PBS Professional Administrator Guide**. (To find out if Globus support is enabled at your site, contact your PBS system administrator.)

Scheduler

The *Job Scheduler* daemon/service, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The Scheduler communicates with the various MOMs to query the state of system resources and with the Server for availability of jobs to execute. The interface to the Server is through the same API as used by the client commands. Note that the Scheduler interfaces with the Server with the same privilege as the PBS manager.

2.2 Defining PBS Concepts and Terms

The following section defines important terms and concepts of PBS. The reader should review these definitions before beginning the planning process prior to installation of PBS. The terms are defined in an order that best allows the definitions to build on previous terms.

Node A *node* to PBS is a computer system with a single *operating system* (OS) image, a unified virtual memory space, one or more CPUs and one or more IP addresses. Frequently, the term *execution host* is used for node. A computer such as the SGI Origin 3000, which contains multiple CPUs running under a single OS, is one node. Systems like the IBM SP and Linux clusters, which contain separate computational units each with their own OS, are collections of nodes. Nodes can be defined as either *cluster nodes* or *timeshared nodes*, as discussed below. (See also *virtual processors*.)

Cluster Node A node whose purpose is geared toward running multi-node or parallel jobs is called a *cluster node*. If a cluster node has more than one virtual processor, the VPs may be assigned to different jobs (*job-shared*) or used to satisfy the requirements of a single job (*exclusive*). This ability to temporally allocate the entire node to the exclusive use of a single job is important for some multi-node parallel applications. Note that PBS enforces a one-to-one allocation scheme of cluster node VPs ensuring that the VPs are not over-allocated or over-subscribed between multiple jobs. (See also *node* and *virtual processors*.)

Timeshared Node In contrast to cluster nodes are hosts that **always** service multiple jobs simultaneously, called *timeshared nodes*. Often the term *host* rather than node is used in conjunction with timeshared, as in *timeshared host.* A timeshared node will never be allocated exclusively or temporarily-shared. However, unlike cluster nodes, a timeshared node **can** be over-committed if the local policy specifies to do so. (See also *virtual processors*.)

Cluster This is any collection of nodes controlled by a single instance of PBS (i.e., by one PBS Server).

Exclusive VP An exclusive VP is one that is used by one and only one job at a time. A set of VPs is assigned exclusively to a job for the duration of that job. This is typically done to improve the performance of message-passing programs.

Temporarily-shared VP

A *temporarily-shared node* is one where one or more of its VPs are temporarily shared by jobs. If several jobs request multiple temporarily-shared nodes, some VPs may be allocated commonly to both jobs and some may be unique to one of the jobs. When a VP is allocated on a temporarily-shared basis, it remains so until all jobs using it are terminated. Then the VP may be re-allocated, either again for temporarily-shared use or for exclusive use.

If a host is defined as timeshared, it will never be allocated exclusively or temporarily-shared.

Load Balance

A policy wherein jobs are distributed across multiple timeshared hosts to even out the workload on each host. Being a policy, the distribution of jobs across execution hosts is solely a function of the Job Scheduler.

Queue

A *queue* is a named container for jobs within a Server. There are two types of queues defined by PBS, *routing* and *execution*. A *routing queue* is a queue used to move jobs to other queues including those that exist on different PBS Servers. Routing queues are similar to the NQS pipe queues. A job must reside in an *execution queue* to be eligible to run and remains in an execution queue during the time it is running. In spite of the name, jobs in a queue need not be processed in queue order (first-come first-served or *FIFO*).

Node Attribute

Nodes have attributes associated with them that provide control information. The attributes defined for nodes are: state, type (ntype), the list of jobs to which the node is allocated, properties, `max_running`, `max_user_run`, `max_group_run`, and both assigned and available resources ("`resources_assigned`" and "`resources_available`").

Node Property

A label for a node. A set of zero or more *properties* may be given to each node in order to have a means of grouping nodes for allocation. The property is nothing more than a string of alphanumeric characters (first character must be alphabetic) without meaning to PBS. The PBS administrator may assign to nodes whatever property names desired. Your PBS administrator will notify you of any locally defined properties.

Portable Batch System

PBS consists of one Job Server (`pbs_server`), one or more Job Schedulers (`pbs_sched`), and one or more execution servers

(`pbs_mom`). The PBS System can be set up to distribute the workload to one large timeshared system, multiple time-shared systems, a cluster of nodes to be used exclusively or temporarily-shared, or any combination of these.

Virtual Processor (VP) A node may be declared to consist of one or more *virtual processors (VPs)*. The term virtual is used because the number of VPs declared does not have to equal the number of real processors (CPUs) on the physical node. The default number of virtual processors on a node is the number of currently functioning physical processors; the PBS Manager can change the number of VPs as required by local policy. (See also *cluster node* and *timeshared node*.)

The remainder of this chapter provides additional terms, listed in alphabetical order.

Account An *account* is arbitrary character string, which may have meaning to one or more hosts in the batch system. Frequently, account is used by sites for accounting or charge-back purposes.

Administrator See Manager.

API PBS provides an *Application Programming Interface (API)* which is used by the commands to communicate with the Server. This API is described in the **PBS Professional External Reference Specification**. A site may make use of the API to implement new commands if so desired.

Attribute An *attribute* is an inherent characteristic of a parent object (Server, queue, job, or node). Typically, this is a data item whose value affects the operation or behavior of the object and can be set by the owner of the object. For example, the user can supply values for attributes of a job.

Batch or Batch Processing This refers to the capability of running jobs outside of the interactive login environment.

Complex A *complex* is a collection of hosts managed by one batch system. It may be made up of nodes that are allocated to only one job at a time or of nodes that have many jobs executing at once on each node or a combination of these two scenarios.

Destination This is the location within PBS where a job is sent for processing. A destination may be a single queue at a single Server or it may map into multiple possible locations, tried in turn until one accepts the job.

Destination Identifier This is a string that names the destination. It is composed of two parts and has the format queue@server where server is the name of a PBS Server and queue is the string identifying a queue on that Server.

Directive A means by which the user specifies to PBS the value of a variable such as number of cpus, the name of a job, etc. The default start of a directive is "#PBS". PBS directives either specify resource requirements or job control options. See page 30.

File Staging *File staging* is the movement of files between a specified location and the execution host. See "Stage In" and "Stage Out" below.

Group ID (GID) This unique number represents a specific group (see Group).

Group *Group* refers to collection of system users (see Users). A user must be a member of a group and may be a member of more than one. Membership in a group establishes one level of privilege, and is also often used to control or limit access to system resources.

Hold An artificial restriction which prevents a job from being selected for processing. There are three types of holds. One is applied by the job owner, another is applied by a PBS *Operator*, and a third applied by the system itself or the PBS *Manager*. (See also Operator and Manager in this glossary.)

Job or Batch Job The basic execution object managed by the batch subsystem. A job is a collection of related processes which is managed as a whole. A job can often be thought of as a shell script running a set of tasks.

Manager A *manager* is authorized to use all restricted capabilities of PBS. The Manager may act upon the Server, queues, or jobs. The Manager is also called the administrator.

Operator A person authorized to use some but not all of the restricted capabilities of PBS is an *operator*.

Owner The user who submitted a specific job to PBS.

PBS_HOME Refers to the path under which PBS was installed on the local system. Your local system administrator can provide the specific location.

POSIX This acronym refers to the various standards developed by the "Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society" under standard P1003.

Requeue The process of stopping a running (executing) job and putting it back into the queued ("Q") state. This includes placing the job as close as possible to its former position in that queue.

Rerunnable If a PBS job can be terminated and its execution restarted from the beginning without harmful side effects, the job is rerunnable.

Stage In This process refers to moving a file or files to the execution host prior to the PBS job beginning execution.

Stage Out This process refers to moving a file or files off of the execution host after the PBS job completes execution.

User Each system *user* is identified by a unique character string (the user name) and by a unique number (the user id).

Task *Task* is a POSIX session started by MOM on behalf of a job.

User ID (UID) Privilege to access system resources and services is typically established by the *user id*, which is a numeric identifier uniquely assigned to each user (see User).

Job Array A collection of jobs submitted under a single job id. These jobs can be modified, queried and displayed as a set.

Chapter 3

Getting Started With PBS

This chapter introduces the user to the Portable Batch System, PBS. It describes new user-level features in this release, explains the different user interfaces, introduces the concept of a PBS "job", and shows how to set up your environment for running batch jobs with PBS.

3.1 New Features in PBS Professional 7

The following is a list of new features and changes in PBS Professional release 7.0 which affect users. More detail is given in the indicated sections.

1. Job Arrays (See "Job Arrays" on page 131.)

2. Change to `mpirun` command (See "MPICH Jobs With PBS" on page 158.)

3. Integration with AIX MPI (See "MPI Jobs Using AIX, POE" on page 159.)

4. Support for CSA user job accounting on Altix machines (See "Using Comprehesive System Accounting on SGI Altix Machines" on page 122.)

Important: The full list of new features in this release of PBS is given in the **PBS Professional Administrator Guide**.

3.2 Introducing PBS Pro

From the user's perspective, a workload management system allows you to make more efficient use of your time. You specify the tasks you need executed. The system takes care of running these tasks and returning the results back to you. If the available computers are full, then the workload management system holds your work and runs it when the resources are available.

With PBS you create a *batch job* which you then submit to PBS. A batch job is a file (a shell script under UNIX or a `cmd` batch file under Windows) containing the set of commands you want to run on some set of execution machines. It also contains directives which specify the characteristics (attributes) of the job, and resource requirements (e.g. memory or CPU time) that your job needs. Once you create your PBS job, you can reuse it if you wish. Or, you can modify it for subsequent runs. For example, here is a simple PBS batch job:

UNIX:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
./my_application
```

Windows:

```
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
.\my_application
```

Don’t worry about the details just yet; the next chapter will explain how to create a batch job of your own.

3.3 The Two Faces of PBS

PBS provides two user interfaces: a command line interface (CLI) and a graphical user interface (GUI). The CLI lets you type commands at the system prompt. The GUI is a graphical point-and-click interface. Table 1 lists all the PBS Professional user and administrator commands. The “user commands” are discussed in this book; the “administrator commands” are discussed in the **PBS Professional Administrator Guide**. The subsequent chapters of this book will explain how to use both the CLI and GUI versions of the user commands to create, submit, and manipulate PBS jobs.

.

Table 1: PBS Professional User and Manager Commands

User Commands		Administrator Commands	
Command	**Purpose**	**Command**	**Purpose**
nqs2pbs	Convert from NQS	pbs-report	Report job statistics
pbs_rdel	Delete Adv. Reservation	pbs_hostid	Report host identifier
pbs_rstat	Status Adv. Reservation	pbs_hostn	Report host name(s)
pbs_password	Update per user / per server password[1]	pbs_migrate_users	Migrate per user / per server passwords [1]
pbs_rsub	Submit Adv.Reservation	pbs_probe	PBS diagnostic tool
pbsdsh	PBS distributed shell	pbs_rcp	File transfer tool
qalter	Alter job	pbs_tclsh	TCL with PBS API
qdel	Delete job	pbsfs	Show fairshare usage
qhold	Hold a job	pbsnodes	Node manipulation
qmove	Move job	printjob	Report job details
qmsg	Send message to job	qdisable	Disable a queue
qorder	Reorder jobs	qenable	Enable a queue
qrls	Release hold on job	qmgr	Manager interface
qselect	Select jobs by criteria	qrerun	Requeue running job
qsig	Send signal to job	qrun	Manually start a job
qstat	Status job, queue, Server	qstart	Start a queue
qsub	Submit a job	qstop	Stop a queue
tracejob	Report job history	qterm	Shutdown PBS
xpbs	Graphical User Interface	xpbsmon	GUI monitoring tool

Notes:
1 Available on Windows only.

3.4 User's PBS Environment

In order to have your system environment interact seamlessly with PBS, there are several items that need to be checked. In many cases, your system administrator will have already set up your environment to work with PBS.

In order to use PBS to run your work, the following are needed:

User must have access to the resources/hosts that the site has configured for PBS
User must have a valid account (username and group) on the execution hosts
User must be able to transfer files between hosts (e.g. via `rcp` or `scp`)

The subsequent sections of this chapter discuss these requirements in details, and provide various setup procedures.

3.5 Usernames Under PBS

By default PBS will use your login identifier as the username under which to run your job. This can be changed via the "`-u`" option to `qsub` (see section 4.9.14 "Specifying Job User ID" on page 46). The user submitting the job must be authorized to run the job under the execution user name (whether explicitly specified or not).

Important: PBS enforces a maximum username length of 15 characters. If a job is submitted to run under a username longer than this limit, the job will be rejected.

3.6 Setting Up Your UNIX/Linux Environment

A user's job may not run if the user's start-up files (i.e `.cshrc`, `.login`, or `.profile`) contain commands which attempt to set terminal characteristics. Any such command sequence within these files should be skipped by testing for the environment variable **`PBS_ENVIRONMENT`**. This can be done as shown in the following sample `.login`:

```
...
setenv MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
      do terminal settings here
endif
```

You should also be aware that commands in your startup files should not generate output when run under PBS. As in the previous example, commands that write to stdout should not be run for a PBS job. This can be done as shown in the following sample `.login`:

```
...
setenv MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS_ENVIRONMENT ) then
      do terminal settings here
      run command with output here
endif
```

When a PBS job runs, the "exit status" of the last command executed in the job is reported by the job's shell to PBS as the "exit status" of the job. (We will see later that this is important for job dependencies and job chaining.) However, the last command executed might not be the last command in your job. This can happen if your job's shell is `csh` on the execution host and you have a `.logout` there. In that case, the last command executed is from the `.logout` and not your job. To prevent this, you need to preserve the job's exit status in your `.logout` file, by saving it at the top, then doing an explict `exit` at the end, as shown below:

```
set EXITVAL = $status
previous contents of .logout here
exit $EXITVAL
```

Likewise, if the user's login shell is `csh` the following message may appear in the standard output of a job:

```
Warning: no access to tty, thus no job control in this shell
```

This message is produced by many `csh` versions when the shell determines that its input is not a terminal. Short of modifying `csh`, there is no way to eliminate the message. Fortunately, it is just an informative message and has no effect on the job.

3.6.1 Setting MANPATH on SGI Systems

The PBS "man pages" (UNIX manual entries) are installed on SGI systems under `/usr/bsd`. In order to find the PBS man pages, users will need to ensure that `/usr/bsd` is set

within their MANPATH. The following example illustrates this for the C shell:

```
setenv MANPATH /usr/man:/usr/local/man:/usr/bsd:$MANPATH
```

3.7 Setting Up Your Windows Environment

This section discusses the setup steps needed for running PBS Professional in a Microsoft Windows environment, including host and file access, passwords, and restrictions on home directories.

3.7.1 Windows User's HOMEDIR

Each Windows user is assumed to have a home directory (HOMEDIR) where his/her PBS job would initially be started. (The home directory is also the starting location of files when users specify relative path arguments to qsub/qalter -W stagein/stage-out options.)

If a user has not been explicitly assigned a home directory, then PBS will use this windows-assigned default as the base location for the user's default home directory. More specifically, the actual home path will be:

```
[ PROFILE_PATH]\My Documents\PBS Pro
```

For instance, if a *userA* has not been assigned a home directory, it will default to a local home directory of:

```
\Documents and Settings\userA\My Documents\PBS Pro
```

UserA's job will use the above path as working directory, any relative pathnames in stagein, stageout, output, error file delivery will resolve to the above path.

Note that Windows can return as PROFILE_PATH one of the following forms:

```
\Documents and Settings\username
\Documents and Settings\username.local-hostname
\Documents and Settings\username.local-hostname.00N where N is a number
\Documents and Settings\username.domain-name
```

3.7.2 Windows Usernames and Job Submission

A PBS job is run from a user account and the associated username string must conform to the POSIX-1 standard for portability. That is, the username must contain only alphanumeric characters, dot (.), underscore (_), and/or hyphen "-". The hyphen must not be the first letter of the username. If "@" appears in the username, then it will assumed to be in the context of a windows domain account: `username@domainname`. An exception to the above rule is the space character which is allowed. If a space character appears in a username string, then it will be displayed quoted and must specified in a quoted manner. The following example requests the job to run under account "Bob Jones".

```
qsub -u "Bob Jones" my_job
```

3.7.3 Windows rhosts File

The Windows `rhosts` file (and PBS configuration files such as `.xpbsrc` and `.xpbsmonrc`) is located in the user's [`PROFILE_PATH`], for example: `\Documents and Settings\`*`username`*`\.rhosts`, with the format:

`hostname username`

Important: Be sure the `.rhosts` file is owned by user or an administrator-type group, and has write-type access granted only to the owning user or an administrator-type user or group.

This file can also determine if a remote user is allowed to submit jobs to the local PBS Server, if the mapped user is an Administrator-type of account. For example, the following entry in user `susan`'s `.rhosts` file on the execution host `server1` would permit user `susan` to run jobs submitted from her workstation wks031:

```
wks031 susan
```

Furthermore, in order for Susan's output files from her job to be returned to her automatically by PBS, she would need to add an entry to her `.rhosts` file on her workstation naming the execution host `server1`.

```
server1 susan
```

If instead, Susan has access to several execution hosts, she would need to add all of them

to her `.rhosts` file:

```
server1 susan
server2 susan
server3 susan
```

Note that Domain Name Service (DNS) on Windows may return different permutations for a full hostname, thus it is important to list all the names that a host may be known by. For instance, if host server4 is known as "server4", "server4.<subdomain>", or "server4.<subdomain>.<domain>" you should list all three in the `.rhosts` file.

```
server4 susan
server4.subdomain susan
server4.subdomain.domain susan
```

As discussed in the previous section, usernames with embedded white space must also be quoted if specified in any `hosts.equiv` or `.rhosts` files, as shown below.

```
server5.subdomain.domain "Bob Jones"
```

3.7.4 Windows Mapped Drives and PBS

In Windows XP (unlike Windows 2000), when you map a drive, it is mapped "locally" to your session. The mapped drive cannot be seen by other processes outside of your session. A drive mapped on one session cannot be un-mapped in another session even if it's the same user. This has implications for running jobs under PBS. Specifically if you map a drive, `cd` to it, and submit job from that location, the node that executes the job may not be able to deliver the files back to the same location from which you issued `qsub`. The workaround is to use the "`-o`" or "`-e`" options to `qsub` and specify a local (non-mapped) directory location for the job output and error files. For details see section 4.9.2 "Redirecting output and error files" on page 41.

3.8 Environment Variables

There are a number of environment variables provided to the PBS job. Some are taken from the user's environment and carried with the job. Others are created by PBS. Still others can be explicitly created by the user for exclusive use by PBS jobs. All PBS-provided environment variable names start with the characters "`PBS_`". Some are then followed by a capital O ("`PBS_O_`") indicating that the variable is from the job's originating environment (i.e. the user's). Appendix A gives a full listing of all environment variables provided to PBS jobs and their meaning. The following short example lists some of the more useful

variables, and typical values.

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/new/bin:/usr/local/bin:/bin
PBS_O_SHELL=/sbin/csh
PBS_O_HOST=cray1
PBS_O_WORKDIR=/u/user1
PBS_O_QUEUE=submit
PBS_JOBID=16386.cray1
PBS_QUEUE=crayq
PBS_ENVIRONMENT=PBS_INTERACTIVE
```

There are a number of ways that you can use these environment variables to make more efficient use of PBS. In the example above we see **PBS_ENVIRONMENT**, which we used earlier in this chapter to test if we were running under PBS. Another commonly used variable is **PBS_O_WORKDIR** which contains the name of the directory from which the user submitted the PBS job.

There are also two environment variables that you can set to affect the behavior of PBS. The environment variable **PBS_DEFAULT** defines the name of the default PBS Server. Typically, it corresponds to the system name of the host on which the Server is running. If **PBS_DEFAULT** is not set, the default is defined by an administrator established file (usually `/etc/pbs.conf` on UNIX, and `[PBS Destination Folder]\pbs.conf` on Windows).

The environment variable **PBS_DPREFIX** determines the prefix string which identifies directives in the job script. The default prefix string is "`#PBS`"; however the Windows user may wish to change this as discussed in section 4.5 "Changing the Job's PBS Directive" on page 31.

3.9 Temporary Scratch Space: TMPDIR

PBS creates an environment variable, `TMPDIR`, which contains the full path name to a temporary "scratch" directory created for each PBS job. The directory will be removed when the job terminates.

Under Windows, `TMP` will also be set to the value of `%TMPDIR%`. The temporary directory

will be created under either `\winnt\temp` or `\windows\temp`, unless an alternative directory was specified by the administrator in the MOM configuration file.

User can access the job-specific temporary space, by changing directory to it inside their job script. For example:

UNIX:

```
...
cd $TMPDIR
...
```

Windows:

```
...
cd %TMPDIR%
...
```

Chapter 4
Submitting a PBS Job

This chapter discusses the different parts of a PBS job and how to create and submit a PBS job. Topics such as requesting resources and specifying limits on jobs are also covered.

4.1 PBS Jobs

A PBS job can be run at the command line or via xpbs.

At the command line, the user can create a job script, and submit it. During submission it is possible to override elements in the job script. Alternatively, PBS will read from input typed at the command line.

4.1.1 PBS Job Script

A PBS job script consists of:

1. An optional shell specification (UNIX)
2. PBS directives
3. Tasks -- programs or commands

To run a PBS job, the user can type

```
qsub <name of script>
```

4.1.1.1 1. Specifying the shell

UNIX Users: Since the job file under UNIX is a "shell script", the first line of the job file specifies which shell to use to execute the script. The Bourne shell (`sh`) is the default, but you can change this to your favorite shell. This first line can be omitted if (1) it is acceptable for the job file to be interpreted using the Bourne shell, and (2) if the job file is "executable" (that is mode `rwx-r-x-r-x`, or 755—see the `chmod`(1) command for details). The remainder of the examples in this manual will assume these conditions are true. If this is not true for your site, simply add the shell specifier as shown on line one above.

Windows Users: Windows does not use a shell specification. This line will not appear for a Windows job.

4.1.1.2 2. PBS Directives

PBS directives are at the top of the script file. They are used to request resources or job control options. A directive begins with the default string "#PBS". Using a directive to request a resource or job control option performs the action of setting a job attribute. These attributes can also be set using options to the qsub command, which will override directives.

4.1.1.3 3. The User's Tasks

These can be programs or commands. This is where the user specifies an application to be run.

Important: In Windows, if you use `notepad` to create a job script, the last line does not automatically get newline terminated. Be sure to put one explicitly, otherwise, PBS job will get the following error message:

```
More?
```

when the Windows command interpreter tries to execute that last line.

4.1.2 Job Attributes

Job attributes can be set either by using directives or by giving options to the qsub command. These two methods have the same functionality. Options to the qsub command will override PBS directives, which override defaults. Some job attributes have default values preset in PBS. Some job attributes' default values are set at the user's site.

4.2 Submitting a PBS Job

There are a few ways to submit a PBS job using the command line. The first is to create a job script and run it using qsub.

`Submitting a Job Script`

For example, with job script "myjob", the user can submit it by typing

```
qsub myjob
16387.foo.exampledomain
```

PBS returns a *job identifier* (e.g. "`16387.foo.exampledomain`" in the example above.) Its format will be:

```
sequence-number.servername.domain
```

You'll need the job identifier for any actions involving the job, such as checking job status, modifying the job, tracking the job, or deleting the job.

If "my_job" contains the following, the user is naming the job "testjob", and running a program called "myprogram".

```
#!/bin/sh
#PBS -N testjob
./myprogram
```

4.2.0.1 Overriding Directives

PBS directives in a script can be overridden by using the equivalent options to qsub. For example, to override the PBS directive naming the job, and name it "newjob", the user could type

```
qsub -N newjob my_job
```

4.2.0.2 Submitting a Simple Job

Jobs can also be submitted without specifying values for attributes. The simplest way to submit a job is to type

```
qsub myapplication <ret>
```

Here, the user has simply told PBS to run myapplication.

4.2.0.3Jobs Without a Job Script

It is possible to submit a job to PBS without first creating a job script file. If you run the `qsub` command, with the resource requests on the command line, and then press "enter" without naming a job file, PBS will read input from the keyboard. (This is often referred to as a "here document".) You can direct `qsub` to stop reading input and submit the job by typing on a line by itself a `control-d` (UNIX) or `control-z`, then `enter` (Windows).

Note that, under UNIX, if you enter a `control-c` while `qsub` is reading input, `qsub` will terminate the process and the job will not be submitted. Under Windows, however, often the `control-c` sequence will, depending on the command prompt used, cause `qsub` to submit the job to PBS. In such case, a `control-break` sequence will usually terminate the `qsub` command.

```
qsub <ret>
     [directives]
     [tasks]
ctrl-D
```

4.2.1 Requesting Resources

Note that you are *not* required to use a separate "`-l`" for each resource you request. You can combine multiple requests by separating them with a comma:

```
qsub -l ncpus=16,walltime=4:00:00 my_job
16389.cluster
```

The same rule applies to the job script as well, as the next example Windows script shows.

```
#PBS -l walltime=1:00:00,mem=400mb
#PBS -l ncpus=4
#PBS -j oe

date /t
.\my_application
date /t
```

4.3 How PBS Parses a Job Script

The `qsub` command scans the lines of the script file for directives. Scanning will continue until the first executable line, that is, a line that is not blank, not a directive line, nor a line whose first non white space character is "`#`". If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to `qsub` if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix (i.e. "`#PBS`"). The remainder of the directive line consists of the options to `qsub` in the same syntax as they appear on the command line. The option character is to be preceded with the "`-`" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence. If an option is present in a directive and not on the command line, that option and its argument, if any, will be taken from there.

4.4 A Sample PBS Job

Let's look at an example PBS job in detail:

UNIX

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l mem=400mb
#PBS -j oe

date
./my_application
date
```

Windows

```
1
2  #PBS -l walltime=1:00:00
3  #PBS -l mem=400mb
4  #PBS -j oe
5
6  date /t
7  .\my_application
8  date /t
```

Note the job script is an executable file under UNIX and is not under Windows. On line one in the example above Windows does not show a shell directive. (The default on Windows is the batch command language.) Also note that it is possible under both Windows and UNIX to specify to PBS the scripting language to use to interpret the job script (see the "-S" option to `qsub` in section 4.9.9 "Specifying which scripting language to use" on page 44).

Lines 2-8 of both files are almost identical. The primary differences will be in file and directory path specification (such as the use of drive letters and slash vs. backslash as the path separator).

Lines 2-4 are PBS directives. PBS reads down the shell script until it finds the first line that is not a valid PBS directive, then stops. It assumes the rest of the script is the list of commands or tasks that the user wishes to run. In this case, PBS sees lines 6-8 as being user commands.

We will see shortly how to use the `qsub` command to submit PBS jobs. Any option that you specify to the `qsub` command line (except "`-I`") can also be provided as a PBS directive inside the PBS script. PBS directives come in two types: resource requirements and job control options.

In our example above, lines 2-3 specify the "`-l`" resource list option, followed by a specific resource request. Specifically, lines 2-3 request 1 hour of wall-clock time and 400 megabytes (MB) of memory. We will see later that a typical job will also request either CPUs or nodes (differences discussed in section 4.6 "Submitting Single-node vs Multi-node Jobs" on page 32).

Line 4 is not a resource directive. Instead it specifies how PBS should handle some aspect of this job. (Specifically, the "`-j oe`" requests that PBS *join* the `stdout` and `stderr` output streams of the job into a single stream.)

Finally lines 6-8 are the command lines for executing the program(s) we wish to run. You can specify as many programs, tasks, or job steps as you need.

4.5 Changing the Job's PBS Directive

By default, the text string "`#PBS`" is used by PBS to determine which lines in the job file are PBS directives. The leading "`#`" symbol was chosen because it is a comment delimiter to all shell scripting languages in common use on UNIX systems. As a comment the scripting language itself ignores the directives, but PBS finds them.

Under Windows, however, the command interpreter does not recognize the '`#`' symbol as a comment, and will generate a benign, non-fatal warning when it encounters each "`#PBS`" string. While it does not cause a problem for the batch job, it can be annoying or disconcerting to the user. Therefore Windows users may wish to specify a different PBS directive, via either the `PBS_DPREFIX` environment variable, or the "`-C`" option to `qsub`. For example, we can direct PBS to use the string "`REM PBS`" instead of "`#PBS`" and use this directive string in our job script::

```
REM PBS -l walltime=1:00:00
REM PBS -l mem=400mb
REM PBS -j oe

date /t
.\my_application
date /t
```

Given the above job script, we can submit it to PBS in one of two ways:

```
set PBS_DPREFIX=REM PBS
qsub my_job_script
```

or

```
qsub -C "REM PBS" my_job_script
```

For additional details on the "`-C`" option to `qsub`, see section 4.9 "Job Submission Options" on page 39.

4.6 Submitting Single-node vs Multi-node Jobs

In PBS, jobs can be run either on a single system (single-node jobs) or on two or more systems (multi-node jobs). In the simplest usage, a single node job is submitted using the "`ncpus`" resource while a multi-node job is submitted using the "`nodes`" resource. For example:

Single node job: `qsub -l ncpus=3 myjob.pbs`
Multi-node job: `qsub -l nodes=3 myjob.pbs`

Any job submitted using the nodes resource is a multi-node job so that the following example is treated as a multi-node job:

Multi-node job: `qsub -l nodes=1 myjob.pbs`

Resource requests for single node jobs are simple, requesting the number of cpus and other resources using the "`-l`" option. For instance,

```
qsub -l ncpus=4,mem=8GB,walltime=1:00:00
```

Important: Do not submit multi-node jobs (not even "`-lnodes=1`"), to SGI Altix and Origin systems, as doing so would request the entire system, not a node within the system.

Requesting resources for multi-node jobs can be rather more complex. Please consult Chapter 9 "Running Multi-node Jobs" on page 151 before submitting any jobs with resource requests.

4.7 Passwords and Windows Jobs

When running PBS in a password protected Windows environment, you will need to specify to PBS the password needed in order to run your jobs. There are two methods of doing this: (1) by providing PBS with a password once to be used for all jobs ("single signon method"), or (2) by specifying the password for each job when submitted ("per job method"). Check with your system administrator to see which method was configured at your site.

4.7.1 Single-Signon Password Method

To provide PBS with a password to be used for all your PBS jobs, use the `pbs_password` command. This command can be used wether or not you have jobs enqueued in PBS. The command usage syntax is:

```
pbs_password [ -s server] [ -r] [ -d] [ user]
```

The available options to `pbs_password` are:

(none) the password credential of the current user (one who executed the command) on the default PBS Server is updated to the prompted password. Any user jobs previously held due to an invalid password will not be released.

-r any job previously held due to an invalid password is released. This can also be accomplished by using the `qrls` command.

-s serverA the password credential of the current user on PBS serverA is updated to the prompted password.

-s serverA userA the password credential of userA on PBS serverA is updated to the prompted password. However, if userA is not the same as the current user, then this action is permitted only if:

1. current user is root or admin type of account,
2. userA has given the current user explicit access via an `ruserok()` mechanism (i.e. an appropriate entry appears on the system `hosts.equiv` file, or current user has an entry in userA's `rhosts` file).

-d delete the current user's password from the default PBS Server.

Note that `pbs_password` encrypts the password obtained from user before sending it to the PBS Server. The `pbs_password` command does not change the user's password on the current host; only the password that is cached in PBS.

4.7.2 Per Job Password Method

If you are running in a password-protected Windows environment, but the single-signon method has not been configured at your site, then you will need to supply a password with the submission of each job. You can do this via the `qsub` command, with the `-Wpwd` option, and supply the password when prompted.

```
qsub -Wpwd="-Wpwd="" job.script
```

The password specified will not be shown on screen and will be passed onto the program, which will then encrypt it and save it securely for use by the job.

The password can also be specified in `xpbs` using the "`SUBMIT-PASSWORD`" entry box in the Submit window. The password you type in will not be shown on the screen.

Important: Both the `-Wpwd` option to `qsub`, and the `xpbs SUBMIT-PASSWORD` entry box can only be used when submitting jobs to Windows. The UNIX `qsub` does not support the `-Wpwd` option; and if you type a password into the `xpbs SUBMIT-PASSWORD` entry box under UNIX, the job will be rejected..

Keep in mind that in a multi-node job, the password supplied will be propagated to all the sister nodes. This requires that the password be the same on user's accounts on all the nodes. The use of domain account for a multi-node job will be ideal in this case.

Important: Because of enhanced security features found in Windows 2003 Server, you may not be able to run non-passworded jobs.

Accessing network share drives/resources within a job session also requires that you submit the job with a password via `qsub -W pwd=""` or the "`SUBMIT-PASSWORD`" entry box in `xpbs`.

Furthermore, if the job is submitted *without* a password, do not use the native `rcp` command from within the job script, as it will generate the error: "unable to get user name". Instead, please use `pbs_rcp`.

4.8 PBS System Resources

You can request a variety of resources that can be allocated and used by your job, including CPUs, nodes, memory, time (walltime or cputime), and/or disk space. As we saw above, resources are specified using the "`-l resource_list`" option to `qsub` or in

your job script. Doing so defines the resources that are required by the job as a whole, and establishes a limit to the amount of resource that can be consumed. If not set for a generally available resource, the limit is infinite.

The *resource_list* argument is of the form:

```
resource_name[ =value][ ,resource_name[ =value] ,...]
```

The resource values are specified using the following data types:

node_spec — specifies the number and type of nodes, processors per node, tasks per node, etc, as needed by multi-node jobs. See "Running Multi-node Jobs" on page 151 for a complete explanation of use.

resc_spec — specifies a set of resources and the conditions under which they should be allocated to a single-node job. See section 4.10 "Single-Node Conditional Requests" on page 51.

time — specifies a maximum time period the resource can be used. Time is expressed in seconds as an integer, or in the form:

```
[[ hours:] minutes:] seconds[ .milliseconds]
```

size — specifies the maximum amount in terms of bytes (default) or words. It is expressed in the form `integer[ suffix]`. The suffix is a multiplier defined in the following table. The size of a word is the word size on the execution host.

b or w	bytes or words.
kb or kw	Kilo (1024) bytes or words.
mb or mw	Mega (1,048,576) bytes or words.
gb or gw	Giga (1,073,741,824) bytes or words.

string — is comprised of a series of alpha-numeric characters containing no whitespace, beginning with an alphabetic character.

unitary — specifies the maximum amount of a resource which is expressed as a simple integer.

Different resources are available on different systems, often depending on the architecture of the computer itself. The table below lists the available resources that can be requested by PBS jobs on any system. Note that the Job Type column indicates if the resource is supported for single-node or multi-node/parallel jobs.

Table 2: PBS Resources Available on All Systems

Resource	Meaning and Usage	Data Type	Job Type
`arch`	System architecture needed by job. `qsub -l arch=linux ...`	string	single & multi-node
`cput`	Maximum, aggregate CPU time required by all processes in job. `qsub -l cput=5:00:00 ...`	time	single node
`file`	Maximum disk space requirements for any single file to be created by job. `qsub -l file=300mb ...`	size	single & multi-node
`host`	Name of requested host/node for this job. `qsub -l host=`*`hostname`*` ...`	string	single node
`mem`	Maximum amount of physical memory (RAM) required by job. `qsub -l mem=512mb ...`	size	single node
`ncpus`	Number of CPUs (processors) required by job. `qsub -l ncpus=16 ...`	unitary	single node
`nice`	Requested job priority (e.g. “nice” on UNIX). For Windows, affects the job PRIORITY_CLASS; nice < 0 results in HIGH; 0 results in NORMAL; nice > 0 results in IDLE. `qsub -l nice=30 ...`	unitary	single node
`nodes`	Number and/or type of nodes needed by job. (See also section 10.2 “Detailed Node Specification Syntax” on page 152.)	node_spec	multi-node

Table 2: PBS Resources Available on All Systems

Resource	Meaning and Usage	Data Type	Job Type
pcput	Per-process maximum CPU time (i.e. for any single process in the job). `qsub -l pcput=3600 ...`	time	single node
pmem	Per-process maximum amount of physical memory (i.e. for any single process of the job). `qsub -l pmem=100mb ...`	size	single node
pvmem	Per-process maximum amount of virtual memory (i.e. for any single process in the job). `qsub -l pvmem=200mb ...`	size	single node
resc	Single-node variable resource specification string. (See also section 4.10 “Single-Node Conditional Requests” on page 51.)	resc_spec	single node
vmem	Maximum, aggregate amount of virtual memory used by all concurrent processes in the job. `qsub -l vmem=400mb ...`	size	single node
walltime	Maximum amount of real time (wall-clock elapsed time) which the job needs to execute (run). `qsub -l walltime=4:00:00 ...`	time	single & multi-node

On Cray systems running UNICOS 8 or later, there are additional resources that may be requested by PBS jobs, as shown below. (See also the Cray addendum to this manual for Cray-specific resources.)

Table 3: PBS Resources on Cray UNICOS

Resource	Meaning	Units
mppe	The number of processing elements used by a single process in the job. `qsub -l mppe=512 ...`	unitary
mppt	Maximum wallclock time used by job on the MPP. `qsub -l mppt=4:00:00 ...`	time
mta, mtb...mth	Maximum number of magnetic tape drives required in the corresponding device class of a or b. `qsub -l mta=1 ...`	unitary
pf	Maximum number of file system blocks that can be used by all process in the job. `qsub -l pf=1000 ...`	size
pmppt	Maximum amount of wall clock time used on the MPP by a single process in the job. `qsub -l pmppt=4:00:00 ...`	time
pncpus	Maximum number of processors used by any single process in the job. `qsub -l pncpus=4 ...`	unitary
ppf	Maximum number of file system blocks that can be used by a single process in the job. `qsub -l ppf=500 ...`	size
procs	Maximum number of processes in the job. `qsub -l procs=128 ...`	unitary
psds	Maximum number of data blocks on the SDS (secondary data storage) for any process in the job. `qsub -l psds=300 ...`	size
sds	Maximum number of data blocks on the SDS (secondary data storage) for the job. `qsub -l sds=1000 ...`	size

4.9 Job Submission Options

There are many options to the `qsub` command. The table below gives a quick summary of the available options; the rest of this chapter explains how to use each one.

Table 4: Options to the `qsub` Command

Option	Function and Page Reference
`-A account_string`	“Specifying a local account” on page 48
`-a date_time`	“Deferring execution” on page 45
`-C "DPREFIX"`	“Changing the Job’s PBS Directive” on page 31
`-c interval`	“Specifying job checkpoint interval” on page 46
`-e path`	“Redirecting output and error files” on page 41
`-h`	“Holding a job (delaying execution)” on page 45
`-I`	“Interactive-batch jobs” on page 50
`-J X-Y[:Z]`	“Job Array” on page 131
`-j` *`join`*	“Merging output and error files” on page 49
`-k` *`keep`*	“Retaining output and error files on execution host” on page 49
`-l resources_list` `-l resc=`*`resc_spec`* `-l nodes=`*`node_spec`*	“PBS System Resources” on page 34 “Single-Node Conditional Requests” on page 51 “Running Multi-node Jobs” on page 151
`-M user_list`	“Setting e-mail recipient list” on page 43
`-m MailOptions`	“Specifying e-mail notification” on page 42
`-N name`	“Specifying a job name” on page 43
`-o path`	“Redirecting output and error files” on page 41
`-p priority`	“Setting a job’s priority” on page 44
`-q destination`	“Specifying Queue and/or Server” on page 40
`-r value`	“Marking a job as “rerunnable” or not” on page 44

Table 4: Options to the `qsub` Command

Option	Function and Page Reference
`-S path_list`	"Specifying which scripting language to use" on page 44
`-u user_list`	"Specifying Job User ID" on page 46
`-V`	"Exporting environment variables" on page 42
`-v variable_list`	"Expanding environment variables" on page 42
`-W depend=list`	"Specifying Job Dependencies" on page 107
`-W group_list=list`	"Specifying job groupID" on page 48
`-W stagein=list`	"Input/Output File Staging" on page 110
`-W stageout=list`	"Input/Output File Staging" on page 110
`-W cred=dce`	"Running PBS in a UNIX DCE Environment" on page 128
`-W block=opt`	"Requesting qsub Wait for Job Completion" on page 106
`-W pwd='password'`	"Per Job Password Method" on page 34 and "Running PBS in a UNIX DCE Environment" on page 128
`-W umask=nnn`	"Changing UNIX Job umask" on page 106
`-z`	"Suppressing job identifier" on page 50

4.9.1 Specifying Queue and/or Server

The "`-q destination`" option to `qsub` allows you to specify a particular destination to which you want the job submitted. The *destination* names a queue, a Server, or a queue at a Server. The `qsub` command will submit the script to the Server defined by the *destination* argument. If the *destination* is a routing queue, the job may be routed by the Server to a new destination. If the `-q` option is not specified, the `qsub` command will submit the script to the default queue at the default Server. (See also the discussion of **`PBS_DEFAULT`** in "Environment Variables" on page 22.) The destination specification takes the following form:

```
-q [ queue[ @host] ]
```

```
qsub -q queue my_job

qsub -q @server my_job
```

```
#PBS -q queueName
...
```

```
qsub -q queueName@serverName my_job

qsub -q queueName@serverName.domain.com my_job
```

4.9.2 Redirecting output and error files

The "`-o path`" and "`-e path`" options to `qsub` allows you to specify the name of the files to which the standard output (stdout) and the standard error (stderr) file streams should be written. The path argument is of the form: `[ hostname:]path_name` where *hostname* is the name of a host to which the file will be returned and *path_name* is the path name on that host. You may specify relative or absolute paths. If you specify only a file name, it is assumed to be relative to your home directory. The following examples illustrate these various options.

```
#PBS -o /u/user1/myOutputFile
#PBS -e /u/user1/myErrorFile
```

```
qsub -o myOutputFile my_job
qsub -o /u/user1/myOutputFile my_job
qsub -o myWorkstation:/u/user1/myOutputFile my_job
qsub -e myErrorFile my_job
qsub -e /u/user1/myErrorFile my_job
qsub -e myWorkstation:/u/user1/myErrorFile my_job
```

Note that if the PBS client commands are used on a Windows host, then special characters like spaces, backslashes (\), and colons (:) can be used in command line arguments such as for specifying pathnames, as well as drive letter specifications. The following are allowed:

```
qsub -o \temp\my_out job.scr
qsub -e "host:e:\Documents and Settings\user\Desktop\output"
```

The error output of the above job is to be copied onto the `e:` drive on *`host`* using the path "`\Documents and Settings\user\Desktop\output`". The quote marks are required when arguments to `qsub` contain spaces.

4.9.3 Exporting environment variables

The "`-V`" option declares that all environment variables in the `qsub` command's environment are to be exported to the batch job.

```
qsub -V my_job
```

```
#PBS -V
...
```

4.9.4 Expanding environment variables

The "`-v variable_list`" option to `qsub` allows you to specify additional environment variables to be exported to the job. *variable_list* names environment variables from the `qsub` command environment which are made available to the job when it executes. The *variable_list* is a comma separated list of strings of the form `variable` or `variable=value`. These variables and their values are passed to the job.

```
qsub -v DISPLAY,myvariable=32 my_job
```

4.9.5 Specifying e-mail notification

The "`-m MailOptions`" defines the set of conditions under which the execution server will send a mail message about the job. The *MailOptions* argument is a string which consists of either the single character "`n`", or one or more of the characters "`a`", "`b`", and "`e`". If no email notification is specified, the default behavior will be the same as for "`-m a`" .

`a`	send mail when job is *aborted* by batch system
`b`	send mail when job *begins* execution
`e`	send mail when job *ends* execution
`n`	*do not* send mail

```
qsub -m ae my_job
```

```
#PBS -m b
...
```

4.9.6 Setting e-mail recipient list

The "`-M user_list`" option declares the list of users to whom mail is sent by the execution server when it sends mail about the job. The *user_list* argument is of the form:

```
user[ @host][ ,user[ @host] ,...]
```

If unset, the list defaults to the submitting user at the `qsub` host, i.e. the job owner.

```
qsub -M user1@mydomain.com my_job
```

Important: PBS on Windows can only send email to addresses that specify an actual hostname that accepts port 25 (sendmail) requests. For the above example on Windows you will need to specify:

```
qsub -M user1@host.mydomain.com
```

where "`host.mydomain.com`" accepts port 25 connections.

4.9.7 Specifying a job name

The "`-N name`" option declares a name for the job. The *name* specified may be up to and including 15 characters in length. It must consist of printable, non-whitespace characters with the first character alphabetic, and contain no "special characters". If the `-N` option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

```
qsub -N myName my_job
```

```
#PBS -N myName
...
```

4.9.8 Marking a job as "rerunnable" or not

The "`-r y|n`" option declares whether the job is rerunnable. To rerun a job is to terminate the job and requeue it in the execution queue in which the job currently resides. The *value* argument is a single character, either "`y`" or "`n`". If the argument is "`y`", the job is rerunnable. If the argument is "`n`", the job is not rerunnable. The default value is "`y`", rerunnable.

```
qsub -r n my_job
```

```
#PBS -r n
...
```

4.9.9 Specifying which scripting language to use

The "`-S path_list`" option declares the path and name of the scripting language to be used in interpreting the job script. The option argument *path_list* is in the form: `path[ @host][ ,path[ @host] ,...]` Only one path may be specified for any host named, and only one path may be specified without the corresponding host name. The path selected will be the one with the host name that matched the name of the execution host. If no matching host is found, then the path specified without a host will be selected, if present. If the `-S` option is not specified, the option argument is the null string, or no entry from the *path_list* is selected, then PBS will use the user's login shell on the execution host.

```
qsub -S /usr/local/bin/perl my_job
```

```
#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter
...
```

Important: Using this option under Windows is more complicated because if you change from the default shell of `cmd`, then a valid `PATH` is not automatically set. Thus if you use the "`-S`" option under Windows, you must explicitly set a valid `PATH` as the first line of your job script.

4.9.10 Setting a job's priority

The "`-p priority`" option defines the priority of the job. The *priority* argument must be an integer between -1024 (lowest priority) and +1023 (highest priority) inclusive. The

default is no priority which is equivalent to a priority of zero.

This option allows the user to specify a priority for their jobs. However, this option is dependant upon the local scheduling policy. By default the "sort jobs by job-priority" feature is disabled. If your local PBS administrator has enabled it, then all queued jobs will be sorted based on the user-specified priority. (If you need an absolute ordering of your own jobs, see "Specifying Job Dependencies" on page 107.)

```
qsub -p 120 my_job
```

```
#PBS -p -300
...
```

4.9.11 Deferring execution

The "`-a date_time`" option declares the time after which the job is eligible for execution. The *date_time* argument is in the form: `[[[[ CC] YY] MM] DD] hhmm[ .SS]` where `CC` is the first two digits of the year (the century), `YY` is the second two digits of the year, `MM` is the two digits for the month, `DD` is the day of the month, `hh` is the hour, `mm` is the minute, and the optional `SS` is the seconds. If the month, `MM`, is not specified, it will default to the current month if the specified day `DD`, is in the future. Otherwise, the month will be set to next month. Likewise, if the day, `DD`, is not specified, it will default to today if the time `hhmm` is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a job at 11:15am with a time of "`1110`", the job will be eligible to run at 11:10am tomorrow. Other examples include:

```
qsub -a 0700 my_job
```

```
#PBS -a 10220700
...
```

4.9.12 Holding a job (delaying execution)

The "`-h`" option specifies that a *user hold* be applied to the job at submission time. The job will be submitted, then placed in a hold state. The job will remain ineligible to run until the hold is released. (For details on releasing a held job see "Holding and Releasing Jobs" on page 97.)

```
qsub -h my_job
```

```
#PBS -h
...
```

4.9.13 Specifying job checkpoint interval

The "`-c interval`" option defines the interval (in minutes) at which the job will be checkpointed, if this capability is provided by the operating system (i.e. under SGI IRIX and Cray Unicos). If the job executes upon a host which does not support checkpointing, this option will be ignored. The *interval* argument is specified as:

`n` No checkpointing is to be performed.

`s` Checkpointing is to be performed only when the Server executing the job is shutdown.

`c` Checkpointing is to be performed at the default minimum time for the Server executing the job.

`c`=minutes Checkpointing is to be performed at an interval of *minutes*, which is the integer number of minutes of CPU time used by the job. This value must be greater than zero.

`u` Checkpointing is unspecified, thus resulting in the same behavior as "s".

If "`-c`" is not specified, the checkpoint attribute is set to the value "`u`".

```
qsub -c c my_job
```

```
#PBS -c c=10
...
```

Checkpointing is not supported for job arrays.

4.9.14 Specifying Job User ID

PBS requires that a user's name be consistent across a server and its execution hosts, but not across a submission host and a server. A user may have access to more than one server, and may have a different username on each server. In this environment, if a user wishes to submit a job to any of the available servers, the username for each server is specified. The wildcard username will be used if the job ends up at yet another server not specified, but only if that wildcard username is valid.

For example, our user is UserS on the submission host HostS, UserA on server ServerA, and UserB on server ServerB, and is UserC everywhere else. Note that this user must be UserA on all ExecutionA and UserB on all ExecutionB machines. Then our user can use

"qsub -u UserA@ServerA,UserB@ServerB,UserC" for the job. The job owner will always be UserS.

4.9.14.1 qsub -u: User ID with UNIX

The server's flatuid attribute determines whether it assumes that identical usernames mean identical users. If true, it assumes that if UserS exists on both the submission host and the server host, then UserS can run jobs on that server. If not true, the server calls ruserok() which uses /etc/hosts.equiv and .rhosts to authorize UserS to run as UserA.

Table 5: UNIX User ID and flatuid

Value of flatuid	Submission host username/server host username	
	Same: UserS/UserS	Different: UserS/UserA
True	Server assumes user has permission to run job	Server checks whether UserS can run job as UserA
Not true	Server checks whether UserS can run job as UserS	Server checks whether UserS can run job as UserA

4.9.14.2 qsub -u: User ID with Windows

Under Windows, if a user has a non-admin account, the server's hosts.equiv file is used to determine whether that user can run a job on a given server. For an admin account, [PROFILE_PATH].\rhosts is used, and the server's acl_roots attribute must be set to allow job submissions. Usernames containing spaces are allowed as long as the username length is no more than 15 characters, and the usernames are quoted when used in the command line.

Table 6: Requirements for Admin User to Submit Job

Locatio/Action	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/UserA
[PROFILE_PATH]\ .rhosts contains	For UserS on ServerA, add <HostS> UserS	For UserA on ServerA, add <HostS> UserS
set ServerA's acl_roots attribute	qmgr> set server acl_roots=UserS	qmgr> set server acl_roots=UserA

Table 7: Requirements for Non-admin User to Submit Job

File	Submission host username/Server host username	
	Same: UserS/UserS	Different: UserS/UserA
hosts.equiv on ServerA	<HostS>	<HostS> UserS

4.9.15 Specifying job groupID

The "`-W group_list=g_list`" option defines the group name under which the job is to run on the execution system. The *g_list* argument is of the form:

```
group[ @host][ ,group[ @host],...]
```

Only one group name may be given per specified host. Only one of the group specifications may be supplied without the corresponding host specification. That group name will used for execution on any host not named in the argument list. If not set, the *group_list* defaults to the primary group of the user under which the job will be run.

```
qsub -W group_list=grpA,grpB@jupiter my_job
```

4.9.16 Specifying a local account

The "`-A account_string`" option defines the account string associated with the job. The *account_string* is an opaque string of characters and is not interpreted by the Server which executes the job. This value is often used by sites to track usage by locally defined account names.

Important: Under IRIX and Unicos, if the Account string is specified, it must be a valid account as defined in the system "User Data Base", UDB.

```
qsub -A Math312 my_job
```

```
#PBS -A accountNumber
...
```

4.9.17 Merging output and error files

The "`-j join`" option declares if the standard error stream of the job will be merged with the standard output stream of the job. A *join* argument value of `oe` directs that the two streams will be merged, intermixed, as standard output. A *join* argument value of `eo` directs that the two streams will be merged, intermixed, as standard error. If the *join* argument is `n` or the option is not specified, the two streams will be two separate files.

```
qsub -j oe my_job
```

```
#PBS -j eo
...
```

4.9.18 Retaining output and error files on execution host

The "`-k keep`" option defines which (if either) of standard output (STDOUT) or standard error (STDERR) of the job will be retained on the execution host. If set, this option overrides the path name for the corresponding file. If not set, neither file is retained on the execution host. The argument is either the single letter "e" or "o", or the letters "e" and "o" combined in either order. Or the argument is the letter "n". If "`-k`" is not specified, neither file is retained.

`e` The standard error file is to be retained on the execution host. The file will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: *`job_name`*`.e`*`sequence`* where `job_name` is the name specified for the job, and *`sequence`* is the sequence number component of the job identifier.

`o` The standard output file is to be retained on the execution host. The file will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by: *`job_name`*`.o`*`sequence`* where `job_name` is the

name specified for the job, and *sequence* is the sequence number component of the job identifier.

`eo` Both standard output and standard error will be retained.

`oe` Both standard output and standard error will be retained.

`n` Neither file is retained.

```
qsub -k oe my_job
```

```
#PBS -k eo
...
```

4.9.19 Suppressing job identifier

The "`-z`" option directs the `qsub` command to not write the job identifier assigned to the job to the command's standard output.

```
qsub -z my_job
```

```
#PBS -z
...
```

4.9.20 Interactive-batch jobs

PBS provides a special kind of batch job called *interactive-batch*. An interactive-batch job is treated just like a regular batch job (in that it is queued up, and has to wait for resources to become available before it can run). Once it is started, however, the user's terminal input and output are connected to the job in a matter similar to a `login` session. It appears that the user is logged into one of the available execution machines, and the resources requested by the job are reserved for that job. Many users find this useful for debugging their applications or for computational steering. The "`-I`" option declares that the job is an interactive-batch job.

Important: Interactive-batch jobs are not currently supported on Windows.

If the `-I` option is specified on the command line or in a script directive, the job is an interactive job. If a script is given, it will be processed for directives, but any executable commands will be discarded. When the job begins execution, all input to the job is from the terminal session in which `qsub` is running.

When an interactive job is submitted, the `qsub` command will not terminate when the job

is submitted. qsub will remain running until the job terminates, is aborted, or the user interrupts qsub with a SIGINT (the control-C key). If qsub is interrupted prior to job start, it will query if the user wishes to exit. If the user responds "yes", qsub exits and the job is aborted.

Once the interactive job has started execution, input to and output from the job pass through qsub. Keyboard-generated interrupts are passed to the job. Lines entered that begin with the tilde ('~') character and contain special sequences are interpreted by qsub itself. The recognized special sequences are:

~.	qsub terminates execution. The batch job is also terminated.
~susp	If running under the UNIX C shell, suspends the qsub program. "susp" is the suspend character, usually CNTL-Z.
~asusp	If running under the UNIX C shell, suspends the input half of qsub (terminal to job), but allows output to continue to be displayed. "asusp" is the auxiliary suspend character, usually control-Y.

4.10 Single-Node Conditional Requests

PBS Professional offers the ability to use boolean logic in the specification of certain resources (such as architecture, memory, wallclock time, and CPU count) *within a single node*. A new resource specification string (resc_spec) attribute has been added called "resc"). Used with the resource list option ("-l") to the qsub command, this feature provides more control over selecting nodes on which to run your job.

Important: At this time, this feature controls the selection of single nodes, with the meaning of "allocate my job a node with the following properties". This feature does not apply to multi-node jobs.

Important: Single-node conditional requests (resc_spec specifications) are not currently available on Windows.

For example, say you wanted to submit a job that can run on either the Solaris or Irix oper-

ating system, and you want PBS to run the job on the first available node of either type. You could add the following "resc" specification to your qsub command line (or your job).

```
qsub -l resc="(arch=='solaris7') || (arch=='irix')" my_job
```

```
#PBS -l resc="(arch=='solaris7')||(arch=='irix')"
#PBS -l mem=100MB
#PBS -l ncpus=4
...
```

You could in fact combine all three of the lines in the above example into a single resc_spec specification, if you so desired:

```
qsub -l resc="((arch=='solaris7') || (arch=='irix')) && (mem=100MB) &&(ncpus=4)"
qsub -l resc="(arch==linux)||(arch==irix)"
```

The following example shows requesting different memory amounts depending on the architecture that the job runs on:

```
qsub -l resc="( (arch=='solaris7') && (mem=100MB)||((arch=='irix')&&(mem=1GB) )"
```

Furthermore, it is possible to specify multiple resource specification strings. The first resc specification will be evaluated. If it can be satisfied, then it will be used. If not, then next resc string will be used. The example below indicates that you want 16 CPUs, but if you can't have 16 CPUs, then give you 8 with half the memory and twice the wall-clock time. But if you can't have 8 CPUs, then give you four and 1/4 the memory, and four times the walltime.

```
qsub \
-l resc="(ncpus=16)&& (mem=1GB)   &&(walltime=1:00)" \
-l resc="(ncpus=8) && (mem=512MB)&&(walltime=2:00)" \
-l resc="(ncpus=4) && (mem=256MB)&&(walltime=4:00)" ...
```

This is different then putting them all into one resc specification. If you were to do

```
qsub -l resc= "(ncpus=16)||(ncpus=8)||(ncpus=4)" ...
```

you would be requesting the first available node which has either 16, 8, or 4 CPUs. In this case, PBS doesn't go through all the nodes checking for 16 first, then 8, then 4, as it does when using multiple resc specifications.

Important: Note the difference between "==" (comparison) and "=" (assignment) within a `resc_spec`. The comparison operators only impact which node is selected for the job, they do not establish limits on the job. The assignment operator, however, is equivalent to separate specifications of `-l mem=x` and `-l walltime=y` in order to set the job limits.

You can do more than just using the equality and assignment operators. You can describe the characteristics of a node, but not request them. For example, if you were to request the following:

```
qsub -l resc="(ncpus>16)&&(mem>=2GB)" -lncpus=2 -lmem=100MB
```

you would be indicating that you want a node with more then 16 CPUs but you only want two of them allocated to your job:

4.11 Job Attributes

A PBS job has the following attributes, which may be set by the various options to `qsub` (for details see section 4.9 "Job Submission Options" on page 39).

`Account_Name` Reserved for local site accounting. If specified (using the `-A` option to `qsub`) this value is carried within the job for its duration, and is included in the job accounting records.

`Checkpoint` If supported by the Server implementation and the host operating system, the checkpoint attribute determines when checkpointing will be performed by PBS on behalf of the job. The legal values for checkpoint are described under the `qalter` and `qsub` commands.

`depend` The type of inter-job dependencies specified by the job owner.

`Error_Path` The final path name for the file containing the job's standard error stream. See the `qsub` and `qalter` command description for more detail.

Execution_Time — The time after which the job may execute. The time is maintained in seconds since Epoch. If this time has not yet been reached, the job will not be scheduled for execution and the job is said to be in *wait* state.

group_list — A list of *group_names@hosts* which determines the group under which the job is run on a given host. When a job is to be placed into execution, the Server will select a group name according to the rules specified for use of the qsub command.

Hold_Types — The set of holds currently applied to the job. If the set is not null, the job will not be scheduled for execution and is said to be in the *hold* state. Note, the *hold* state takes precedence over the *wait* state.

Job_Name — The name assigned to the job by the qsub or qalter command.

Join_Path — If the Join_Paths attribute is oe, then the job's standard error stream will be merged, inter-mixed, with the job's standard output stream and placed in the file determined by the Output_Path attribute. The Error_Path attribute is maintained, but ignored. However, if the Join_Paths attribute is eo, then the job's standard output stream will be merged, intermixed, with the job's standard error stream and placed in the file determined by the Error_Path attribute, and the Output_Path attribute will be ignored.

Keep_Files — If Keep_Files contains the values "o" KEEP_OUTPUT and/or "e" KEEP_ERROR the corresponding streams of the batch job will be retained on the execution host upon job termination. Keep_Files overrides the Output_Path and Error_Path attributes.

Mail_Points — Identifies when the Server will send email about the job.

Mail_Users — The set of users to whom mail may be sent when the job makes certain state changes.

`Output_Path` The final path name for the file containing the job's standard output stream. See the `qsub` and `qalter` command description for more detail.

`Priority` The job scheduling priority assigned by the user.

`Rerunnable` The rerunnable flag given by the user.

`Resource_List` The resource list is a set of *name=value* strings of the resources required by the job. The value also establishes the limit of usage of that resource. If not set, the value for a resource may be determinate by a queue or Server default established by the administrator.

`Shell_Path_List`
A set of absolute paths of the program to process the job's script file.

`stagein` The list of files to be staged in prior to job execution.

`stageout` The list of files to be staged out after job execution.

`User_List` The list of *user@hosts* which determines the username under which the job is run on a given host.

`Variable_List` This is the list of environment variables passed with the *Queue Job* batch request.

`comment` An attribute for displaying comments about the job from the system. Visible to any client.

The following attributes are read-only, they are established by the Server and are visible to the user but cannot be set or changed by a user.

`alt_id` For a few systems, such as Irix 6.x running Array Services, the session id is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it will also be recorded in the end-of-job accounting record. For Irix 6.x running Array Services, the `alt_id` attribute is set to the Array Session Handle (ASH) assigned to the job.

array	boolean; true if applied to a job array
array_id	string; applies to subjob; job array identifier for given subjob
array_index	string; applies to subjob; index number of given subjob
array_indices_remaining	string; applies to job array; list of indices of subjobs still queued. Range or list of ranges
array_indices_submitted	string; applies to job array; complete list of indices of subjobs given at submission time. Given as a range.
array_state_count	string; applies to job array; lists number of subjobs in each state
ctime	The time that the job was created.
etime	The time that the job became eligible to run, i.e. in a queued state while residing in an execution queue.
exec_host	If the job is running, this is set to the name of the host or hosts on which the job is executing. The format of the string is "node/N[*C][+...]", where "node" is the name of a node, "N" is process or task slot on that node, and "C" is the number of CPUs allocated to the job. C does not appear if it is one.
egroup	If the job is queued in an execution queue, this attribute is set to the group name under which the job is to be run. [This attribute is available only to the batch administrator.]
euser	If the job is queued in an execution queue, this attribute is set to the user name under which the job is to be run. [This attribute is available only to the batch administrator.]
hashname	The name used as a basename for various files, such as the job file, script file, and the standard output and error of the job. [This attribute is available only to the batch administrator.]
interactive	True if the job is an interactive PBS job.

`Job_Owner` The login name on the submitting host of the user who submitted the batch job.

`job_state` The state of the job.

`mtime` The time that the job was last modified, changed state, or changed locations.

`qtime` The time that the job entered the current queue.

`queue` The name of the queue in which the job currently resides.

Chapter 5
Using the xpbs GUI

The PBS graphical user interface is called **xpbs**, and provides a user-friendly, point and click interface to the PBS commands. xpbs utilizes the tcl/tk graphics toolsuite, while providing the user with the same functionality as the PBS CLI commands. In this chapter we introduce xpbs, and show how to create a PBS job using xpbs.

5.1 Starting xpbs

If PBS is installed on your local workstation, or if you are running under Windows, you can launch xpbs by double-clicking on the xpbs icon on the desktop. You can also start xpbs from the command line with the following command.

UNIX:

```
xpbs &
```

Windows:

```
xpbs.exe
```

Doing so will bring up the main xpbs window, as shown below.

5.1.1 Running xpbs Under Unix

Before running xpbs for the first time under UNIX, you may need to configure your workstation for it. Depending on how PBS is installed at your site, you may need to allow

xpbs to be displayed on your workstation. However, if the PBS client commands are installed locally on your workstation, you can skip this step. (Ask your PBS administrator if you are unsure.)

The most secure method of running xpbs remotely and displaying it on your local XWindows session is to redirect the XWindows traffic through ssh (secure shell), via setting the "X11Forwarding yes" parameter is in the sshd_config file. (Your local system administrator can provide details on this process if needed.)

An alternative, but less secure, method is to direct your X-Windows session to permit the xpbs client to connect to your local X-server. Do this by running the xhost command with the name of the host from which you will be running xpbs, as shown in the example below:

```
xhost + server.mydomain.com
```

Next, on the system from which you will be running xpbs, set your X-Windows **DISPLAY** variable to your local workstation. For example, if using the C-shell:

```
setenv DISPLAY myWorkstation:0.0
```

However, if you are using the Bourne or Korn shell, type the following:

```
export DISPLAY=myWorkstation:0.0
```

5.2 Using xpbs: Definitions of Terms

The various panels, boxes, and regions (collectively called “widgets”) of xpbs and how they are manipulated are described in the following sections. A *listbox* can be multi-selectable (a number of entries can be selected/highlighted using a mouse click) or single-selectable (one entry can be highlighted at a time).

For a multi-selectable listbox, the following operations are allowed:

- left-click to select/highlight an entry.
- shift-left-click to contiguously select more than one entry.
- control-left-click to select multiple non-contiguous entries..
- click the *Select All / Deselect All* button to select all entries or deselect all entries at once.

- double clicking an entry usually activates some action that uses the selected entry as a parameter.

An *entry* widget brought into focus with a left-click. To manipulate this widget, simply type in the text value. Use of arrow keys, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for horizontally scanning a long text entry string.

A *matrix of entry boxes* is usually shown as several rows of entry widgets where a number of entries (called fields) can be found per row. The matrix is accompanied by up/down arrow buttons for paging through the rows of data, and each group of fields gets one scrollbar for horizontally scanning long entry strings. Moving from field to field can be done using the <Tab> (move forward), <Cntrl-f> (move forward), or <Cntrl-b> (move backward) keys.

A *spinbox* is a combination of an entry widget and a horizontal scrollbar. The entry widget will only accept values that fall within a defined list of valid values, and incrementing through the valid values is done by clicking on the up/down arrows.

A *button* is a rectangular region appearing either raised or pressed that invokes an action when clicked with the left mouse button. When the button appears pressed, then hitting the <RETURN> key will automatically select the button.

A *text region* is an editor like widget. This widget is brought into focus with a left-click. To manipulate this widget, simply type in the text. Use of arrow keys, backspace/delete key, mouse selection of text for deletion or overwrite, copying and pasting with sole use of mouse buttons are permitted. This widget is usually accompanied by a scrollbar for vertically scanning a long entry.

5.3 Introducing the xpbs Main Display

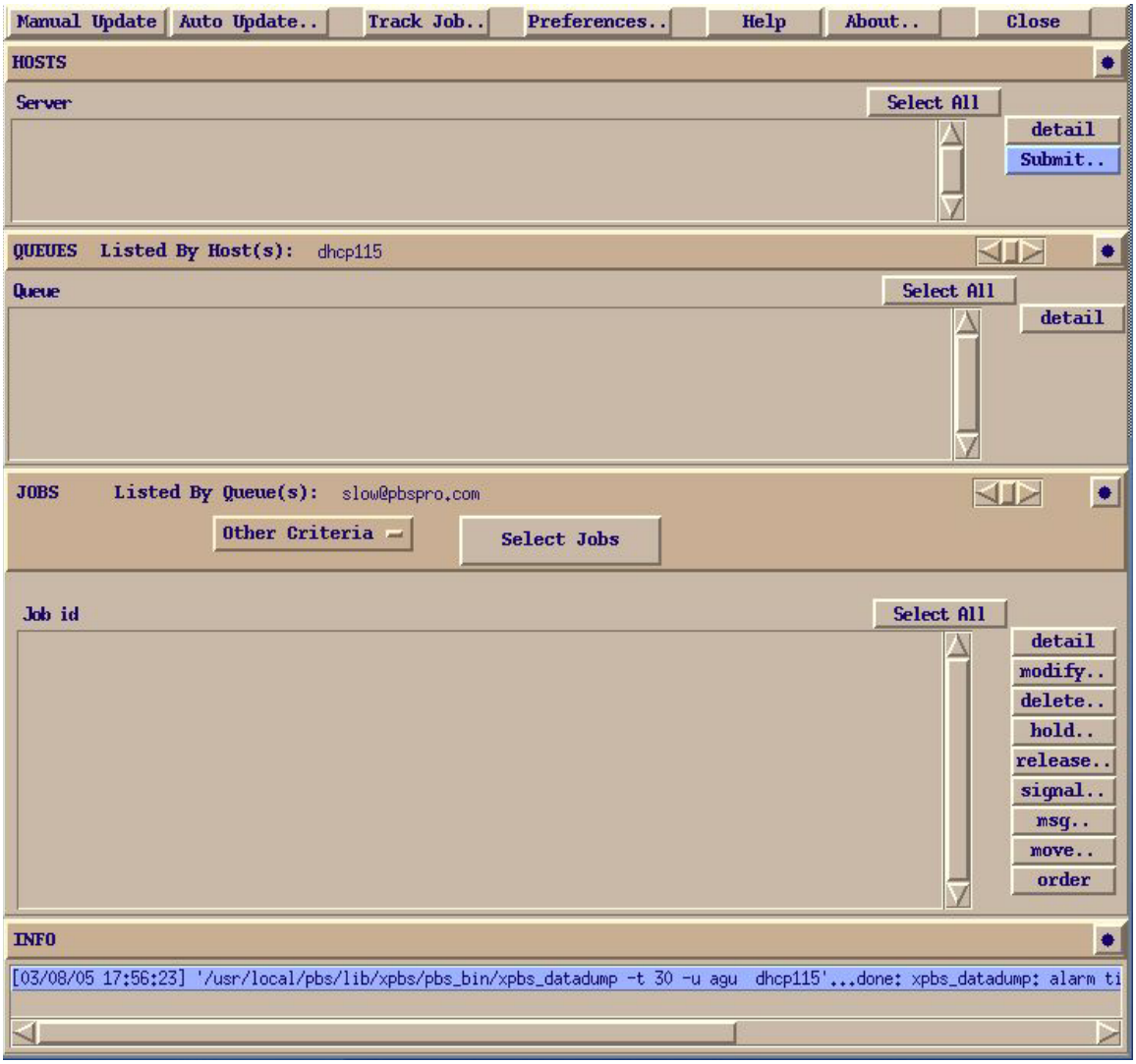
The main window or display of `xpbs` is comprised of five collapsible subwindows or *panels*. Each panel contains specific information. Top to bottom, these panel are: the Menu Bar, Hosts panel, Queues panel, Jobs panel, and the Info panel

5.3.1 xpbs Menu Bar

The Menu Bar is composed of a row of command buttons that signal some action with a click of the left mouse button. The buttons are:

Manual Update	forces an update of the information on hosts, queues, and jobs.
Auto Update	sets an automatic update of information every user-specified number of minutes.
Track Job	for periodically checking for returned output files of jobs.
Preferences	for setting parameters such as the list of Server host(s) to query.
Help	contains some help information.
About	givens general information about the xpbs developer.
Close	for exiting `xpbs` plus saving the current setup information.

.



5.3.2 xpbs Hosts Panel

The Hosts panel is composed of a leading horizontal HOSTS bar, a listbox, and a set of command buttons. The HOSTS bar contains a minimize/maximize button, identified by a dot or a rectangular image, for displaying or iconizing the Hosts region. The listbox displays information about favorite Server host(s), and each entry is meant to be selected via a single left-click, shift-left-click" for contiguous selection, or control-left-click for non-contiguous selection.

To the right of the Hosts Panel are a series of buttons that represent actions that can be performed on selected host(s). Usage of these buttons will be explained in detail below.

detail — Provides information about selected Server host(s). This functionality can also be achieved by double clicking on an entry in the Hosts listbox.

submit — For submitting a job to any of the queues managed by the selected host(s).

terminate — For terminating (shutting down) PBS Servers on selected host(s). (Visible via the "`-admin`" option only.)

Important: Note that some buttons are only visible if `xpbs` is started with the "`-admin`" option, which requires manager or operator privilege to function.

The middle portion of the Hosts Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 8: xpbs Server Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted
Tot	Count of jobs currently enqueued in any state
Que	Count of jobs in the Queued state
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state

Table 8: xpbs Server Column Headings

Heading	Meaning
Trn	Count of jobs in the Transiting state
Ext	Count of jobs in the Exiting state
Status	Status of the corresponding Server
PEsInUse	Count of Processing Elements (CPUs, PEs, Nodes) in Use

5.3.3 xpbs Queues Panel

The Queues panel is composed of a leading horizontal QUEUES bar, a listbox, and a set of command buttons. The QUEUES bar lists the hosts that are consulted when listing queues; the bar also contains a minimize/maximize button for displaying or iconizing the Queues panel. The listbox displays information about queues managed by the Server host(s) selected from the Hosts panel; each listbox entry can be selected as described above for the Hosts panel.

To the right of the Queues Panel area are a series of buttons that represent actions that can be performed on selected queue(s).

detail	provides information about selected queue(s). This functionality can also be achieved by double clicking on a Queue listbox entry.
stop	for stopping the selected queue(s). (`-admin` only)
start	for starting the selected queue(s). (`-admin` only)
disable	for disabling the selected queue(s). (`-admin` only)
enable	for enabling the selected queue(s). (`-admin` only)

The middle portion of the Queues Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 9: xpbs Queue Column Headings

Heading	Meaning
Max	Maximum number of jobs permitted
Tot	Count of jobs currently enqueued in any state
Ena	Is queue enabled? yes or no

Table 9: xpbs Queue Column Headings

Heading	Meaning
Str	Is queue started? yes or no
Que	Count of jobs in the Queued state
Run	Count of jobs in the Running state
Hld	Count of jobs in the Held state
Wat	Count of jobs in the Waiting state
Trn	Count of jobs in the Transiting state
Ext	Count of jobs in the Exiting state
Type	Type of queue: execution or route
Server	Name of Server on which queue exists

5.3.4 xpbs Jobs Panel

The Jobs panel is composed of a leading horizontal JOBS bar, a listbox, and a set of command buttons. The JOBS bar lists the queues that are consulted when listing jobs; the bar also contains a minimize/maximize button for displaying or iconizing the Jobs region. The listbox displays information about jobs that are found in the queue(s) selected from the Queues listbox; each listbox entry can be selected as described above for the Hosts panel.

The region just above the Jobs listbox shows a collection of command buttons whose labels describe criteria used for filtering the Jobs listbox contents. The list of jobs can be selected according to the owner of jobs (Owners), job state (Job_States), name of the job (Job_Name), type of hold placed on the job (Hold_Types), the account name associated with the job (Account_Name), checkpoint attribute (Checkpoint), time the job is eligible for queueing/execution (Queue_Time), resources requested by the job (Resources), priority attached to the job (Priority), and whether or not the job is rerunnable (Rerunnable).

The selection criteria can be modified by clicking on any of the appropriate command buttons to bring up a selection box. The criteria command buttons are accompanied by a *Select Jobs* button, which when clicked, will update the contents of the Jobs listbox based on the new selection criteria. Note that only jobs that meet ***all*** the selected criteria will be displayed.

Finally, to the right of the Jobs panel are the following command buttons, for operating on selected job(s):

detail	provides information about selected job(s). This functionality can also be achieved by double-clicking on a Jobs listbox entry.
modify	for modifying attributes of the selected job(s).
delete	for deleting the selected job(s).
hold	for placing some type of hold on selected job(s).
release	for releasing held job(s).
signal	for sending signals to selected job(s) that are running.
msg	for writing a message into the output streams of selected job(s).
move	for moving selected job(s) into some specified destination.
order	for exchanging order of two selected jobs in a queue.
run	for running selected job(s). (-admin only)
rerun	for requeueing selected job(s) that are running. (-admin only)

The middle portion of the Jobs Panel has abbreviated column names indicating the information being displayed, as the following table shows:

Table 10: xpbs Job Column Headings

Heading	Meaning
Job id	Job Identifier
Name	Name assigned to job, or script name
User	User name under which job is running
PEs	Number of Processing Elements (CPUs) requested
CputUse	Amount of CPU time used
WalltUse	Amount of wall-clock time used
S	State of job
Queue	Queue in which job resides

5.3.5 xpbs Info Panel

The Info panel shows the progress of the commands' executed by `xpbs`. Any errors are written to this area. The INFO panel also contains a minimize/maximize button for displaying or iconizing the Info panel.

5.3.6 xpbs Keyboard Tips

There are a number of shortcuts and key sequences that can be used to speed up using `xpbs`. These include:

Tip 1. All buttons which appear to be depressed into the dialog box/subwindow can be activated by pressing the return/enter key.

Tip 2. Pressing the tab key will move the blinking cursor from one text field to another.

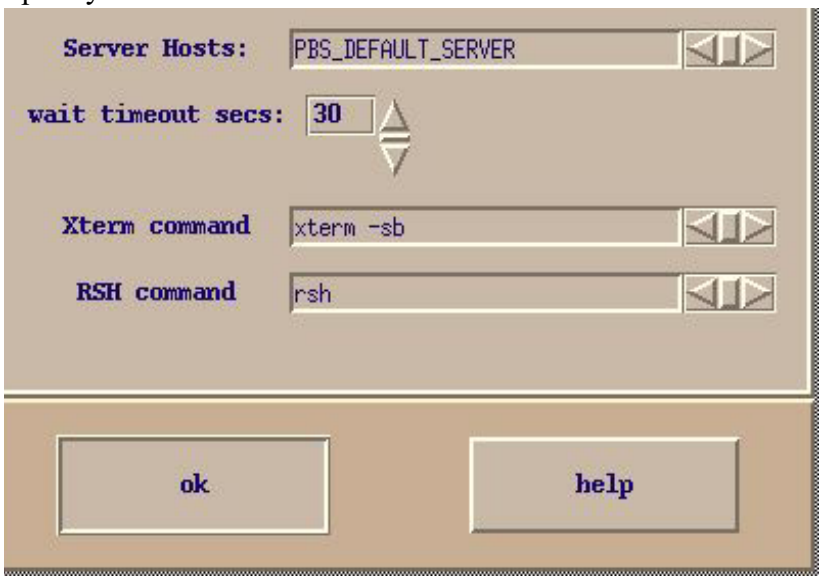
Tip 3. To contiguously select more than one entry: left-click then drag the mouse across multiple entries.

Tip 4. To non-contiguously select more than one entry: hold the control-left-click on the desired entries.

5.4 Setting xpbs Preferences

In the Menu Bar at the top of the main `xpbs` window is the Preferences button. Clicking it will bring up a dialog box that allows you to customize the behavior of xpbs:

1. Define Server hosts to query
2. Select wait timeout in seconds
3. Specify `xterm` command (for interactive jobs, UNIX only)
4. Specify which `rsh`/`ssh` command to use



5.5 Relationship Between PBS and `xpbs`

`xpbs` is built on top of the PBS client commands, such that all the features of the command line interface are available through the GUI. Each "task" that you perform using `xpbs` is converted into the necessary PBS command and then run on your behalf.

Table 11: `xpbs` Buttons and PBS Commands

Location	Command Button	PBS Command
Hosts Panel	detail	`qstat -B -f` *selected server_host(s)*
Hosts Panel	submit	`qsub` *options selected Server(s)*
Hosts Panel	terminate *	`qterm` *selected server_host(s)*
Queues Panel	detail	`qstat -Q -f` *selected queue(s)*
Queues Panel	stop *	`qstop` *selected queue(s)*
Queues Panel	start *	`qstart` *selected queue(s)*
Queues Panel	enable *	`qenable` *selected queue(s)*
Queues Panel	disable *	`qdisable` *selected queue(s)*
Jobs Panel	detail	`qstat -f` *selected job(s)*
Jobs Panel	modify	`qalter` *selected job(s)*
Jobs Panel	delete	`qdel` *selected job(s)*
Jobs Panel	hold	`qhold` *selected job(s)*
Jobs Panel	release	`qrls` *selected job(s)*
Jobs Panel	run	`qrun` *selected job(s)*
Jobs Panel	rerun	`qrerun` *selected job(s)*
Jobs Panel	signal	`qsig` *selected job(s)*
Jobs Panel	msg	`qmsg` *selected job(s)*
Jobs Panel	move	`qmove` *selected job(s)*
Jobs Panel	order	`qorder` *selected job(s)*

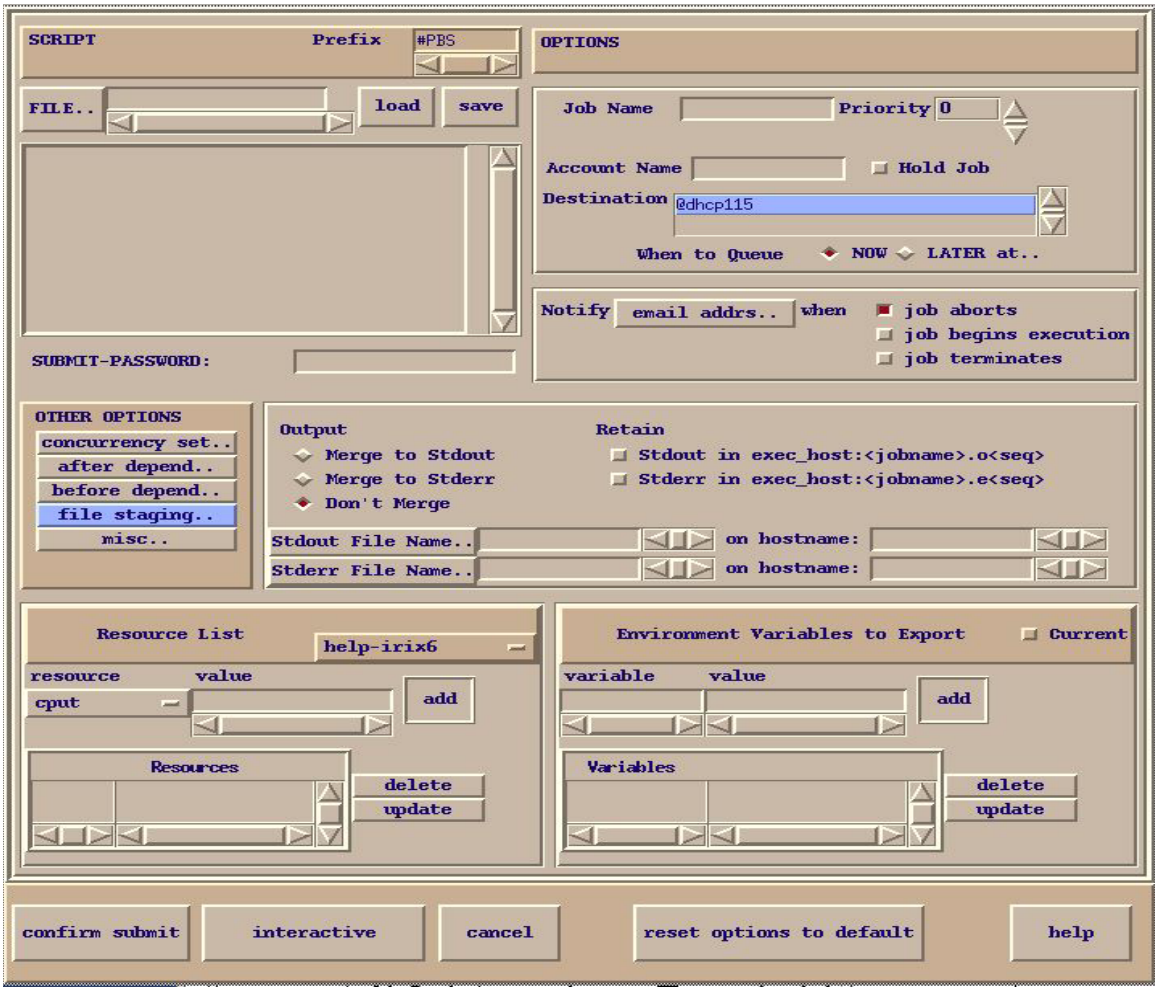
* Indicates command button is visible only if `xpbs` is started with the "`-admin`" option.

5.6 How to Submit a Job Using `xpbs`

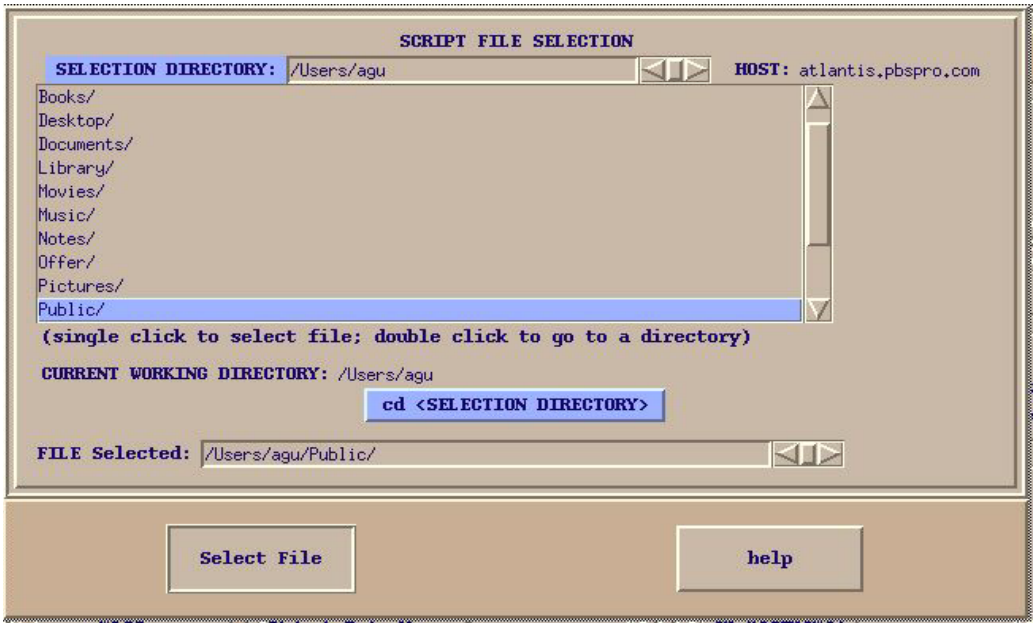
To submit a job using `xpbs`, perform the following steps:

First, select a host from the HOSTS listbox in the main `xpbs` display to which you wish to submit the job.

Next, click on the *Submit* button located next to the HOSTS panel. The *Submit* button brings up the Submit Job Dialog box (see below) which is composed of four distinct regions. The Job Script File region is at the upper left. The OPTIONS region containing various widgets for setting job attributes is scattered all over the dialog box. The OTHER OPTIONS is located just below the Job Script file region, and COMMAND BUTTONS region is at the bottom.



The job script region is composed of a header box, the text box, FILE entry box, and two buttons labeled *load* and *save.* If you have a script file containing PBS options and executable lines, then type the name of the file on the FILE entry box, and then click on the *load* button. Alternatively, you may click on the *FILE* button, which will display a File Selection browse window, from which you may point and click to select the file you wish to open. The File Selection Dialog window is shown below. Clicking on the *Select File* button will load the file into `xpbs`, just as does the *load* button described above.



The various fields in the Submit window will get loaded with values found in the script file. The script file text box will only be loaded with executable lines (non-PBS) found in the script. The job script header box has a *Prefix* entry box that can be modified to specify the PBS directive to look for when parsing a script file for PBS options.

If you don't have a existing script file to load into `xpbs`, you can start typing the executable lines of the job in the file text box.

Next, review the Destination listbox. This box lists all the queues found in the host that you selected. A special entry called "`@host`" refers to the default queue at the indicated host. Select appropriately the destination queue for the job.

Next, define any required resources in the Resource List subwindow. Finally, review the optional settings to see if any should apply to this job.

For example:

- Use the one of the buttons in the “Output” region to merge output and error files.
- Use “Stdout File Name” to define standard output file and to redirect output
- Use the “Environment Variables to Export” subwindow to have current environment variables exported to the job.
- Use the “Job Name” field in the OPTIONS subwindow to give the job a name.
- Use the “Notify email address” and one of the buttons in the OPTIONS subwindow to have PBS send you mail when the job terminates.

Now that the script is built you have four options of what to do next:

Reset options to default
Save the script to a file
Submit the job as a batch job
Submit the job as an interactive-batch job (UNIX only)

Reset clears all the information from the submit job dialog box, allowing you to create a job from a fresh start.

Use the FILE. field (in the upper left corner) to define a filename for the script. Then press the *Save* button. This will cause a PBS script file to be generated and written to the named file.

Pressing the *Confirm Submit* button at the bottom of the Submit window will submit the PBS job to the selected destination. `xpbs` will display a small window containing the job identifier returned for this job. Clicking *OK* on this window will cause it and the Submit window to be removed from your screen.

On UNIX systems (not Windows) you can alternatively submit the job as an interactive-batch job, by clicking the *Interactive* button at the bottom of the Submit Job window. Doing so will cause an X-terminal window (`xterm`) to be launched, and within that window a PBS interactive-batch job submitted. The path for the xterm command can be set via the preferences, as discussed above in section 5.4 “Setting xpbs Preferences” on page 67. For further details on usage, and restrictions, see “Interactive-batch jobs” on page 50.)

5.7 Exiting xpbs

Click on the *Close* button located in the Menu bar to leave `xpbs`. If any settings have been changed, `xpbs` will bring up a dialog box asking for a confirmation in regards to saving state information. The settings will be saved in the `.xpbsrc` configuration file, and will be used the next time you run `xpbs`, as discussed in the following section.

5.8 The xpbs Configuration File

Upon exit, the `xpbs` state may be written to the `.xpbsrc` file in the user's home directory. (See also section 3.7.1 "Windows User's HOMEDIR" on page 20.) Information saved includes: the selected host(s), queue(s), and job(s); the different jobs listing criteria; the view states (i.e. minimized/maximized) of the Hosts, Queues, Jobs, and INFO regions; and all settings in the Preferences section. In addition, there is a system-wide `xpbs` configuration file, maintained by the PBS Administrator, which is used in the absence of a user's personal `.xpbsrc` file.

5.9 xpbs Preferences

The resources that can be set in the `xpbs` configuration file, `~/.xpbsrc`, are:

*serverHosts	List of Server hosts (space separated) to query by `xpbs`. A special keyword **PBS_DEFAULT_SERVER** can be used which will be used as a placeholder for the value obtained from the `/etc/pbs.conf` file (UNIX) or "`[PBS Destination Folder]\pbs.conf`" file (Windows).
*timeoutSecs	Specify the number of seconds before timing out waiting for a connection to a PBS host.
*xtermCmd	The xterm command to run driving an interactive PBS session.
*labelFont	Font applied to text appearing in labels.
*fixlabelFont	Font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font.
*textFont	Font applied to a text widget. Keep this as fixed-width font.
*backgroundColor	The color applied to background of frames, buttons, entries, scrollbar handles.
*foregroundColor	The color applied to text in any context.
*activeColor	The color applied to the background of a selection, a selected command button, or a selected scroll bar handle.
*disabledColor	Color applied to a disabled widget.

*signalColor	Color applied to buttons that signal something to the user about a change of state. For example, the color of the *Track Job* button when returned output files are detected.
*shadingColor	A color shading applied to some of the frames to emphasize focus as well as decoration.
*selectorColor	The color applied to the selector box of a radiobutton or checkbutton.
*selectHosts	List of hosts (space separated) to automatically select/highlight in the HOSTS listbox.
*selectQueues	List of queues (space separated) to automatically select/highlight in the QUEUES listbox.
*selectJobs	List of jobs (space separated) to automatically select/highlight in the JOBS listbox.
*selectOwners	List of owners checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Owners: <list_of_owners>". See `-u` option in `qselect(1B)` for format of <list_of_owners>.
*selectStates	List of job states to look for (do not space separate) when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Job_States: <states_string>". See `-s` option in `qselect(1B)` for format of <states_string>.
*selectRes	List of resource amounts (space separated) to consult when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Resources: <res_string>". See `-l` option in `qselect(1B)` for format of <res_string>.
*selectExecTime	The Execution Time attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Queue_Time: <exec_time>". See `-a` option in `qselect(1B)` for format of <exec_time>.
*selectAcctName	The name of the account that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Account_Name: <account_name>". See `-A` option in `qselect(1B)` for format of <account_name>.
*selectCheckpoint	The checkpoint attribute relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Checkpoint: <checkpoint_arg>". See `-c` option in `qselect(1B)` for format of <checkpoint_arg>.

*selectHold	The hold types string to look for in a job when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Hold_Types: <hold_string>". See `-h` option in `qselect(1B)` for format of <hold_string>.
*selectPriority	The priority relationship (including the logical operator) to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Priority: <priority_value>". See `-p` option in `qselect(1B)` for format of <priority_value>.
*selectRerun	The rerunnable attribute to consult when limiting the list of jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Rerunnable: <rerun_val>". See `-r` option in `qselect(1B)` for format of <rerun_val>.
*selectJobName	Name of the job that will be checked when limiting the jobs appearing on the Jobs listbox in the main `xpbs` window. Specify value as "Job_Name: <jobname>". See `-N` option in `qselect(1B)` for format of <jobname>.
*iconizeHostsView	A boolean value (true or false) indicating whether or not to iconize the HOSTS region.
*iconizeQueuesView	A boolean value (true or false) indicating whether or not to iconize the QUEUES region.
*iconizeJobsView	A boolean value (true or false) indicating whether or not to iconize the JOBS region.
*iconizeInfoView	A boolean value (true or false) indicating whether or not to iconize the INFO region.
*jobResourceList	A curly-braced list of resource names as according to architecture known to `xpbs`. The format is as follows: { <arch-type1> resname1 resname2 ... resnameN } { <arch-type2> resname1 resname2 ... resnameN } { <arch-typeN> resname1 resname2 ... resnameN }

Chapter 6
Checking Job / System Status

This chapter introduces several PBS commands useful for checking status of jobs, queues, and PBS Servers. Examples for use are included, as are instructions on how to accomplish the same task using the `xpbs` graphical interface.

6.1 The `qstat` Command

The **`qstat`** command is used to the request the status of jobs, queues, and the PBS Server. The requested status is written to standard output stream (usually the user's terminal). When requesting job status, any jobs for which the user does not have view privilege are not displayed.

6.1.1 Checking Job Status

Executing the `qstat` command without any options displays job information in the default format. (An alternative display format is also provided, and is discussed below.) The default display includes the following information:

The job identifier assigned by PBS
The job name given by the submitter
The job owner
The CPU time used

The job state
The queue in which the job resides

The job state is abbreviated to a single character:

B	Job arrays only: job array has started
E	Job is exiting after having run
H	Job is held
Q	Job is queued, eligible to run or be routed
R	Job is running
S	Job is suspended by server
T	Job is in transition (being moved to a new location)
U	Job is suspended due to workstation becoming busy
W	Job is waiting for its requested execution time to be reached or job specified a stage-in request which failed for some reason.
X	Subjobs only; subjob is finished (expired.)

The following example illustrates the default display of `qstat`.

```
qstat
Job id    Name         User         Time Use S Queue
--------- ---------- ---------- -------- - -----
16.south  aims14      user1              0 H workq
18.south  aims14      user1              0 W workq
26.south  airfoil     barry       00:21:03 R workq
27.south  airfoil     barry       21:09:12 R workq
28.south  myjob       user1              0 Q workq
29.south  tns3d       susan              0 Q workq
30.south  airfoil     barry              0 Q workq
31.south  seq_35_3    donald             0 Q workq
```

An alternative display (accessed via the "`-a`" option) is also provided that includes extra information about jobs, including the following additional fields:

Session ID
Number of nodes requested
Number of parallel tasks (or CPUs)
Requested amount of memory
Requested amount of wallclock time
Elapsed time in the current job state.

```
qstat -a
                                                     Req'd     Elap
Job ID   User   Queue Jobname Ses NDS TSK Mem Time S Time
-------- ------ ----- ------- --- --- --- --- ---- - ----
16.south user1  workq aims14  --  --   1  --  0:01 H  --
18.south user1  workq aims14  --  --   1  --  0:01 W  --
51.south barry  workq airfoil 930 --   1  --  0:13 R 0:01
52.south user1  workq myjob   --  --   1  --  0:10 Q  --
53.south susan  workq tns3d   --  --   1  --  0:20 Q  --
54.south barry  workq airfoil --  --   1  --  0:13 Q  --
55.south donald workq seq_35_ --  --   1  --  2:00 Q  --
```

Other options which utilize the alternative display are discussed in subsequent sections of this chapter.

6.1.2 Viewing Specific Information

When requesting queue or Server status `qstat` will output information about each destination. The various options to `qstat` take as an operand either a job identifier or a destination. If the operand is a job identifier, it must be in the following form:

```
sequence_number[ .server_name][ @server]
```

where `sequence_number.server_name` is the job identifier assigned at submittal time, see `qsub`. If the `.server_name` is omitted, the name of the default Server will be used. If `@server` is supplied, the request will be for the job identifier currently at that Server.

If the operand is a destination identifier, it takes one of the following three forms:

```
queue
@server
queue@server
```

If *`queue`* is specified, the request is for status of all jobs in that queue at the default Server. If the *`@server`* form is given, the request is for status of all jobs at that Server. If a full destination identifier, *`queue@server`*, is given, the request is for status of all jobs in the named *`queue`* at the named *`server`*.

Important: If a PBS Server is not specified on the `qstat` command line, the default Server will be used. (See discussion of **`PBS_DEFAULT`** in "Environment Variables" on page 22.)

6.1.3 Checking Server Status

The "**`-B`**" option to `qstat` displays the status of the specified PBS Batch Server. One line of output is generated for each Server queried. The three letter abbreviations correspond to various job limits and counts as follows: Maximum, Total, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the status of the Server itself: active, idle, or scheduling.

```
qstat -B
Server       Max  Tot  Que  Run  Hld  Wat  Trn  Ext Status
-----------  ---  ---- ---- ---- ---- ---- ---- ---- ------
fast.pbspro    0   14   13    1    0    0    0    0 Active
```

When querying jobs, Servers, or queues, you can add the "`-f`" option to `qstat` to change the display to the *full* or *long* display. For example, the Server status shown above would be expanded using "`-f`" as shown below:

```
qstat -Bf
Server: fast.mydomain.com
    server_state = Active
    scheduling = True
    total_jobs = 14
    state_count = Transit:0 Queued:13 Held:0 Waiting:0
                  Running:1 Exiting:0
    managers = user1@fast.mydomain.com
    default_queue = workq
    log_events = 511
    mail_from = adm
    query_other_jobs = True
    resources_available.mem = 64mb
    resources_available.ncpus = 2
    resources_default.ncpus = 1
    resources_assigned.ncpus = 1
    resources_assigned.nodect = 1
    scheduler_iteration = 600
    pbs_version = PBSPro_7.1.41640
```

6.1.4 Checking Queue Status

The "**-Q**" option to `qstat` displays the status of all (or any specified) queues at the (optionally specified) PBS Server. One line of output is generated for each queue queried. The three letter abbreviations correspond to limits, queue states, and job counts as follows: Maximum, Total, Enabled Status, Started Status, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the type of the queue: *routing* or *execution*.

For information on options for job arrays, see "qstat: Status of a Job Array" on page 142

```
qstat -Q

Queue Max Tot Ena Str Que Run Hld Wat Trn Ext Type
----- --- --- --- --- --- --- --- --- --- --- ---------
workq   0  10 yes yes   7   1   1   1   0   0 Execution
```

The full display for a queue provides additional information:

```
qstat -Qf
Queue: workq
    queue_type = Execution
    total_jobs = 10
    state_count = Transit:0 Queued:7 Held:1 Waiting:1
                  Running:1 Exiting:0
    resources_assigned.ncpus = 1
    hasnodes = False
    enabled = True
    started = True
```

6.1.5 Viewing Job Information

We saw above that the "`-f`" option could be used to display full or long information for queues and Servers. The same applies to jobs. By specifying the "`-f`" option and a job identifier, PBS will print all information known about the job (e.g. resources requested, resource limits, owner, source, destination, queue, etc.) as shown in the following example. (See "Job Attributes" on page 53 for a description of attribute.)

```
qstat -f 89
Job Id: 89.south
    Job_Name = tns3d
    Job_Owner = susan@south.mydomain.com
    resources_used.cput = 00:00:00
    resources_used.mem = 2700kb
    resources_used.ncpus = 1
    resources_used.vmem = 5500kb
    resources_used.walltime = 00:00:00
    job_state = R
    queue = workq
    server = south
    Checkpoint = u
    ctime = Thu Aug 23 10:11:09 2004
    Error_Path = south:/u/susan/tns3d.e89
    exec_host = south/0
    Hold_Types = n
    Join_Path = oe
    Keep_Files = n
    Mail_Points = a
    mtime = Thu Aug 23 10:41:07 2004
    Output_Path = south:/u/susan/tns3d.o89
    Priority = 0
    qtime = Thu Aug 23 10:11:09 2004
    Rerunnable = True
    Resource_List.mem = 300mb
    Resource_List.ncpus = 1
    Resource_List.walltime = 00:20:00
    session_id = 2083
    Variable_List = PBS_O_HOME=/u/susan,PBS_O_LANG=en_US,
       PBS_O_LOGNAME=susan,PBS_O_PATH=/bin:/usr/bin,
       PBS_O_SHELL=/bin/csh,PBS_O_HOST=south,
       PBS_O_WORKDIR=/u/susan,PBS_O_SYSTEM=Linux,
       PBS_O_QUEUE=workq
    euser = susan
    egroup = myegroup
    queue_type = E
    comment = Job run on node south - started at 10:41
    etime = Thu Aug 23 10:11:09 2004
```

6.1.6 List User-Specific Jobs

The "**-u**" option to `qstat` displays jobs owned by any of a list of user names specified. The syntax of the list of users is:

```
user_name[ @host][ ,user_name[ @host] ,...]
```

Host names are not required, and may be "wild carded" on the left end, e.g. "`* .mydomain.com`". *`user_name`* without a "`@host`" is equivalent to "`user_name@*`", that is at any host.

```
qstat -u user1
                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --

qstat -u user1,barry

51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
```

6.1.7 List Running Jobs

The "**-r**" option to `qstat` displays the status of all running jobs at the (optionally specified) PBS Server. Running jobs include those that are running and suspended. One line of output is generated for each job reported, and the information is presented in the alternative display.

6.1.8 List Non-Running Jobs

The "**-i**" option to `qstat` displays the status of all non-running jobs at the (optionally specified) PBS Server. Non-running jobs include those that are queued, held, and waiting. One line of output is generated for each job reported, and the information is presented in the alternative display (see description above).

6.1.9 Display Size in Gigabytes

The "**-G**" option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in gigabytes (GB) rather than the default of smallest displayable units. Note that if the size specified is less than 1 GB, then the amount if rounded up to 1 GB.

6.1.10 Display Size in Megawords

The "**-M**" option to `qstat` displays all jobs at the requested (or default) Server using the alternative display, showing all size information in megawords (MW) rather than the default of smallest displayable units. A word is considered to be 8 bytes.

6.1.11 List Nodes Assigned to Jobs

The "**-n**" option to `qstat` displays the nodes allocated to any running job at the (optionally specified) PBS Server, in addition to the other information presented in the alternative display. The node information is printed immediately below the job (see job 51 in the example below), and includes the node name and number of virtual processors assigned to the job (i.e. "`south/0`", where "`south`" is the node name, followed by the virtual processor(s) assigned.). A text string of "--" is printed for non-running jobs. Notice the differences between the queued and running jobs in the example below:

```
qstat -n
                                                            Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
    --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
    --
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
    south/0
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
    --
```

6.1.12 Display Job Comments

The "**-s**" option to `qstat` displays the job comments, in addition to the other information presented in the alternative display. The job comment is printed immediately below the job. By default the job comment is updated by the Scheduler with the reason why a given job is not running, or when the job began executing. A text string of "--" is printed for jobs whose comment has not yet been set. The example below illustrates the different type of messages that may be displayed:

```
qstat -s
                                                  Req'd     Elap
Job ID   User  Queue Jobname Sess NDS TSK Mem Time S Time
-------- ----- ----- ------- ---- --- --- --- ---- - ----
16.south user1 workq aims14   --   --   1  -- 0:01 H  --
    Job held by user1 on Wed Aug 22 13:06:11 2004
18.south user1 workq aims14   --   --   1  -- 0:01 W  --
    Waiting on user requested start time
51.south barry workq airfoil  930  --   1  -- 0:13 R 0:01
    Job run on node south - started Thu Aug 23 at 10:56
52.south user1 workq my_job   --   --   1  -- 0:10 Q  --
    Not Running: No available resources on nodes
57.south susan workq solver   --   --   2  -- 0:20 Q  --
    --
```

6.1.13 Display Queue Limits

The "**-q**" option to `qstat` displays any limits set on the requested (or default) queues. Since PBS is shipped with no queue limits set, any visible limits will be site-specific. The limits are listed in the format shown below.

```
qstat -q
server: south

Queue  Memory CPU Time Walltime Node Run Que Lm  State
------ ------ -------- -------- ---- --- --- --  -----
workq    --      --       --       --    1   8 --   E R
```

6.1.14 Show State of Job, Job Array or Subjob

The "-t" option to qstat will show the state of a job, a job array object, and all non-X subjobs. In combination with "-J", qstat will show only the state of subjobs.

6.1.15 Show state of Job Arrays

The "-J" option to qstat will show only the state of job arrays. In combination with "-t", qstat will show only the state of subjobs.

6.1.16 Print Job Array Percentage Completed

The "-p" option to qstat prints the default display, with a column for Percentage Completed. For a job array, this is the number of subjobs completed and deleted, divided by the total number of subjobs.

6.2 Viewing Job / System Status with xpbs

The main display of `xpbs` shows a brief listing of all selected Servers, all queues on those Servers, and any jobs in those queues that match the *selection criteria* (discussed below). Servers are listed in the HOST panel near the top of the display.

To view detailed information about a given Server (i.e. similar to that produced by "`qstat -fB`") select the Server in question, then click the *Detail* button. Likewise, for details on a given queue (i.e. similar to that produced by "`qstat -fQ`") select the queue in question, then click its corresponding *Detail* button. The same applies for jobs as well (i.e. "`qstat -f`"). You can view detailed information on any displayed job by selecting it, and then clicking on the *Detail* button. Note that the list of jobs displayed will be dependent upon the Selection Criteria currently selected. This is discussed in the `xpbs` portion of the next section.

6.3 The `qselect` Command

The **`qselect`** command provides a method to list the job identifier of those jobs, job arrays or subjobs which meet a list of selection criteria. Jobs are selected from those owned by a single Server. When `qselect` successfully completes, it will have written to standard output a list of zero or more job identifiers which meet the criteria specified by the options. Each option acts as a filter restricting the number of jobs which might be listed. With no options, the `qselect` command will list all jobs at the Server which the user is authorized to list (query status of). The `-u` option may be used to limit the selection to jobs owned by this user or other specified users.

When an option is specified with a optional *op* component to the option argument, then *op* specifies a relation between the value of a certain job attribute and the value component of the option argument. If an *op* is allowable on an option, then the description of the option letter will indicate that *op* is allowable. The only acceptable strings for the *op* component, and the relation the string indicates, are shown in the following list:

`.eq.`	The value represented by the attribute of the job is equal to the value represented by the option argument.
`.ne.`	The value represented by the attribute of the job is not equal to the value represented by the option argument.
`.ge.`	The value represented by the attribute of the job is greater than or equal to the value represented by the option argument.
`.gt.`	The value represented by the attribute of the job is greater than the value represented by the option argument.
`.le.`	The value represented by the attribute of the job is less than or equal to the value represented by the option argument.
`.lt.`	The value represented by the attribute of the job is less than the value represented by the option argument.

The available options to `qselect` are:

-a [op]date_time

Restricts selection to a specific time, or a range of times. The `qselect` command selects only jobs for which the value of the *Execution_Time* attribute is related to the *date_time* argument by the optional *op* operator. The *date_time* argument is in the POSIX date format:

```
[[CC]YY]MMDDhhmm[.SS]
```

where the `MM` is the two digits for the month, `DD` is the day of the month, `hh` is the hour, `mm` is the minute, and the optional `SS` is the seconds. `CC` is the century and `YY` the year. If op is not specified, jobs will be selected for which the *Execution_Time* and *date_time* values are equal.

-A account_string

Restricts selection to jobs whose *Account_Name* attribute matches the specified account_string.

-c [op] interval

Restricts selection to jobs whose *Checkpoint* interval attribute matches the specified relationship. The values of the *Checkpoint* attribute are defined to have the following ordered relationship:

```
n > s > c=minutes > c > u
```

If the optional op is not specified, jobs will be selected whose *Checkpoint* attribute is equal to the interval argument.

-h hold_list

Restricts the selection of jobs to those with a specific set of hold types. Only those jobs will be selected whose *Hold_Types* attribute exactly match the value of the *hold_list* argument. The *hold_list* argument is a string consisting of one or more occurrences the single letter `n`, or one or more of the letters `u`, `o`, p, or `s` in any combination. If letters are duplicated, they are treated as if they occurred once. The letters represent the hold types:

Letter	Meaning
n	none
u	user
o	operator
p	bad password (Windows only)
s	system

-l resource_list

Restricts selection of jobs to those with specified resource amounts. Only those jobs will be selected whose *Resource_List* attribute matches the specified relation with each resource and value listed in the *resource_list* argument. The relation operator *op* **must** be present. The *resource_list* is in the following format:

```
resource_nameopvalue[ ,resource_nameopval,...]
```

-N name

Restricts selection of jobs to those with a specific name.

-p [op]priority — Restricts selection of jobs to those with a priority that matches the specified relationship. If *op* is not specified, jobs are selected for which the job Priority attribute is equal to the priority.

-q destination — Restricts selection to those jobs residing at the specified destination. The destination may be one of the following three forms:

```
queue
@server
queue@server
```

If the `-q` option is not specified, jobs will be selected from the default Server. If the destination describes only a queue, only jobs in that queue on the default batch Server will be selected. If the destination describes only a Server, then jobs in all queues on that Server will be selected. If the destination describes both a queue and a Server, then only jobs in the named queue on the named Server will be selected.

-r rerun — Restricts selection of jobs to those with the specified *Rerunnable* attribute. The option argument must be a single character. The following two characters are supported by PBS: `y` and `n`.

-s states — Restricts job selection to those in the specified states. The *states* argument is a character string which consists of any combination of the characters: `E`, `H`, `Q`, `R`, `S`, `T`, `U`, and `W`. The characters in the *states* argument have the following interpretation:

Table 12: Job States Viewable by Users

State	Meaning
E	Job is in the process of Exiting.
H	Job has been placed on Hold.
Q	Job is in the Queued state.
R	Job is in the Running state.
S	Job has been Suspended.
T	Job is Transiting between states.
U	Job suspended due to workstation user activity
W	Job is in the Waiting state.

Jobs will be selected which are in any of the specified *states*.

-u user_list Restricts selection to jobs owned by the specified user names. This provides a means of limiting the selection to jobs owned by one or more users. The syntax of the *user_list* is:

```
user_name[ @host][ ,user_name[ @host] ,...]
```

Host names may be wild carded on the left end, e.g. "`*.mydomain.com`". *User_name* without a "`@host`" is equivalent to "`user_name@*`", i.e. at any host. Jobs will be selected which are owned by the listed users at the corresponding hosts.

For example, say you want to list all jobs owned by user “barry” that requested more than 16 CPUs. You could use the following `qselect` command syntax:
:

```
qselect -u barry -l ncpus.gt.16
121.south
133.south
154.south
```

Notice that what is returned is the job identifiers of jobs that match the selection criteria. This may or may not be enough information for your purposes. Many users will use shell syntax to pass the list of job identifiers directly into `qstat` for viewing purposes, as shown in the next example (necessarily different between UNIX and Windows).

UNIX:

```
qstat -a  ` qselect -u barry -l ncpus.gt.16 `
                                                  Req'd     Elap
Job ID    User  Queue Jobname Sess NDS TSK Mem Time S Time
--------  ----- ----- ------- ---- --- --- --- ---- - ----
121.south barry workq airfoil  --   --  32  -- 0:01 H   --
133.south barry workq trialx   --   --  20  -- 0:01 W   --
154.south barry workq airfoil 930   --  32  -- 1:30 R 0:32
```

Windows (type the following at the `cmd` prompt, all on one line):

```
for /F "usebackq" %j in (`qselect -u barry -l ncpus.gt.16`)  do
( qstat -a %j )
121.south
133.south
154.south
```

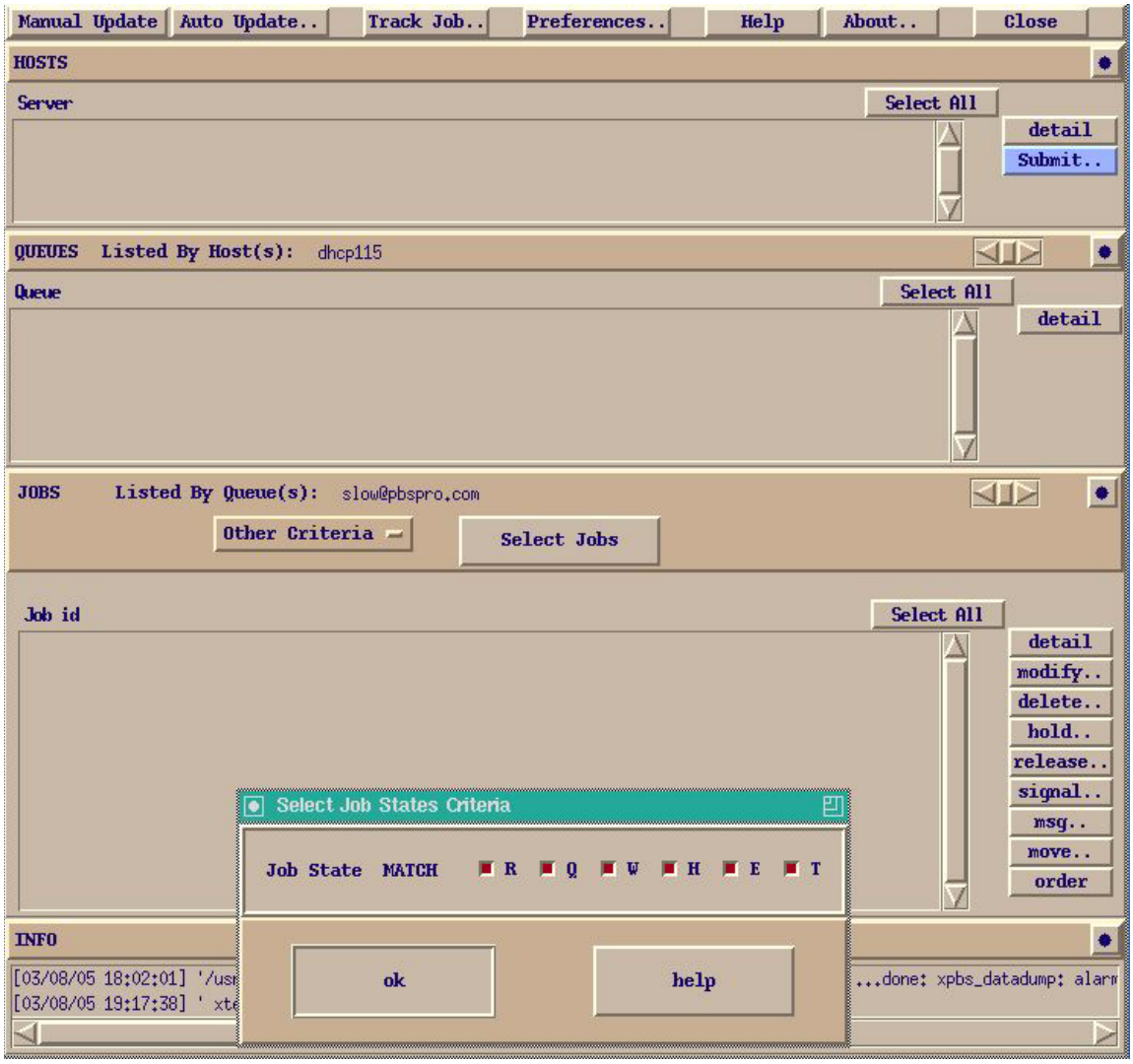
Note: This technique of using the output of the `qselect` command as input to `qstat` can also be used to supply input to other PBS commands as well.

-J	Shows only job array identifiers
-t	Shows job, job array and subjob identifiers
-s [argument]	Restricts selection to states specified in arguments

6.4 Selecting Jobs Using xpbs

The `xpbs` command provides a graphical means of specifying job selection criteria, offering the flexibility of the `qselect` command in a point and click interface. Above the JOBS panel in the main `xpbs` display is the *Other Criteria* button. Clicking it will bring up a menu that lets you choose and select any job selection criteria you wish.

The example below shows a user clicking on the *Other Criteria* button, then selecting *Job States*, to reveal that all job states are currently selected. Clicking on any of these job states would remove that state from the selection criteria.



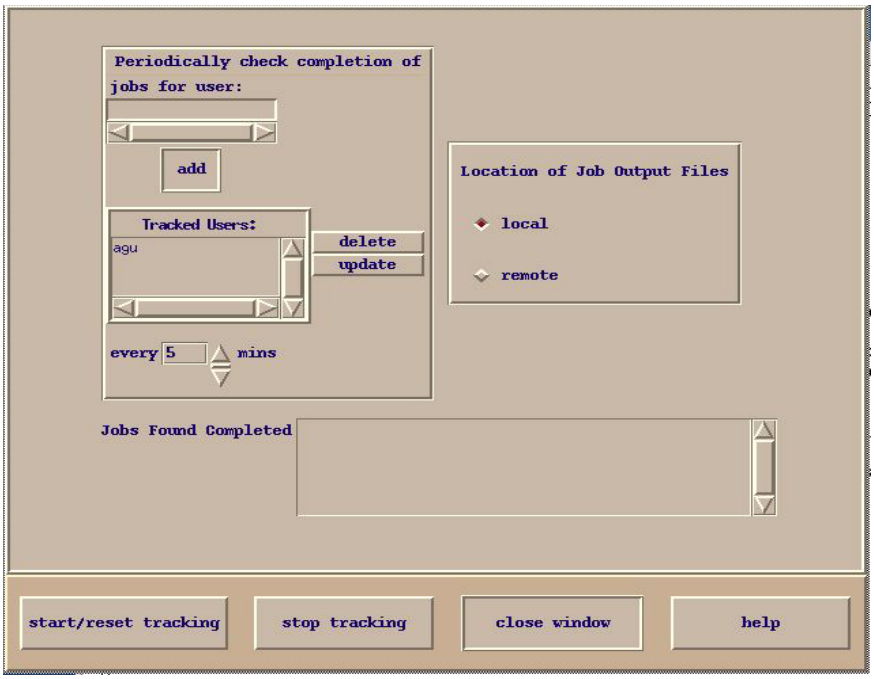
You may specify as many or as few selection criteria as you wish. When you have completed your selection, click on the *Select Jobs* button above the HOSTS panel to have `xpbs` refresh the display with the jobs that match your selection criteria. The selected criteria will remain in effect until you change them again. If you exit `xpbs`, you will be prompted if you wish to save your configuration information; this includes the job selection criteria.

6.5 Using `xpbs` TrackJob Feature

The `xpbs` command includes a feature that allows you to track the progress of your jobs. When you enable the *Track Job* feature, `xpbs` will monitor your jobs, looking for the output files that signal completion of the job. The *Track Job* button will flash red on the `xpbs` main display, and if you then click it, `xpbs` will display a list of all completed jobs (that you were previously tracking). Selecting one of those jobs will launch a window containing the standard output and standard error files associated with the job.

Important: The Track Job feature is not currently available on Windows.

To enable `xpbs` job tracking, click on the *Track Job* button at the top center of the main `xpbs` display. Doing so will bring up the Track Job dialog box shown below.



From this window you can name the users whose jobs you wish to monitor. You also need to specify where you expect the output files to be: either local or remote (e.g. will the files be retained on the Server host, or did you request them to be delivered to another host?). Next, click the *start/reset tracking button* and then the *close window* button. Note that you can disable job tracking at any time by clicking the *Track Job* button on the main `xpbs` display, and then clicking the *stop tracking* button.

Chapter 7
Working With PBS Jobs

This chapter introduces the reader to various commands useful in working with PBS jobs. Covered topics include: modifying job attributes, holding and releasing jobs, sending messages to jobs, changing order of jobs within a queue, sending signals to jobs, and deleting jobs. In each section below, the command line method for accomplishing a particular task is presented first, followed by the `xpbs` method.

7.1 Modifying Job Attributes

At some point you may need to change an attribute on a job previously submitted. Perhaps you made a mistake on the resource requirements, or perhaps a previous job ran out of time, so you want to add more time to a queued job before it starts running. Whatever the reason, PBS provides the **`qalter`** command. Most attributes can be changed by the owner of the job (or a manager or operator) while the job is still queued. However, once a job begins execution, certain resource limits cannot be changed by anyone. These include:

cputime
walltime
number of CPUs
memory (both `mem` and `vmem`)
nodes

Furthermore, job resource limits (such as `cput` or `walltime`), for a running job can be modified by a PBS Manager or Operator, but per-process limits, such as `pcput` and `pmem`, for a running job cannot be modified regardless of privilege.

The usage syntax for `qalter` is:

```
qalter job-resources job-list
```

The *`job-resources`* are the same option and value pairs used on the `qsub` command line. Only those attributes listed as options on the command will be modified. If any of the specified attributes cannot be modified for a job for any reason, none of that job's attributes will be modified.

The following examples illustrate how to use the `qalter` command. First we list all the jobs of a particular user. Then we modify two attributes as shown (increasing the wall-clock time from 13 to 20 minutes, and changing the job name from "airfoil" to "engine"):

```
qstat -u barry

                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --

qalter -l walltime=20:00 -N engine 54

qstat -a 54

                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq engine   --   --   1  -- 0:20 Q  --
```

To alter a job attribute via `xpbs`, first select the job(s) of interest, and the click on *modify* button. Doing so will bring up the *Modify Job Attributes* dialog box. From this window you may set the new values for any attribute you are permitted to change. Then click on the *confirm modify* button at the lower left of the window.

The qalter command can be used on job arrays, but not on subjobs or ranges of subjobs.

7.2 Holding and Releasing Jobs

PBS provides a pair of commands to hold and release jobs. To hold a job is to mark it as ineligible to run until the hold on the job is "released".

The **`qhold`** command requests that a Server place one or more holds on a job. A job that has a hold is not eligible for execution. There are three types of holds: *user*, *operator*, and *system*. A user may place a *user* hold upon any job the user owns. An "operator", who is a user with "operator privilege", may place either an *user* or an *operator* hold on any job. The PBS Manager may place any hold on any job. The usage syntax of the `qhold` command is:

```
qhold [ -h hold_list ] job_identifier ...
```

The *`hold_list`* defines the type of holds to be placed on the job. The *`hold_list`* argument is a string consisting of one or more of the letters `u`, p, `o`, or `s` in any combination, or the letter `n`. The hold type associated with each letter is:

Letter	Meaning
n	none - no hold type specified
u	user - the user may set and release this hold type
p	password - set if job fails due to a bad password; can be unset by the user
o	operator; require operator privilege to unset
s	system - requires manager privilege to unset

If no `-h` option is given, the *user* hold will be applied to the jobs described by the *`job_identifier`* operand list. If the job identified by *`job_identifier`* is in the queued, held, or waiting states, then all that occurs is that the hold type is added to the job. The job is then placed into held state if it resides in an execution queue.

If the job is in running state, then the following additional action is taken to interrupt the execution of the job. If checkpoint / restart is supported by the host system, requesting a hold on a running job will cause (1) the job to be checkpointed, (2) the resources assigned to the job to be released, and (3) the job to be placed in the held state in the execution queue. If checkpoint / restart is not supported, `qhold` will only set the requested hold attribute. This will have no effect unless the job is requeued with the `qrerun` command.

The qhold command can be used on job arrays, but not on subjobs or ranges of subjobs. On job arrays, the qhold command can be applied only in the ‘Q’, ‘B’ or ‘W’ states. This will put the job array in the ‘H’, held, state. If any subjobs are running, they will run to completion. Job arrays cannot be moved in the ‘H’ state if any subjobs are running.

Checkpointing is not supported for job arrays. Even on systems that support checkpointing, no subjobs will be checkpointed -- they will run to completion.

Similarly, the **`qrls`** command releases a hold on a job. However, the user executing the `qrls` command must have the necessary privilege to release a given hold. The same rules apply for releasing a hold as exist for setting a hold.

The qrls command can only be used with job array objects, not with subjobs or ranges. The job array will be returned to its pre-hold state, which can be either ‘Q’, ‘B’, or ‘W’.

The usage syntax of the `qrls` command is:

```
qrls [ -h hold_list ] job_identifier ...
```

The following examples illustrate how to use both the `qhold` and `qrls` commands. Notice that the state (“S”) column shows how the state of the job changes with the use of these two commands.

```
qstat -a 54
                                                          Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --

qhold 54
qstat -a 54
                                                          Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 H  --

qrls -h u 54
qstat -a 54
                                                          Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --
```

If you attempted to release a hold on a job which is not on hold, the request will be ignored. If you use the `qrls` command to release a hold on a job that had been previously running, and subsequently checkpointed, the hold will be released, and the job will return to the queued (Q) state (and be eligible to be scheduled to run when resources come available).

To hold (or release) a job using `xpbs`, first select the job(s) of interest, then click the *hold* (or *release*) button.

7.3 Deleting Jobs

PBS provides the **`qdel`** command for deleting jobs from the system. The `qdel` command deletes jobs in the order in which their job identifiers are presented to the command. A job that has been deleted is no longer subject to management by PBS. A batch job may be deleted by its owner, a PBS operator, or a PBS administrator.

```
qdel 17
```

To delete a job using `xpbs`, first select the job(s) of interest, then click the *delete* button.

7.4 Sending Messages to Jobs

To send a message to a job is to write a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job. Such messages can be written using the **`qmsg`** command.

> **Important:** A message can only be sent to running jobs.

The usage syntax of the `qmsg` command is:

```
qmsg [ -E ][ -O ] message_string job_identifier
```

The `-E` option writes the message into the error file of the specified job(s). The `-O` option writes the message into the output file of the specified job(s). If neither option is specified, the message will be written to the error file of the job.

The first operand, *`message_string`*, is the message to be written. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the job's file. All remaining operands are *`job_identifiers`* which specify the jobs to receive the message string. For example:

```
qmsg -E "hello to my error (.e) file" 55
qmsg -O "hello to my output (.o) file" 55
qmsg "this too will go to my error (.e) file" 55
```

To send a message to a job using `xpbs`, first select the job(s) of interest, then click the *msg* button. Doing so will launch the *Send Message to Job* dialog box. From this window, you may enter the message you wish to send and indicate whether it should be written to the standard output or the standard error file of the job. Click the *Send Message* button to complete the process.

7.5 Sending Signals to Jobs

The **`qsig`** command requests that a signal be sent to executing PBS jobs. The signal is sent to the session leader of the job. Usage syntax of the `qsig` command is:

```
qsig [  -s signal ]  job_identifier
```

If the `-s` option is not specified, `SIGTERM` is sent. If the `-s` option is specified, it declares which *`signal`* is sent to the job. The *`signal`* argument is either a signal name, e.g. `SIGKILL`, the signal name without the `SIG` prefix, e.g. `KILL`, or an unsigned signal number, e.g. 9. The signal name `SIGNULL` is allowed; the Server will send the signal 0 to the job which will have no effect. Not all signal names will be recognized by `qsig`. If it doesn't recognize the signal name, try issuing the signal number instead. The request to signal a batch job will be rejected if:

The user is not authorized to signal the job.
The job is not in the running state.
The requested signal is not supported by the execution host.
The job is exiting.

Two special signal names, "suspend" and "resume", (note, all lower case), are used to suspend and resume jobs. When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. Manager or operator privilege is required to suspend or resume a job.

The three examples below all send a signal 9 (SIGKILL) to job 34:

```
qsig -s SIGKILL 34
qsig -s KILL 34
qsig -s 9 34
```

Important: On most UNIX systems the command "`kill -l`" (that's 'minus ell') will list all the available signals.

To send a signal to a job using `xpbs`, first select the job(s) of interest, then click the *signal* button. Doing so will launch the *Signal Running Job* dialog box.

From this window, you may click on any of the common signals, or you may enter the signal number or signal name you wish to send to the job. Click the *Signal* button to complete the process.

7.6 Changing Order of Jobs Within Queue

PBS provides the **qorder** command to change the order (or reorder) two jobs. To order two jobs is to exchange the jobs' positions in the queue or queues in which the jobs resides. The two jobs must be located at the same Server, and both jobs must be owned by the user. No attribute of the job (such as priority) is changed. The impact of changing the order within the queue(s) is dependent on local job scheduling policy; contact your systems administrator for details for details.

Important: A job in the running state cannot be reordered.

Usage of the `qorder` command is:

```
qorder job_identifier1 job_identifier2
```

Both operands are *job_identifiers* which specify the jobs to be exchanged.

```
qstat -u barry
                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq twinkie  --   --   1  -- 0:20 Q  --
63.south barry  workq airfoil  --   --   1  -- 0:13 Q  --

qorder 54 63
qstat -u barry
                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
63.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
54.south barry  workq twinkie  --   --   1  -- 0:20 Q  --
```

To change the order of two jobs using `xpbs`, select the two jobs, and then click the *order* button.

The qorder command can only be used with job array objects, not on subjobs or ranges. This will change the queue order of the job array in association with other jobs or job arrays in the queue.

7.7 Moving Jobs Between Queues

PBS provides the **qmove** command to move jobs between different queues (even queues on different Servers). To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue.

Important: A job in the running state cannot be moved.

The usage syntax of the qmove command is:

```
qmove destination job_identifier(s)
```

The first operand is the new destination for

queue
@server
queue@server

If the *destination* operand describes only a queue, then qmove will move jobs into the queue of the specified name at the job's current Server. If the *destination* operand describes only a Server, then qmove will move jobs into the default queue at that Server. If the *destination* operand describes both a queue and a Server, then qmove will move the jobs into the specified queue at the specified Server. All following operands are *job_identifiers* which specify the jobs to be moved to the new *destination*.

To move jobs between queues or between Servers using xpbs, select the job(s) of interest, and then click the move button. Doing so will launch the Move Job dialog box from which you can select the queue and/or Server to which you want the job(s) moved.

The qmove command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the 'Q', 'H', or 'W' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a qstat on the server from which the job array was moved will not show the job array. A qstat on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

Chapter 8
Advanced PBS Features

This chapter covers the less commonly used commands and more complex topics which will add substantial functionality to your use of PBS. The reader is advised to read chapters 5 - 7 of this manual first.

8.1 UNIX Job Exit Status

On UNIX systems, the exit status of a job is normally the exit status of the shell executing the job script. If a user is using `csh` and has a `.logout` file in the home directory, the exit status of `csh` becomes the exit status of the last command in `.logout`. This may impact the use of job dependencies which depend on the job's exit status. To preserve the job's exit status, the user may either remove `.logout` or edit it as shown in this example:

```
set EXITVAL = $status
[  .logout's original content ]
exit $EXITVAL
```

Doing so will ensure that the exit status of the job persists across the invocation of the `.logout` file.

The exit status of a job array is determined by the status of each of the completed subjobs.

It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob. See "Job Array Exit Status" on page 149.

8.2 Changing UNIX Job umask

The "`-W umask=`*nnn*" option to `qsub` allows you to specify, on UNIX systems, what `umask` PBS should use when creating and/or coping your `stdout` and `stderr` files, and any other files you direct PBS to transfer on your behalf.

Important: This feature does not apply to Windows.

The following example illustrates how to set your `umask` to 022 (i.e. to have files created with write permission for owner only: `-rw-r--r--`).

```
qsub -W umask=022 my_job
```

```
#PBS -W umask=022
...
```

8.3 Requesting qsub Wait for Job Completion

The "`-W block=true`" option to `qsub` allows you to specify that you want `qsub` to wait for the job to complete (i.e. "block") and report the exit value of the job. If job submission fails, no special processing will take place. If the job is successfully submitted, `qsub` will block until the job terminates or an error occurs.

If `qsub` receives one of the signals: `SIGHUP`, `SIGINT`, or `SIGTERM`, it will print a message and then exit with the exit status 2. If the job is deleted before running to completion, or an internal PBS error occurs, an error message describing the situation will be printed to this error stream and `qsub` will exit with an exit status of 3. Signals `SIGQUIT` and `SIGKILL` are not trapped and thus will immediately terminate the `qsub` process, leaving the associated job either running or queued. If the job runs to completion, `qsub` will exit with the exit status of the job. (See also section 8.1 "UNIX Job Exit Status" on page 105 for further discussion of the job exit status.)

For job arrays, blocking qsub waits until the entire job array is complete, then returns the exit status of the job array.

8.4 Specifying Job Dependencies

PBS allows you to specify dependencies between two or more jobs. Dependencies are useful for a variety of tasks, such as:

1. Specifying the order in which jobs in a set should execute
2. Requesting a job run only if an error occurs in another job
3. Holding jobs until a particular job starts or completes execution

The "`-W depend=dependency_list`" option to `qsub` defines the dependency between multiple jobs. The *dependency_list* has the format:

```
type[ :argument[ :argument...][ ,type:argument...]
```

Job dependencies are supported:

- between job arrays and job arrays
- between job arrays and jobs
- between jobs and job arrays

The *argument* is either a numeric count or a PBS job identifier according to type. If *argument* is a count, it must be greater than 0. If it is a job identifier and not fully specified in the form `seq_number.server.name`, it will be expanded according to the default Server rules which apply to job identifiers on most commands. If *argument* is null (the proceeding colon need not be specified), the dependency of the corresponding type is cleared (unset).

after:jobid[:jobid...]
: This job may be scheduled for execution at any point after all jobs specified have started execution.

afterok:jobid[:jobid...]
: This job may be scheduled for execution only after all jobs specified have terminated with no errors. See the `csh` warning under "User's PBS Environment" on page 18.

afternotok:jobid[:jobid...]
: This job may be scheduled for execution only after all jobs specified have terminated with errors. See previous `csh` warning.

afterany:jobid[:jobid...]

This job may be scheduled for execution after jobs *jobid* have terminated with any exit status (i.e. either with or without errors).

on:count — This job may be scheduled for execution after *count* dependencies on other jobs have been satisfied. This form is used in conjunction with one of the *before* dependencies.

before:jobid[:jobid...]
When this job has begun execution, then jobs *jobid*... may begin.

beforeok:jobid[:jobid...]
If this job terminates execution without errors, then jobs *jobid*... may begin. See previous `csh` warning.

beforenotok:jobid[:jobid...]
If this job terminates execution with errors, then jobs *jobid*... may begin. See previous `csh` warning.

beforeany:jobid[:jobid...]
When this job terminates execution (with any termination status, either with or without errors), the specified jobs *jobid*... may begin. If any of the before forms are used, the jobs referenced by *jobid* must have been submitted with a dependency type of on. If any of the before forms are used, the jobs referenced by *jobid* must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

Error processing of the existence, state, or condition of the job on which the newly submitted job depends is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the new job will be deleted by the Server. Email will be sent to the job submitter stating the error.

The following examples illustrate the most common uses for job dependencies.

Suppose you have three jobs (job1, job2, and job3) and you want job3 to start *after* job1 and job2 have *ended*. The first example below illustrates the options you would use on the

`qsub` command line to implement these job dependencies.

```
qsub job1
16394.jupiter
qsub job2
16395.jupiter
qsub -W depend=afterany:16394:16395 job3
16396.jupiter
```

As another example, suppose instead you want job2 to start *only if* job1 ends with no errors (i.e. it exits with a no error status):

```
qsub job1
16397.jupiter
qsub -W depend=afterok:16397 job2
16396.jupiter
```

Similarly, you can use before dependencies, as the following example exhibits. Note that unlike after dependencies, before dependencies require the use of the "on" dependency (see discussion on page 108 above).

```
qsub -W depend=on:2 job1
16397.jupiter
qsub -W depend=beforeany:16397 job2
16398.jupiter
qsub -W depend=beforeany:16397 job3
16398.jupiter
```

You can use `xpbs` to specify job dependencies as well. On the *Submit Job* window, in the other options section (far left, center of window) click on one of the three dependency buttons: *after depend*, *before depend*, or *concurrency*. These will launch a *Dependency* window in which you will be able to set up the dependencies you wish.

8.5 Delivery of Output Files

To transfer output files or to transfer staged-in or staged-out files to/from a remote destination, PBS uses either `rcp` or `scp` depending on the configuration options. (PBS includes a version of the `rcp`(1) command from the BSD 4.4 lite distribution, renamed

pbs_rcp.) This version of rcp is provided because it, unlike some rcp implementations, always exits with a non-zero exits status for any error. Thus MOM knows if the file was delivered or not. Fortunately, the secure copy program, scp, is also based on this version of rcp and exits with the proper status code.

If using rcp, the copy of output or staged files can fail for (at least) two reasons.

1. The user lacks authorization to access the specified system. (See discussion in "User's PBS Environment" on page 18.)

2. Under UNIX, if the user's .cshrc outputs any characters to standard output, e.g. contains an echo command, pbs_rcp will fail.

If using *Secure Copy (*scp), then PBS will first try to deliver output or stage-in/out files using scp. If scp fails, PBS will try again using rcp (assuming that scp might not exist on the remote host). If rcp also fails, the above cycle will be repeated after a delay, in case the problem is caused by a temporary network problem. All failures are logged in MOM's log, and an email containing the errors is sent to the job owner.

For delivery of output files on the local host, PBS uses the cp command (UNIX) or the xcopy command (Windows). Local and remote Delivery of output may fail for the following additional reasons:

1. A directory in the specified destination path does not exist.
2. A directory in the specified destination path is not searchable by the user.
3. The target directory is not writable by the user.

8.6 Input/Output File Staging

File staging is a way to specify which files should be copied onto the execution host before the job starts, and which should be copied off the execution host when it completes. (For file staging under Globus, see "PBS File Staging through GASS" on page 126.) The "-W stagein=file_list" and "-W stageout=file_list" options to qsub specifies which files are staged (copied) in before the job starts or staged out after the job completes execution. On completion of the job, all staged-in and staged-out files are removed from the execution system. The *file_list* is in the form:

```
local_file@hostname:remote_file[,...]
```

regardless of the direction of the copy. Note that the '@' character is used for separating the local specification from the remote specification. The name *local_file* is the name of the file on the system where the job executes. It may be an absolute path or relative to the home directory of the user. The name *remote_file* is the destination name on the host specified by hostname. The name may be absolute or relative to the user's home directory on the destination host. Thus for stage-in, the direction of travel is:

```
local_file◄──  remote_host:remote_file
```

and for stage out, the direction of travel is:

```
local_file──►  remote_host:remote_file
```

Note that all relative paths are relative to the user's home directory on the respective hosts. The following example shows how to stage-in a file named `grid.dat` located in the directory `/u/user1` of the computer called `server`. The staged-in file is requested to be placed relative to the users home directory under the name of `dat1`. (Note that the example uses UNIX-style path separators "/".)

```
#PBS -W stagein=dat1@server:/u/user1/grid.dat
...
```

Note that under Windows special characters such as spaces, backslashes (\), colons (:), and drive letter specifications are valid pathnames. For example, the following will stage-in the grid.dat file at hostB to a local file ("dat1") on drive C.:

```
qsub -W stagein=C:\temp\dat1@hostB:grid.dat
```

In windows the stagein and stageout string must be contained in double quotes when using ^array_index^.

Example of a stagein:

```
qsub -W stagein="foo.^array_index^
   @host-3:C:\WINNT\Temp\foo.^array_index^"
      -J 1-5 stage_script
```

Example of a stageout :

```
qsub -W stageout="C:\WINNT\Temp\foo.^array_index^
   @vmwhost-3:Q:\pbsuser31\foo.^array_index^_out"
      -J 1-5 stage_script
```

PBS uses `rcp` or `scp` (or `cp` if the remote host is the local host) to perform the transfer. Hence, stage-in and stage-out are just:

```
rcp -r remote_host:remote_file local_file
rcp -r local_file remote_host:remote_file
```

As with `rcp`, the *`remote_file`* and *`local_file`* portions for both stage-in and stage-out may name a directory. For stage-in, if *`remote_file`* is a directory, then *`local_file`* must also be a directory. Likewise, for stage out, if *`local_file`* is a directory, then *`remote_file`* must be a directory. If *`local_file`* on a stage out directive is a directory, that directory on the execution host, including all files and subdirectories, will be copied. At the end of the job, the directory, including all files and subdirectories, will be deleted. Users should be aware that this may create a problem if multiple jobs are using the same directory. The same requirements and hints discussed above in regard to delivery of output apply to staging files in and out. Wildcards should not be used in either the *`local_file`* or the *`remote_file`* name. PBS does not expand the wildcard character on the local system. If wildcards are used in the *`remote_file`* name, since `rcp` is launched by `rsh` to the remote system, the expansion will occur. However, at job end, PBS will attempt to delete the file whose name actually contains the wildcard character and will fail to find it. This will leave all the staged-in files in place (undeleted).

File staging is supported for job arrays, where a variable is used to specify the subjob index. See “Staging” on page 137.

Using `xpbs` to set up file staging directives may be easier than using the command line. On the *Submit Job* window, in the miscellany options section (far left, center of window) click on the *file staging* button. This will launch the *File Staging* dialog box (shown below) in which you will be able to set up the file staging you desire.

The *File Selection Box* will be initialized with your current working directory. If you wish to select a different directory, double-click on its name, and `xpbs` will list the contents of the new directory in the *File Selection Box*. When the correct directory is displayed, simply click on the name of the file you wish to stage (in or out). Its name will be written in the *File Selected* area.

Next, click either of the *Add file selected...* buttons to add the named file to the stage-in or stage-out list. Doing so will write the file name into the corresponding area on the lower half of the *File Staging* window. Now you need to provide location information. For stage-in, type in the path and filename where you want the named file placed. For stage-out, specify the hostname and pathname where you want the named file delivered. You may repeat this process for as many files as you need to stage.

When you are done selecting files, click the *OK* button.

8.7 File Staging Between UNIX and Windows

If the files being staged are being copied from a UNIX host to a Windows host, or the other way around, ensure that all pathnames in the stage-in / stage-out specification are absolute, not relative. For example, this directive to `qsub`,

```
-W stagein="\Documents and Settings\user\inputfile@unix-host:/a/b/file"
```

will copy the input file `/a/b/file` from `unix-host` over to `\Documents and Settings\user\inputfile` on the Windows host before the job is sent to that host.

Important: If you're staging in or staging out a directory to the Windows host, the destination pathname must be an existing directory.

8.8 The `pbsdsh` Command

The **`pbsdsh`** command allows you to distribute and execute a task on each of the nodes assigned to your job. (`pbsdsh` uses the PBS Task Manager API, see `tm`(3), to distribute the program on the allocated nodes.)

Important: The `pbsdsh` command is not available under Windows.

Usage of the `pbsdsh` command is:

```
pbsdsh [ -c N | -n N] [ -o] [ -s] [ -v] program args
```

The available options are:

`-c N` The program is spawned on the first *N* nodes allocated. If the value of *N* is greater than the number of nodes, it will "wrap" around, running multiple copies on the nodes. This option is mutually exclusive with `-n`.

`-n N` The program is spawned on a single node which is the *N*-th node allocated. This option is mutual exclusive with `-c`.

`-o` The program will not wait for the tasks to finish.

`-s` If this option is given, the program is run sequentially on each node, one after the other.

`-v` Verbose output about error messages and task exit status is produced.

When run without the `-c` or the `-n` option, `pbsdsh` will spawn the program on all nodes allocated to the PBS job. The execution take place concurrently--all copies of the task execute at (about) the same time.

The following example shows the `pbsdsh` command inside of a PBS batch job. The options indicate that the user wants `pbsdsh` to run the *`myapp`* program with one argument (*`app-arg1`*) on all four nodes allocated to the job (i.e. the default behavior).

```
#PBS -l nodes=4
#PBS -l walltime=1:00:00

pbsdsh myapp app-arg1
```

8.9 Advance Reservation of Resources

An *Advance Reservation* is a set of resources with availability limited to a specific user (or group of users), a specific start time, and a specified duration. Advance Reservations are implemented in PBS by a user submitting a reservation with the `pbs_rsub` command. PBS will then confirm that the reservation can be met (or else reject the request). Once the scheduler has confirmed the reservation, the queue that was created to support this reser-

vation will be enabled, allowing jobs to be submitted to it. The queue will have an user level access control list set to the user who submitted the reservation and any other users the owner specified. The queue will accept jobs in the same manner as normal queues. When the reservation start time is reached, the queue will be started. Once the reservation is complete, any jobs remaining in the queue (running or not) will be deleted, and the reservation removed from the Server.

When a reservation is requested and confirmed, it means that a check was made to see if the reservation would conflict with currently running jobs, other confirmed reservations, and dedicated time. A reservation request that fails this check is denied by the Scheduler. If the submitter did not indicate that the submission command should wait for confirmation or rejection (`-I` option), he will have to periodically query the Server about the status of the reservation (via `pbs_rstat`) or wait for a mail message regarding its denial or confirmation.

Important: Hosts/nodes that have been configured to accept jobs only from a specific queue (node-queue restrictions) cannot be used for advance reservations. See your local PBS Administrator to determine if this affects your site.

8.9.1 Submitting a PBS Reservation

The **`pbs_rsub`** command is used to request a reservation of resources. If the request is granted, PBS provides for the requested resources to be available for use during the specified future time interval. A queue is dynamically allocated to service a *confirmed* reservation. Users who are listed as being allowed to run jobs using the resources of this reservation will submit their jobs to this queue via the standard `qsub` command.

Although a confirmed resources reservation will accept jobs into its queue at any time, the scheduler is not allowed to schedule jobs from the queue before the reservation period arrives. Once the reservation period arrives, these jobs will begin to run but they will not in aggregate use up more resources than the reservation requested.

The `pbs_rsub` command returns an ID string to use in referencing the reservation and an indication of its current status. The actual specification of resources is done in the same way as it is for submission of a job. Following is a list and description of options to the `pbs_rsub` command.

-R datetime — Specifies reservation starting time. If the reservation's end time and duration are the only times specified, this start time is calculated. The datetime argument adhers to the POSIX time specification:

```
[[[[CC]YY]MM]DD]hhmm[.SS]
```

If the day, `DD`, is not specified, it will default to today if the time `hhmm` is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a reservation having a specification `-R 1110` at 11:15am, it will be interpreted as being for 11:10am tomorrow. If the month portion, `MM`, is not specified, it defaults to the current month provided that the specified day `DD`, is in the future. Otherwise, the month will be set to next month. Similar comments apply to the two other optional, left hand components.

-E datetime — Specifies the reservation end time. See the `-R` flag for a description of the datetime string. If start time and duration are the only times specified, the end time value is calculated.

-D timestring — Specifies reservation duration. Timestring can either be expressed in seconds of walltime or it can be expressed as a colon delimited timestring e.g. HH:MM:SS or MM:SS. If the start time and end time are the only times specified, this duration time is calculated.

-m mail_point — Specifies whether mail is sent to user_list and when. The argument mail_point is a string. It can be either "n", for no mail, or a string composed of any combination of "a", "b", "e", or "c". Default is "ac". Must be enclosed in double quotes.

"n"	do not send mail
"a"	notify if the reservation is terminated for any reason
"b"	notify when the reservation period begins
"e"	notify when the reservation period ends
"c"	notify when the reservation is confirmed

-M mail_list
: Specifies the list of users to whom the Server will attempt to send a mail message whenever the reservation transitions to one of the mail states specified in the `-m` option. Default: reservation's owner

-u user_list
: Specifies a comma separated list of entries of the form: `user@host`. Entries on this list are used by the Server in conjunction with an ordered set of rules to associate a user name with the reservation.

-g group_list
: Specifies a comma separated list of entries of the form: `group@host` names. Entries on this list are used by the Server in conjunction with an ordered set of rules to associate a group name with the reservation.

-U auth_user_list
: Specifies a comma separated list of entries of the form: `[+|-]user@host`. These are the users who are allowed (+) or denied (-) permission to submit jobs to the queue associated with this reservation. This list becomes the `acl_users` attribute for the reservation's queue.

-G auth_group_list
: Specifies a comma separated list of entries of the form: `[+|-]group_name`. Entries on this list help control the enqueuing of jobs into the reservation's queue. Jobs owned by members belonging to these groups are either allowed (+) or denied (-) entry into the queue. Any group on the list is to be interpreted in the context of the Server's host not the context of the host from which `qsub` was submitted. This list becomes the `acl_groups` list for the reservation's queue.

-H auth_host_list
: Specifies a comma separated list of entries of the form: `[+|-]hostname`. These entries help control the enqueuing of jobs into the reservation's queue by allowing (denying) jobs submitted from these hosts. This list becomes the `acl_hosts` list for the reservation's queue.

-N reservation_name
: Declares a name for the reservation. The name specified may be up to 15 characters in length. It must consist of printable, non-white space characters with the first character alphabetic.

-l resource_list
: Specifies a list of resources required for the reservation. These resources will be used for the limits on the queue that's dynamically created to service the reservation. The aggregate amount of resources for currently running jobs from this queue will not exceed these resource limits. In addition, the queue inherits the value of any resource limit set on the Server if the reservation request itself is silent about that resource.

-I seconds
: Interactive mode is specified if the submitter wants to wait for an answer to the request. The `pbs_rsub` command will block, up to the number of seconds specified, while waiting for the scheduler to either confirm or deny the reservation request. A negative number of seconds may be specified and is interpreted to mean: if the confirm/deny decision isn't made in the number of seconds specified, automatically delete the reservation request from the system. If automatic deletion isn't being requested and if the scheduler doesn't make a decision in the specified number of seconds, the command will return the ID string for the reservation and show the status as *unconfirmed*. The requester may periodically issue the `pbs_rstat` command with ID string as input to monitor the reservation's status.

-W other-attributes=value...
: This allows a site to define any extra attribute on the reservation.

The following example shows the submission of a reservation asking for 1 node, 30 minutes of wall-clock time, and a start time of 11:30. Note that since an end time is not specified, PBS will calculate the end time based on the reservation start time and duration.

```
pbs_rsub -l nodes=1 -R 1130 -D 30:00:00
R226.south UNCONFIRMED
```

A reservation queue named "R226" was created on the local PBS Server. Note that the reservation is currently *unconfirmed*. Email will be sent to the reservation owner either confirming the reservation, or rejecting it. Upon confirmation, the owner of the reservation can submit jobs against the reservation using the `qsub` command, naming the reser-

vation queue on the command line with the -q option, e.g.:

```
qsub -q R226 aims14
299.south
```

Important: The ability to submit, query, or delete advance reservations using the xpbs GUI is not available

8.9.2 Waiting for Confirmation

When the user requests an advance reservation of resources via the pbs_rsub command, an option ("-I n") is available to wait for confirmation response. The value "n" that is specified is taken as the number of seconds that the command is willing to wait. This value can be either positive or negative. A non-negative value means that the Server/scheduler response is needed in "n or less" seconds. After that time the submitter will need to use pbs_rstat or some other means to discern success or failure of the request. For a negative value, the command will wait up to "n" seconds for the request to be either confirmed or denied. If the response does not come back in "n" or fewer seconds, the Server will automatically delete the request from the system.

8.9.3 Showing Reservation Status

The **pbs_rstat** command is used to show the status of all the reservations on the PBS Server. There are three different output formats: brief, short (default), and long. The following examples illustrate these three options.

The short option (-S) will show all the reservations in a short concise form. (This is the default display if no options are given.) The information provided is the identifier of the reservation, name of the queue that got created for the reservation, user who owns the reservation, the state, the start time, duration in seconds, and the end time.

```
pbs_rstat -S
Name Queue User   State   Start   / Duration / End
-------------------------------------------------------
R226 R226  user1  CO  Today 11:30 / 1800 /  Today 12:00
R302 R302  barry  CO  Today 15:50 / 1800 /  Today 16:20
R304 R304  user1  CO  Today 15:46 / 1800 /  Today 16:16
```

The full option (`-f`) will print out the name of the reservation followed by all the attributes of the reservation.

```
pbs_rstat -f R226
Name: R226.south
Reserve_Owner = user1@south
reserve_type = 2
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Fri Aug 24 11:30:00 2004
reserve_end = Fri Aug 24 12:00:00 2004
reserve_duration = 1800
queue = R226
Resource_List.ncpus = 1
Resource_List.neednodes = 1
Resource_List.nodect = 1
Resource_List.nodes = 1
Resource_List.walltime = 00:30:00
Authorized_Users = user1@south
server = south
ctime = Fri Aug 24 06:30:53 2004
mtime = Fri Aug 24 06:30:53 2004
Variable_List = PBS_O_LOGNAME=user1,PBS_O_HOST=south
euser = user1
egroup = pbs
```

The brief option (`-B`) will only show the identifiers of all the reservations:

```
pbs_rstat -B
Name: R226.south
Name: R302.south
Name: R304.south
```

8.9.4 Delete PBS Reservations

The **`pbs_rdel`** command deletes reservations in the order in which their reservation identifiers are presented to the command. A reservation may be deleted by its owner, or a PBS operator/manager. Note that when a reservation is deleted, all jobs belonging to the reservation are deleted as well, regardless of whether or not they are currently running.

```
pbs_rdel R304
```

8.9.5 Accounting

Accounting records for advance resource reservations are available in the Server's job accounting file. The format of such records closely follows the format that exists for job records. In addition, any job that belongs to an advance reservation will have the reservation ID recorded in the accounting records for the job.

8.9.6 Access Control

A site administrator can inform the Server as to those hosts, groups, and users whose advance resource reservation requests are (or are not) to be considered. The philosophy in this regard is same as that which currently exists for jobs.

In a similar vein, the user who submits the advance resource reservation request can specify to the system those other parties (user(s) or group(s)) that are authorized to submit jobs to the reservation-queue that's to be created.

When this queue is instantiated, these specifications will supply the values for the queue's user/group access control lists. Likewise, the party who submits the reservation can, if desired, control the username and group name (at the Server) that the Server associates with the reservation.

8.10 Checkpointing SGI MPI Jobs

Under Irix 6.5 and later, MPI parallel jobs as well as serial jobs can be checkpointed and restarted on SGI systems provided certain criteria are met. SGI's checkpoint system call cannot checkpoint processes that have open sockets. Therefore it is necessary to tell mpirun to not create or to close an open socket to the array services daemon used to start the parallel processes. One of two options to `mpirun` must be used:

-cpr This option directs mpirun to close its connection to the array services daemon when a checkpoint is to occur.

-miser This option directs mpirun to directly create the parallel process rather than use the array services. This avoids opening the socket connection at all.

The `-miser` option appears the better choice as it avoids the socket in the first place. If the `-cpr` option is used, the checkpoint will work, but will be slower because the socket connection must be closed first. Note that interactive jobs or MPMD jobs (more than one executable program) can not be checkpointed in any case. Both use sockets (and TCP/IP)

to communicate, outside of the job for interactive jobs and between programs in the MPMD case.

8.11 Using Comprehesive System Accounting on SGI Altix Machines

PBS supports Comprehensive System Accounting (CSA) on SGI Altix machines that are running SGI's Pro Pack 2.4, 3.0, 3.2 or 4.0 and have the Linux job container facility available. CSA provides accounting information about user jobs, called user job accounting.

CSA works the same with and without PBS. To run user job accounting, either the user must specifiy the file to which raw accounting information will be written, or an environment variable must be set. The environment variable is "ACCT_TMPDIR". This is the directory where a temporary file of raw accounting data is written.

To run user job accounting, the user issues the CSA command "`ja <filename>`" or, if the environment variable "ACCT_TMPDIR" is set, "`ja`". In order to have an accounting report produced, the user issues the command "`ja -<options>`" where the options specify that a report will be written and what kind. To end user job accounting, the user issues the command "`ja -t`"; the -t option can be included in the previous set of options. See the manpage on `ja` for details.

The starting and ending ja commands must be used before and after any other commands the user wishes to monitor. Here are examples of command line and a script:

On the command line:

```
qsub -N myjobname -l ncpus=1
     ja myrawfile
     sleep 50
     ja -c > myreport
     ja -t myrawfile
ctrl-D
```

Accounting data for the user's job (sleep 50) is written to myreport.

If the user creates a file foo with these commands:

```
#PBS -N myjobname
#PBS -l ncpus=1
ja myrawfile
sleep 50
ja -c > myreport
ja -t myrawfile
```

The user could run this script via qsub:

```
qsub foo
```

This does the same thing, via a script.

8.12 Globus Support

Globus is a computational software infrastructure that integrates geographically distributed computational and information resources. Jobs are normally submitted to Globus using the utility `globusrun`. When Globus support is enabled for PBS, jobs can be routed between Globus and PBS. (Contact your PBS system administrator to learn if Globus support has been enabled on your PBS systems.)

Important: Globus is currently supported on UNIX, not Windows.

8.12.1 Running Globus jobs

To submit a Globus job, users must specify the globus resource name (gatekeeper), as the following example shows:

`qsub -l site=globus:`*`globus-resource-name pbsjob`*

The `pbs_mom_globus` daemon/service must be running on the same host where the `pbs_server` is running. Be sure the `pbs_server` has a `nodes` file entry *`server-host`*`:gl` in order for Globus job status to be communicated back to the Server by `pbs_mom_globus`.

Before a user can use the Globus infrastructure to successfully send work to some execution host(s) in the Globus Grid, a valid grid-proxy certificate must be obtained by the user. The utility grid-proxy-init is used to obtain a valid grid-proxy certificate. If a user's job fails to run due to an expired proxy credential or non-existent credential, then the job will be put on hold and the user will be notified of the error by email.

8.12.2 PBS and Globusrun

If you're familiar with the `globusrun` utility, the following mappings of options from PBS to an RSL string may be of use to you:

Table 13: qsub Option vs. Globus RSL

PBS Option	Globus RSL Mapping
-l site=globus:<gatekeeper>	specifies the gatekeeper to contact
-l ncpus=yy	count=yy
-A <account_name>	project=<account_name>
-l cput=yy	maxcputime=yy (in minutes)
-l pcput=yy	maxtime=yy (in minutes)
-l walltime=yy	maxwalltime=yy (in minutes)
-l mem=zz	maxmemory=zz (in megabytes)
-o <output_path>	stdout=<local_output_path>
-e <error_path>	stderr=<local_error_path> PBS will deliver from <local_path> to user-specified <output_path> and <stderr_path>
-v <variable_list>	environment=<variable_list>, job-type=single

Table 14: `qsub` Options vs. Globus RSL

PBS Option	Globus RSL Mapping
`-l site=`globus:<gatekeeper>	specifies the gatekeeper to contact
`-l ncpus=yy`	`count=yy`
`-A` <account_name>	`project=`<account_name>
`-l cput=yy`	`maxcputime=yy` (in minutes)
`-l pcput=`yy	`maxtime=yy` (in minutes)
`-l walltime=`yy	`maxwalltime=yy` (in minutes)
`-l mem=zz`	`maxmemory=zz` (in megabytes)
`-o` <output_path>	`stdout=`<local_output_path>
`-e` <error_path>	`stderr=`<local_error_path> PBS will deliver from *local_path* to user specified output_path and stderr_path
`-v` <variable_list>	`environment=`<variable_list>, `job-type=single`

When the job gets submitted to Globus, PBS `qstat` will report various state changes according to the following mapping:

Table 15: PBS Job States vs. Globus States

PBS State	Globus State
TRANSIT (T)	PENDING
RUNNING (R)	ACTIVE
EXITING (E)	FAILED
EXITING (E)	DONE

8.12.3 PBS File Staging through GASS

The stagein/stageout feature of "Globus-aware" PBS works with Global Access to Secondary Storage (GASS) software. Given a stagein directive:

```
localfile@host:inputfile
```

PBS will take care of copying *inputfile* at *host* over to *localfile* at the executing Globus machine.

The same process is used for a stageout directive:

```
localfile@host:outputfile
```

PBS will take care of copying the *localfile* on the executing Globus host over to the *outputfile* at *host*. Globus file transfer mechanisms are used when transferring files between hosts that run Globus; otherwise, `pbs_scp`, `pbs_rcp` or `cp` is used. This means that if the *host* given in the stage-in/stage-out argument runs Globus, then Globus communication will be opened to that system.

8.12.4 Limitation

PBS does not currently support "co-allocated" Globus jobs, where two or more jobs are simultaneously run (distributed) over two or more Globus resource managers.

8.12.5 Examples

Below are some examples of using PBS with Globus. They all assume that the user has acquired a valid grid-proxy certificate beforehand, probably by running Globus' grid-proxy-init utility.

Example 1: If you want to run a single processor job on globus gatekeeper `mars.mydomain.com`, using whatever job manager is currently configured at that site, then you could create a PBS script like the following example:

```
cat job.script
#PBS -l site=globus:mars.mydomain.com
echo "`hostname`:Hello world! Globus style."
```

Upon execution, this will give the sample output:

```
mars:Hello world! Globus style.
```

Example 2: If you want PBS to submit a multi-processor job to the Globus gatekeeper `pluto.mydomain.com/jobmanager-fork` with CPU count set to 4, and shipping the architecture compatible executable, `mpitest` over to the Globus host `pluto` for execution, then compose a script and submit as follows:

```
cat job.script
#PBS -l site=' globus:pluto.domain.com:763/jobmanager-
fork:/C=US/O=Communications Package/OU=Stellar Divi-
sion/CN=shirley.com.org'
#PBS -l ncpus=4
#PBS -W stagein=mpitest@earth.domain.com:progs/mpitest

mpirun -n=4 ~/mpitest

wait
```

Upon execution, this sample script would produce the following output:

```
Process #2 of 4 on host: pluto at time: Mon Aug 29 17:39:01 2004
Process #3 of 4 on host: pluto at time: Mon Aug 29 17:39:01 2004
Process #1 of 4 on host: pluto at time: Mon Aug 29 17:39:01 2004
Process #0 of 4 on host: pluto at time: Mon Aug 29 17:39:01 2004
```

Example 3: Here is a more complicated example. Say you want to transfer and run a SGI-specific MPI job requiring 4 CPUs on host "sgi.galaxy.com") via Globus. Furthermore, let's say that host is running a different batch system, and you want the job's output returned to the submitting host. The following job script illustrates this:

```
 cat job.script
#PBS -l site=globus:sgi.galaxy.com/jobmanager-lsf,
ncpus=4
#PBS -W stagein=/u/jill/mpi_sgi@earth:progs/mpi_sgi
#PBS -W stageout=mpi_sgi.out@earth:mpi_sgi.out
mpirun -np 4 /u/jill/mpi_sgi >> mpi_sgi.out
echo "Done it"
```

Upon execution, the sample output is:

```
Done it
```

And the output of the run would have been written to the file `mpi_sgi.out`, and returned to the user's home directory on host earth, as specified.

Note: Just like a job that's completely managed by PBS, a job that's submitted to a Globus Grid through PBS can be deleted, signaled, held, released, rerun, have text appended to its output/error files, and be moved from one location to another.

8.13 Running PBS in a UNIX DCE Environment

PBS Professional includes optional support for UNIX-based DCE. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the DCE module.)

There are two `-W` options available with `qsub` which will enable a `dcelogin` context to be set up for the job when it eventually executes. The user may specify either an encrypted password or a forwardable/renewable Kerberos V5 TGT.

Specify the "`-W cred=dce`" option to `qsub` if a forwardable, renewable, Kerberos V5, TGT (ticket granting ticket) with the user as the listed principal is what is to be sent with the job. If the user has an established credentials cache and a non-expired, forwardable, renewable, TGT is in the cache, that information is used.

The other choice, "`-W cred=dce:pass`", causes the `qsub` command to interact with the user to generate a DES encryption of the user's password. This encrypted password is sent to the PBS Server and MOM processes, where it is placed in a job-specific file for later use by `pbs_mom` in acquiring a DCE login context for the job. The information is destroyed if the job terminates, is deleted, or aborts.

Important: The "`-W pwd=''`" option to `qsub` has been superseded by the above two options, and therefore should no longer be used.

Any acquired login contexts and accompanying DCE credential caches established for the job get removed on job termination or deletion.

```
qsub -Wcred=dce <other qsub options> job-script
```

Important: The "`-W cred`" option to `qsub` is not available under Windows.

8.14 Running PBS in a UNIX Kerberos Environment

PBS Professional includes optional support for Kerberos-only (i.e. no DCE) environment. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the KRB5 module.)

To use a forwardable/renewable Kerberos V5 TGT specify the "`-W cred=krb5`" option to `qsub`. This will cause `qsub` to check the user's credential cache for a valid forwardable/renewable TGT which it will send to the Server and then eventually to the execution MOM. While it's at the Server and the MOM, this TGT will be periodically refreshed until either the job finishes or the maximum refresh time on the TGT is exceeded, which ever comes first. If the maximum refresh time on the TGT is exceeded, no KRB5 services will be available to the job, even though it will continue to run.

Chapter 9

Job Arrays

This chapter describes job arrays and their use. A job array represents a collection of jobs which only differ by a single index parameter. The purpose of a job array is twofold. It offers the user a mechanism for grouping related work, making it possible to submit, query, modify and display the set as a single unit. Second, it offers a way to possibly improve performance, because the batch system can use certain known aspects of the collection for speedup.

9.1 Definitions

Subjob — Individual entity within a job array (e.g. **1234[7]**, where **1234[]** is the job array itself, and **7** is the index) which has many properties of a job as well as additional semantics (defined below)

Sequence_number — The numeric part of a job or job array identifier, e.g. **1234**

Subjob index — The unique index which differentiates one subjob from another. This must be a non-negative integer

Job array identifier — The identifier returned upon success when submitting a job array. The format is **sequence_number[]** or **sequence_number[].server.domain.com**

Job array range A set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices

9.1.1 Description

A job array is a compact representation of one or more jobs, called subjobs when part of a Job array, which have the same job script, and have the same values for all attributes and resources, with the following exceptions:

- each subjob has a unique index
- Job Identifiers of subjobs only differ by their indices
- the state of subjobs can differ

All subjobs within a job array have the same scheduling priority.

A job array is submitted through a single command which returns, on success, a "job array identifier" with a server-unique sequence number. Subjob indices are specified at submission time. These can be:

- a contiguous range, e.g. 1 through 100
- a range with a stepping factor, e.g. every second entry in 1 through 100 (1, 3, 5, ... 99)

A job array identifier can be used

- by itself to represent the set of all subjobs of the job array
- with a single index (a "job array identifier") to represent a single subjob
- with a range (a "job array range") to represent the subjobs designated by the range

9.1.2 Identifier Syntax

Job arrays have three identifier syntaxes:

- The job array object itself : 1234[].server or 1234[]
- A single subjob of a job array with index M: 1234[M].server or 1234[M]
- A range of subjobs of a job array: 1234[X-Y:Z].server or 1234[X-Y:Z]

Examples:

1234[].server.domain.com	Full job array identifier
1234[]	Short job array identifier
1234[73]	Subjob identifier of the 73rd index of job array 1234[]
1234	Error, if 1234[] is a job array
1234.server.domain.com	Error, if 1234[].server.domain.com is a job array

The sequence number (1234 in 1234[].server) is unique, so that jobs and job arrays cannot share a sequence number.

Note: Since some shells, for example csh and tcsh, read "[" and "]" as shell metacharacters, job array names and subjob names will need to be enclosed in double quotes for all PBS commands.

Example:

```
qdel "1234.myhost[5]"
qdel "1234.myhost[]"
```

Single quotes will work, except where you are using shell variable substitution.

9.2 qsub: Submitting a Job Array

To submit a job array, qsub is used with the option **-J range**, where **range** is of the form **X-Y[:Z]**. **X** is the starting index, **Y** is the ending index, and **Z** is the optional **stepping factor**. X and Y must be whole numbers, and Z must be a positive integer. Y must be greater than X. If Y is not a multiple of the stepping factor above X, (i.e. it won't be used as an index value) the highest index used will be the next below Y. For example, 1-100:2 gives 1, 3, 5, ... 99.

Blocking qsub waits until the entire job array is complete, then returns the exit status of the job array.

Interactive submission of job arrays is not allowed.

Examples:

To submit a job array of 10,000 subjobs, with indices 1, 2, 3, ... 10000:

```
$ qsub -J 1-10000 job.scr
    1234[].server.domain.com
```

To submit a job array of 500 subjobs, with indices 500, 501, 502, ... 1000:

```
$ qsub -J 500-1000 job.scr
    1235[].server.domain.com
```

To submit a job array with indices 1, 3, 5 ... 999:

```
$ qsub -J 1-1000:2 job.scr
    1236[].server.domain.com
```

9.2.1 Interactive Job Submission

Job arrays do not support interactive submission.

9.3 Job Array Attributes

Job arrays and subjobs have all of the attributes of a job. In addition, they have the following when appropriate. These attributes are read-only.

Table 16: Job Array Attributes

Name	Type	Applies To	Value
array	boolean	job array	True if item is job array
array_id	string	subjob	Subjob's job array identifier
array_index	string	subjob	Subjob's index number
array_state_count	string	job array	Similar to state_count attribute for server and queue objects. Lists number of subjobs in each state.
array_indices_remaining	string	job array	List of indices of subjobs still queued. Range or list of ranges, e.g. 500, 552, 596-1000
array_indices_submitted	string	job array	Complete list of indices of subjobs given at submission time. Given as range, e.g. 1-100

9.4 Job Array States

Job array states map closely to job states except for the 'B' state. The 'B' state applies to job arrays and indicates that at least one subjob has left the queued state and is running or has run, but not all subjobs have run. Job arrays will never be in the 'R', 'S' or 'U' states.

Table 17: Job Array States

State	Indication
B	The job array has started
W	The job array has a wait time in the future
H	The job array is held
T	The job array is in transit between servers
Q	The job array is queued, or has been qre-run
E	All subjobs are finished and the server is cleaning up the job array

9.4.1 Subjob States

Subjobs can be in one of four states, listed here.

Table 18: Subjob States

State	Indication
Q	Queued
R	Running
E	Ending
X	Expired; subjob has completed execution
S	Suspended
U	Suspended by keyboard activity

9.5 PBS Environmental Variables

Table 19: PBS Environmental Variables

Environment Variable Name	Used For	Description
$PBS_ARRAY_INDEX	subjobs	Subjob index in job array, e.g. 7
$PBS_ARRAY_ID	subjobs	Identifier for a job array. Sequence number of job array, e.g. 1234
$PBS_JOBID	Jobs, subjobs	Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 1234[7]

9.6 Staging

File staging for job arrays is like that for jobs, with a variable to specify the subjob index. This variable is **^array_index^**. This is the name of the variable that will be used for the actual array index. The stdout and stderr files follow the naming convention for jobs, but include the identifier of the job array, which includes the subscripted index. As with jobs, the stagein and stageout keywords require the -W option to qsub.

9.6.1 Job Array Staging Syntax on UNIX

stagein= local_path@host:remote_path

stageout = local_path @host: remote_path

“Local_path” is the path on the execution host.
Local_path is either relative to the user’s home directory, or an absolute path.
“Host” is the machine where the data normally resides.
“Remote_path” is the path on the machine where the data normally resides.

Examples:

Remote_path: /film
Data files used as input: frame1, frame2, frame3
Host: store
Local_path: /tmp
Executable: a.out

The filename prefix "frame" is optional. Using it will prevent collisions on the execution machine. For this example, a.out produces frame2.out from frame2.

```
#PBS -W stagein=/tmp/in/frame^array_index^@store:/film/frame^array_index^
#PBS -W stageout=/tmp/out/frame^array_index^.out
      @store:/film/frame^array_index^
#PBS -J 1-3
a.out -N frame   /tmp/in /tmp/out
```

The result will be that the user's directory named "film" contains the original files frame1, frame2, frame3, plus the new files frame1.out, frame2.out and frame3.out.

9.6.1.1Scripts

Example 1
In this example, we have a script named ArrayScript which calls scriptlet1 and scriptlet2. All three scripts are located in `/homedir/testdir`.

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-2
echo "Main script: index " $PBS_ARRAY_INDEX
/homedir/testdir/scriptlet$PBS_ARRAY_INDEX
```

In our example, scriptlet1 and scriptlet2 simply echo their names. We run ArrayScript using the qsub command:

```
qsub ArrayScript
```

Example 2
In this example, we have a script called StageScript. It takes two input files, `dataX` and `extraX`, and makes an output file, `newdataX`, as well as echoing which iteration it is on. The `dataX` and `extraX` files will be staged from `inputs` to `work`, then `newdataX` will be staged from `work` to `outputs`.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein=/homedir/work/data^array_index^
      @host1:/homedir/inputs/data^array_index^
#PBS -W stagein=/homedir/work/extra^array_index^
      @host1:/homedir/inputs/extra^array_index^
#PBS -W stageout=/homedir/work/newdata^array_index^
      @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cd /homedir/work
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX
      >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory): `/homedir/work`
Remote path for inputs (original data files `dataX` and `extraX`): `/homedir/inputs`
Remote path for results (output of computation `newdataX`): `/homedir/outputs`
Host (data storage host): host1

`StageScript` resides in `/homedir/testdir`. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in `/homedir`, our home directory, which is why the line
"`cd /homedir/work`" is in the script.

9.6.1.2 Output filenames

The name of the job array will default to the script name if no name is given via qsub -N. For example, if the sequence number were 1234,

#PBS -N fixgamma

would give stdout for index number 7 the name fixgamma.o1234.7 and stderr the name fixgamma.e1234.7.

The name of the job array can also be given through stdin.

9.6.2 Job Array Staging Syntax on Windows

In windows the stagein and stageout string must be contained in double quotes when using ^array_index^.

Example of a stagein:

qsub -W stagein="foo.^array_index^@host-1:C:\WINNT\Temp\foo.^array_index^" -J 1-5 stage_script

Example of a stageout:

qsub -W stageout="C:\WINNT\Temp\foo.^array_index^@host-1:Q:\my_username\foo.^array_index^_out" -J 1-5 stage_script

9.7 PBS Commands

9.7.1 PBS Commands Taking Job Arrays as Arguments

Note: Some shells such as csh and tcsh use the square bracket ("[", "]") as a metacharacter. When using one of these shells, and a PBS command taking subjobs, job arrays or job array ranges as arguments, the subjob, job array or job array range must be enclosed in double quotes.

The following table shows PBS commands that take job arrays, subjobs or ranges as arguments. The cells in the table indicate which objects are acted upon. In the table,

Array[] =	the job array object
Array[Range] =	the set of subjobs of the job array with indices in range given
Array[Index] =	the individual subjob of the job array with the index given
Array[RUNNING] =	the set of subjobs of the job array which are currently running
Array[QUEUED] =	the set of subjobs of the job array which are currently queued
Array[REMAINING] =	the set of subjobs of the job array which are queued or running
Array[DONE]=	the set of subjobs of the job array which have finished running

Table 20: PBS Commands Taking Job Arrays as Arguments

	Argument to Command		
Command	Array[]	Array[Range]	Array[Index]
qstat	Array[]	Array[Range]	Array[Index]
qdel	Array[] & Array[REMAINING]	Array[Range] where Array[REMAINING]	Array[Index]
qalter	Array[]	erroneous	erroneous
qorder	Array[]	erroneous	erroneous
qmove	Array[] & Array[QUEUED]	erroneous	erroneous
qhold	Array[] & Array[QUEUED]	erroneous	erroneous
qrls	Array[] & Array[QUEUED]	erroneous	erroneous
qrerun	Array[RUNNING] & Array[DONE]	Array[Range] where Array[RUNNING]	Array[Index]
qrun	erroneous	Array[Range] where Array[QUEUED]	Array[Index]
tracejob	erroneous	erroneous	Array[Index]
qsig	Array[RUNNING]	Array[Range] where Array[RUNNING]	Array[Index]
qmsg	erroneous	erroneous	erroneous

9.7.2 qstat: Status of a Job Array

The qstat command is used to query the status of a Job Array. The default output is to list the Job Array in a single line, showing the Job Array Identifier. Options can be combined. To show the state of all running subjobs, use -t -r. To show the state only of subjobs, not job arrays, use -t -J.

Table 21: Job Array and Subjob Options to qstat

Option	Result
-t	Shows state of job array object and subjobs. Will also show state of jobs.
-J	Shows state only of job arrays.
-p	Prints the default display, with column for Percentage Completed. For a job array, this is the number of subjobs completed or deleted divided by the total number of subjobs. For a job, it is time used divided by time requested.

Examples:
We run an example job and an example job array, on a machine with 2 processors:
demoscript:

```
#!/bin/sh
#PBS -N JobExample
sleep 60
```

arrayscript:

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-5
sleep 60
```

We run these scripts using qsub.

```
   qsub arrayscript
1235[].host
   qsub demoscript
1236.host
```

Then:

qstat

```
Job id       Name          User         Time Use S Queue
------------ ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
1236.host   JobExample   user1              0 Q workq
```

qstat -J

```
Job id       Name          User         Time Use S Queue
------------ ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
```

qstat -p

```
Job id       Name          User         % done  S Queue
------------ ------------ ----------  ------- - -----
1235[].host ArrayExample user1            0  B workq
1236.host   JobExample   user1           --  Q workq
```

qstat -t

```
Job id        Name          User         Time Use S Queue
------------  ------------ ----------  -------- - -----
1235[].host  ArrayExample user1              0 B workq
1235[1].host ArrayExample user1       00:00:00 R workq
1235[2].host ArrayExample user1       00:00:00 R workq
1235[3].host ArrayExample user1              0 Q workq
1235[4].host ArrayExample user1              0 Q workq
1235[5].host ArrayExample user1              0 Q workq
1236.host    JobExample   user1              0 Q workq
```

qstat -Jt

```
Job id        Name          User   Time Use S Queue
------------ ------------ -----  -------- - -----
1235[1].host ArrayExample user1  00:00:00 R workq
1235[2].host ArrayExample user1  00:00:00 R workq
1235[3].host ArrayExample user1         0 Q workq
1235[4].host ArrayExample user1         0 Q workq
1235[5].host ArrayExample user1         0 Q workq
```

After the first two subjobs finish:

```
        qstat -Jtp
Job id        Name          User   % done S Queue
------------ ------------ -----  ------ - -----
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq

        qstat -pt
Job id        Name          User   % done S Queue
------------ ------------ -----  ------ - -----
1235[].host  ArrayExample user1      40 B workq
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
1236.host    JobExample   user1      -- Q workq
```

Now if we wait until only the last subjob is still running:

```
        qstat -rt
host:
                                                    Req'd  Req'd   Elap
Job ID     Username Queue Jobname    SessID NDS TSK Memory Time  S Time
---------- ------  ----- ---------  ------ --- --- ------ ----- - -----
1235[5].host user1  workq ArrayExamp 3048    --   1    --    --  R 00:00
1236.host    user1  workq JobExample 3042    --   1    --    --  R 00:00


        qstat -Jrt
host:
                                                    Req'd  Req'd   Elap
Job ID      Username Queue Jobname   SessID NDS TSK Memory Time  S Time
---------- -------- ----- --------- ------ --- --- ------ ----- - -----
1235[5].host   user1 workq ArrayExamp  048   --   1    --    --  R 00:01
```

9.7.3 qdel: Deleting a Job Array

The qdel command will take a job array identifier, subjob identifier or job array range. The indicated object(s) are deleted, including any currently running subjobs. Running subjobs are treated like running jobs. Subjobs not running will be deleted and never run.

9.7.4 qalter: Altering a Job Array

The qalter command can only be used on a job array object, not on subjobs or ranges. Job array attributes are the same as for jobs.

9.7.5 qorder: Ordering Job Arrays in the Queue

The qorder command can only be used with job array objects, not on subjobs or ranges. This will change the queue order of the job array in association with other jobs or job arrays in the queue.

9.7.6 qmove: Moving a Job Array

The qmove command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the ‘Q’, ‘H’, or ‘W’ states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a qstat on the server from which the job array was moved will not show the job array. A qstat on the job array object will be redirected to the new server.

Note: The subjob accounting records will be split between the two servers.

9.7.7 qhold: Holding a Job Array

The qhold command can only be used with job array objects, not with subjobs or ranges. A hold can be applied to a job array only from the ‘Q’, ‘B’ or ‘W’ states. This will put the job array in the ‘H’, held, state. If any subjobs are running, they will run to completion. No queued subjobs will be started while in the ‘H’ state.

9.7.8 qrls: Releasing a Job Array

The qrls command can only be used with job array objects, not with subjobs or ranges. If the job array was in the 'Q' or 'B' state, it will be returned to that state. If it was in the 'W' state, it will be returned to that state unless its waiting time was reached, it will go to the 'Q' state.

9.7.9 qrerun: Requeuing a Job Array

The qrerun command will take a job array identifier, subjob identifier or job array range. If a job array identifier is given as an argument, it is returned to its initial state at submission time, or to its altered state if it has been qaltered. All of that job array's subjobs are requeued, which includes those that are currently running, and completed and deleted. If a subjob or range is given, those subjobs are requeued as jobs would be.

9.7.10 qrun: Running a Job Array

The qrun command takes a subjob or a range of subjobs, not a job array object. If a single subjob is given as the argument, it is run as a job would be. If a range of subjobs is given as the argument, the non-running subjobs within that range will be run.

9.7.11 tracejob on Job Arrays

The tracejob command can be run on job arrays and individual subjobs. When tracejob is run on a job array or a subjob, the same information is displayed as for a job, with additional information for a job array. Note that subjobs do not exist until they are running, so tracejob will not show any information until they are. When tracejob is run on a job array, the information displayed is only that for the job array object, not the subjobs. Job arrays themselves do not produce any MOM log information. Running tracejob on a job array will give information about why a subjob did not start.

9.7.12 qsig: Signaling a Job Array

If a job array object, subjob or job array range is given to qsig, all currently running subjobs within the specified set will be sent the signal.

9.7.13 qmsg: Sending Messages

The qmsg command is not supported by job arrays.

9.8 Other PBS Commands Supported for Job Arrays

9.8.1 qselect: Selection of Job Arrays

The default behavior of qselect is to return the job array identifier, without returning subjob identifiers.

Note: qselect will not return any job arrays when the state selection (-s) option restricts the set to ‘R’, ‘S’, ‘T’ or ‘U’, because a job array will never be in any of these states. However, qselect can be used to return a list of subjobs by using the -t option.

Options to qselect can be combined. For example, to restrict the selection to subjobs, use both the -J and the -T options. To select only running subjobs, use -J -T -sR.

Table 22: Options to qselect for Job Arrays

Option	Selects	Result
(none)	jobs, job arrays	Shows job and job array identifiers
-J	job arrays	Shows only job array identifiers
-T	jobs, subjobs	Shows job and subjob identifiers

9.9 Job Arrays and xpbs

xpbs does not support job arrays.

9.10 More on Job Arrays

9.10.1 Job Array Run Limits

Jobs and subjobs are treated the same way by job run limits. For example, if *max_user_run* is set to 5, a user can have a maximum of 5 subjobs and/or jobs running.

9.10.2 Starving

A job array's starving status is based on the queued portion of the array. This means that if there is a queued subjob which is starving, the job array is starving. A running subjob retains its sarving status when it was started.

9.10.3 Job Array Dependencies

Job dependencies are supported:

- between job arrays and job arrays
- between job arrays and jobs
- between jobs and job arrays

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

9.10.4 Accounting

Job accounting records for job arrays and subjobs are the same as for jobs. When a job array has been moved from one server to another, the subjob accounting records are split between the two servers, except that there will be no 'Q' records for subjobs.

9.10.5 Checkpointing

Checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed, instead they run to completion.

9.10.6 Prologues and Epilogues

If defined, prologues and epilogues will run at the beginning and end of each subjob, but not for job arrays.

9.10.7 Job Array Exit Status

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob.

Exit Status	Meaning
0	All subjobs of the job array returned an exit status of 0. No PBS error occurred. Deleted subjobs are not considered
1	At least 1 subjob returned a non-zero exit status. No PBS error occurred.
2	A PBS error occurred.

9.10.8 Scheduling Job Arrays

All subjobs within a job array have the same scheduling priority.

9.10.8.1 Preemption

Individual subjobs may be preempted by higher priority work.

9.10.8.2 Peer Scheduling

Peer scheduling does not support job arrays.

9.10.8.3 Fairshare

Subjobs are treated like jobs with respect to fairshare ordering, fairshare accounting and fairshare limits. If running enough subjobs of a job array causes the priority of the owning entity to change, additional subjobs from that job array may not be the next to start.

9.10.8.4 Node Grouping

All nodes associated with a single subjob should belong to the same node group. Different subjobs can be put on different node groups.

Chapter 10
Running Multi-node Jobs

If PBS has been set up to manage a cluster of computers or on a parallel system, it is likely with the intent of managing multi-node parallel jobs. PBS can allocate individual nodes to multiple jobs at the same time (called *time-sharing*) or to a single job at a time (providing *exclusive access*; also called *space-sharing*). When a job requests exclusive access, the entire node is allocated to the job regardless of the number of processors or the amount of memory in the node. Users should explicitly request such exclusive access if desired. To have PBS allocate nodes to a user's job, the user must specify how many nodes and of what type are required for the job. The user's parallel job must then execute tasks on the allocated nodes. The chapter explains how to submit such multi-node parallel jobs to PBS.

Important: Recall that the previously discussed "`resc_spec`" requests are for single-node jobs, and are *not* supported for multi-node jobs. (See also section 4.10 "Single-Node Conditional Requests" on page 51.)

10.1 Basic `node_spec` Usage

A simple `node_spec` requests a specific number of nodes. This is the most common case for single- and dual-processor Linux clusters, where all the nodes are identical. It doesn't matter to your job which nodes are allocated to it. Thus requesting simply the number of nodes needed is sufficient. The first example below requests any 8 nodes for the job; the second example request exclusive access to 4 nodes (via the `#excl` property).

```
qsub -l nodes=8 my_job
```

```
#PBS -l nodes=4#excl
...
```

10.1.1Job-specific Nodes File

PBS provides the names of the all the nodes allocated to a particular job in a job-specific file. The file is owned by root but is world readable. The full path and name of the file is passed to the job in the environment variable `PBS_NODEFILE`. The order that the nodes are listed in the `PBS_NODEFILE` depends on the `node_spec` request. This is further illustrated in the Examples section below.

10.1.2Common node_spec Usage

Most users only need to use a few simple, common cases of node specification. These are (1) requesting a specific number of nodes, and (2) requesting a nodes with a particular property, the following examples show. (For details see the discussion of properties in the next section.)

```
#PBS -l nodes=N
```

```
#PBS -l nodes=N#property
...
```

10.2 Detailed Node Specification Syntax

The nodes `resources_list` item is set by the user (via the `qsub` command) to declare the node requirements for the job. It is a string of the form

```
-l nodes=node_spec[ +node_spec...][ #suffix]
```

where *node_spec* can be any of the following:

`N`	Number of nodes needed; if number is used, it must be listed first
`nodename`	Host name of the specific node requested
`property[ :property...]`	One or more site-specific node properties (see also “Node Property” on page 11).

`ppn=X` The number of processes (tasks) per node (defaults to 1)

`ncpus=Y` The number of CPUs (threads) per process (defaults to 1)

`N:spec[ :spec]`
Number of nodes, followed by any of the above requests, which may be further followed by additional of the above requests.

An important distinction in PBS is that you can only ask for resources at the job level (see "PBS System Resources" on page 34), and you must ask for properties at the node level (see "Node Property" on page 11). If a multi-node job is submitted that requests resources, it will be accepted, but the resource requests will be ignored.

The 'node specification' value is one or more `node_spec` joined with the '+' character. Each `node_spec` represents one or more nodes at run time. If no number is specified, one (1) is assumed. The total number of (virtual) processors allocated per node is the product of the number of processes per node (`ppn`) times the number of CPUs per process (`ncpus`).

The `node_spec` may be followed by one or more *global modifiers* (see the `#suffix` in the syntax line above). These are special flags which apply to every `node_spec` in the entire specification. Any site-defined property may be used as global modifiers, as well as the two pre-defined modifiers: "`#shared`" (requesting shared access to a node) and "`#excl`" (requesting exclusive access to the entire node, regardless of the number of CPUs requested).

The "`#shared`" modifier is used when you are willing to share your nodes with other jobs. This typically results in less wait in the queue, but the individual runtime of the job may be longer.

The "`#excl`" modifier is used to request exclusive access the nodes, regardless of the number of CPUs on the nodes. No other job will be allocated those nodes while your jobs is running on them. This typically results in longer queue wait times, but faster runtime for the individual application. Exclusive access will not be granted to time-shared nodes.

Important: The keywords `shared` and `excl` only apply when used as global modifiers.

10.3 Examples

This section contains several examples for using the various options of the node specification syntax.

10.3.1 Requesting Multiple Processes Per Node

A user may request to run multiple processes per node by adding the terms `ppn=#` (for processes per node) to each node expression. As a simple case, consider:

```
#PBS -l nodes=1:ppn=3
...
```

which requests to run three processes on a single node. One node would be allocated to the job, as would three virtual processors on that node. Furthermore, the allocated node will be listed in the `PBS_NODEFILE` three times (one per line).

The following example request two virtual processors on each of three nodes, and the `PBS_NODEFILE` would contain the names of the three allocated nodes, each listed twice, as such: A, B, C, A, B, C.

```
#PBS -l nodes=3:ppn=2
...
```

A more elaborate (and possibly contrived) example would be the following, which requests two virtual processors on one node plus an additional four virtual processors on a second node. (The `PBS_NODEFILE` would contain the two node names multiple times, in the following order: A, B, A, B, B, B).

```
#PBS -l nodes=1:ppn=2+1:ppn=4
...
```

This allows a user to request varying numbers of processes on nodes and by setting the number of *`NPROC`* on the `mpirun` command to the total number of allocated nodes, run one process on each node. (This is useful if one set of files have to be created on each node by a setup process regardless of the number of processes that will run on the nodes during the computation phase.)

10.3.2 Processes (Tasks) vs CPUs

It is further possible to specify the number of CPU (threads) per process, using the `ncpus=#` modifier. For example, the node specification:

```
#PBS -l nodes=4:ppn=3:ncpus=2
...
```

requests a total of four separate nodes; three parallel processes (tasks) should be run on each node (This means each node will appear in the `PBS_NODEFILE` three times.); and two CPUs should be allocated to each process so that each process can run two threads (which results in the `OMP_NUM_THREADS` and `NCPUS` environment variables to be set to two). The above specification yields (4 nodes * 3 processes * 2 CPUs), for a total 24 CPUs.

10.3.3Order of Nodes in `PBS_NODEFILE`

If the job only requests one process per node the order of the nodes will match the order requested. However, if multiple processes are placed per node, the file will contain each separate node first, listed in order to match the request, followed by the required number of repeating occurrences of each node. For example, if a user requests the following nodes:

```
#PBS -l nodes=1:ppn=3+1:ppn=2+1:ppn=1
...
```

then the `PBS_NODEFILE` will contain: A, B, C, A, B, A. This allows the user to have a parallel job step that runs only one process on each node, (e.g. by setting `-nproc=3`). This is useful if the job requires files to setup, one per node, on each node before the main computation is preformed.

If a user specifies `-l nodes=1:ppn=3:ncpus=2` then node A will be listed 3 times and the environment variables *`OMP_NUM_THREADS`* and *`NCPUS`* will both be set to 2.

If the user specifies `-l nodes=1:ncpus=2` then node A will be listed in the `PBS_NODEFILE` once, and the environment variables *`OMP_NUM_THREADS`* and *`NCPUS`* will both be set to 2. Which method is used depends on the individual application, and how the user wants the processes to be distributed across the nodes allocated to the job.

10.3.4Interaction of `nodes` vs `ncpus`

There is typically no need to use both the `ncpus` and `nodes` resource in a single `qsub` command. This is required in some situations, however. This section explains how a combined `ncpus` and `nodes` request is satisfied.

If a job is submitted to PBS with a `nodes` resource specification (`-l nodes=`*X*) and without an `ncpus` specification (no `-l ncpus=`*y*), then PBS will set the `ncpus` resource to match the number of CPUs required by the `nodes` specification.

The following relates what happens if a job is submitted with a combination of `-l ncpus=`*value* and `-l nodes=`*value*

The variable *X* is used to indicate any integer larger than 1; *Y* and *Z* can be any legal integer. The variable *N* is any integer equal to or greater than 1.

Given "`-l ncpus=`*N* `-l nodes=`*value*", if "*value*" is:

1	Then ok and then nodes value is equivalent to `1:ncpus=`*N*
Xi[+Xj..]	Then let X = sum(Xi+Xj+...) The value of `ncpus` must be a multiple of the number of nodes: if N%X == 0, then `ncpus=N` and the `nodes` value becomes X:ncpus=(N/X) That is N/X cpus per node If N%X != 0, then this is an error

If "*value*" includes "`ppn`" and does not include "`ncpus`", then the results are the same as in the above case; `#ncpus=N/`(number of tasks)

For any other case, X:ppn=Y:ncpus=Z, a default N value is replaced by sum of X*Y*Z across the node spec. E.g. `-l nodes=2:blue:ppn=2:ncpus=3+3` is 15 cpus, so `ncpus` is reset to 15. A non-default `ncpus` value is an error when `ncpus=` is specified in the node specification. Here are a few more examples:

`-lncpus=6 -lnodes=3:blue+3:red` is correct
`-lncpus=12 -lnodes=3:blue+3:red` is also correct

but

`-lncpus=10 -lnodes=3:blue+3:red` is not correct because 10 CPUs cannot be spread evenly across 6 nodes.

A job that does not have a node specification (resource requirement) but does specify a number of CPUs via the `-l ncpus=#` syntax is allocated processors as if the job did have a node specification of the form:

```
-l nodes=1:ncpus=#
```

10.4 Summary of Node Specification Options

Table 23: Node Specification Options

Desired Node/ CPU/ Process Layout	Node Specification Option to *qsub*	Resulting *PBS_NODEFILE*	Resulting Value of *NCPUS* Variable
1 CPU anywhere	`-l ncpus=1`	S	1
1 CPU on 1 node	`-l nodes=1`	A	1
1 CPUon each of 4 nodes	`-l nodes=4`	ABCD	1
less common, advanced examples:			
3 CPUs on 1 node	`-l nodes=1:ppn=3`	AAA	1
3 CPUs on each of 4 nodes with 2 CPUs allocated per process	`-l nodes=4:ppn=3:ncpus=2`	ABCDAB-CDABCD	2
2 nodes with 2 virtual processors per node	`-l nodes=1:ppn=2+1:ppn=2`	ABAB	2, 2
1 node with 2 virtual processors plus 1 node with 3 virtual processors	`-l nodes=1:ppn=2+1:ppn=3`	ABABB	2, 3

10.4.1 Time-shared vs Cluster Nodes

The difference between time-shared and cluster nodes is:

1. Time-shared nodes may not be requested exclusively with the `#excl` suffix.
2. More processes than CPUs can be run on time-shared nodes but not on cluster nodes.
3. Allocation of cluster nodes is based on the number of processors.

10.5 MPI Jobs with PBS

For most implementations of the Message Passing Interface (MPI), you would use the `mpirun` command to launch your application. For example, here is a sample PBS script for an MPI job:

```
#PBS -l nodes=32
#
mpirun -np 32 -machinefile $PBS_NODEFILE a.out
```

10.5.1 MPICH Jobs With PBS

For users of PBS with MPICH on Linux, the `mpirun` command has been changed slightly. The syntax and arguments are the same except for two options, which should not be set by the user:

-np option:	The number of processes is set by PBS. If the user tries to set this, PBS prints a warning that it is overwriting the value.
-machinefile option	PBS supplies the machinefile. If the user tries to specify it, PBS will print a warning that it is replacing the machinefile.

```
#PBS -l nodes=32
#
mpirun a.out
```

Notes:

1 When submitting a job via qsub, use ppn instead of ncpus in the node specification. For example, to specify 8 nodes of any type with at least two processors available on each,

```
qsub -lnodes=8:ppn=2
```

2 Under Windows the `-localroot` option to `MPICH`'s `mpirun` command may be needed in order to allow the job's processes to run more efficiently.

10.5.2 MPI Jobs Using AIX, POE

For PBS users of AIX machines running IBM's Parallel Operating Environment, or POE, the `poe` command has been changed slightly. The syntax and arguments are the same except for

-procs option:	If this is left blank, the number of nodes is set by PBS If the user sets this value to a number higher than the number of nodes, PBS prints a warning that it is assigning multiple processes per node.
-hostfile option	PBS supplies the hostfile to POE. This should not be supplied by the user, and if the user specifies it, PBS will print a warning that it is replacing the hostfile.

Notes:

1 In order to run more than one process per node, use ppn instead of ncpus in the node specification. For example, to specify 10 nodes of any type with at least two processors available on each,

```
qsub -lnodes=10:ppn=2
```

2 Debugging using pdbx will work in the expected fashion.

3 Your system administrator may have set PBS up so that applications which are directly invoked will be part of a PBS job.

4 Since PBS is tracking tasks started by poe, these tasks are counted towards a user's run limits.

5 Usernames must be identical across nodes.

For more information on using IBM's Parallel Operating Environment, see

"IBM Parallel Environment for AIX 5L Hitchhiker's Guide"

10.6 PVM Jobs with PBS

On a typical system, to execute a Parallel Virtual Machine (PVM) program you would use the `pvmexec` command. For example, here is a sample PBS script for a PVM job:

```
#PBS -l nodes=32
#
pvmexec a.out -inputfile data_in
```

10.7 OpenMP Jobs with PBS

To provide support for OpenMP jobs, the environment variable `OMP_NUM_THREADS` is created for the job with the value of the number of CPUs allocated to the job. The variable `NCPUS` is also set to this value.

10.8 Cpusets with PBS

PBS Professional software supports SGI cpusets on both Altix and IRIX systems. Using cpusets improves performance by exclusively allocating chunks of the system to individual jobs. This may affect where and when your jobs run.

PBS does not support cpusets with multinode jobs.

Appendix A: PBS Environment Variables

Table 24: PBS Environment Variables

Variable	Meaning
NCPUS	Number of threads (or cpus per process (ncpus)) on the node
OM_NUM_THREADS	Same as NCPUS.
PBS_ARRAY_ID	Identifier for job arrays. Consists of sequence number.
PBS_ARRAY_INDEX	Index number of subjob in job array.
PBS_ENVIRONMENT	Indicates job type: **PBS_BATCH** or **PBS_INTERACTIVE**
PBS_JOBCOOKIE	Unique identifier for inter-MOM job-based communication.
PBS_JOBID	The job identifier assigned to the job or job array by the batch system.
PBS_JOBNAME	The job name supplied by the user.
PBS_MOMPORT	Port number on which this job's MOMs will communicate.
PBS_NODEFILE	The filename containing a list of nodes assigned to the job.
PBS_NODENUM	Logical node number of this node allocated to the job.
PBS_O_HOME	Value of **HOME** from submission environment.
PBS_O_HOST	The host name on which the qsub command was executed.
PBS_O_LANG	Value of **LANG** from submission environment
PBS_O_LOGNAME	Value of **LOGNAME** from submission environment
PBS_O_MAIL	Value of **MAIL** from submission environment
PBS_O_PATH	Value of **PATH** from submission environment
PBS_O_QUEUE	The original queue name to which the job was submitted.

Table 24: PBS Environment Variables

Variable	Meaning
PBS_O_SHELL	Value of **SHELL** from submission environment
PBS_O_SYSTEM	The operating system name where qsub was executed.
PBS_O_TZ	Value of **TZ** from submission environment
PBS_O_WORKDIR	The absolute path of directory where qsub was executed.
PBS_QUEUE	The name of the queue from which the job is executed.
PBS_TASKNUM	The task (process) number for the job on this node.
TMPDIR	The job-specific temporary directory for this job.

Appendix B: Converting From NQS to PBS

For those converting to PBS from NQS or NQE, PBS includes a utility called **nqs2pbs** which converts an existing NQS job script so that it will work with PBS. (In fact, the resulting script will be valid to both NQS and PBS.) The existing script is copied and PBS directives ("#PBS") are inserted prior to each NQS directive (either "#QSUB" or "#Q$") in the original script.

```
nqs2pbs existing-NQS-script new-PBS-script
```

Important: Converting NQS date specifications to the PBS form may result in a warning message and an incomplete converted date. PBS does not support date specifications of "today", "tomorrow", or the name of the days of the week such as "Monday". If any of these are encountered in a script, the PBS specification will contain only the time portion of the NQS specification (i.e. `#PBS -a hhmm[.ss]`). It is suggested that you specify the execution time on the `qsub` command line rather than in the script. All times are taken as local time. If any unrecognizable NQS directives are encountered, an error message is displayed. The new PBS script will be deleted if any errors occur.

Section "Setting Up Your UNIX/Linux Environment" on page 18 discusses PBS environment variables.

A queue complex in NQS was a grouping of queues within a batch Server. The purpose of a complex was to provide additional control over resource usage. The advanced scheduling features of PBS eliminates the requirement for queue complexes.

Appendix B: Converting From NQS to PBS

Index

O

P

Q

R

S

T

U