PBS Pro™ 5.1

User Guide

PBS Pro

Release 5.1

User Guide

Portable Batch System User Guide

PBS-3BU01: Release: PBS Pro™ 5.1, Updated: September 5, 2001

Edited by: James Patton Jones

Contributing authors include: Albeaus Bayucan, Robert L. Henderson, James Patton Jones, Casimir Lesiak, Bhroam Mann, Bill Nitzberg, Tom Proett.

For more information, redistribution, licensing, or additional copies of this publication, contact:

Veridian Systems
PBS Products Dept.
2672 Bayshore Parkway, Suite 810
Mountain View, CA 94043

Phone: +1 (650) 967-4675
FAX: +1 (650) 967-3080
URL: www.pbspro.com
Email: sales@pbspro.com

Table of Contents

List of Tables

List of Tables

Preface

Intended Audience

PBS Pro is the professional workload management system from Veridian 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-3BA01 **PBS Administrator Guide**: provides the system administrator with information required to install, configure, and manage PBS, as well as a through discussion of how the various components of PBS interoperate.

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

Ordering Software and Publications

To order additional copies of this and other PBS publications, or to purchase additional software licenses, contact the PBS Products Department of Veridian. Full 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 Pro is an 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*, Major Shared Research Center; 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 Pro) is intended as your knowledgeable companion to the PBS Pro software. The information herein pertains to PBS in general, with specific information for PBS Pro 5.1.

1.1 Book organization

This book is organized into 9 chapters, plus an appendix. 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 Products Department of Veridian.

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 the user to PBS, describing the user interfaces and the user's UNIX 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	explains how PBS interacts with parallel applications, and illustrates how to run such applications under PBS.
Appendix A	provides a quick reference summary of PBS environment variables.
Index	includes references of key words, terms, and concepts.

1.2 What is PBS Pro?

PBS Pro 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 resource 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 Pro addresses these problems for computing-intensive industries such as science, engineering, finance, and entertainment.

Now you can use the power of PBS Pro to take better control of 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 Pro can also help you effectively manage growth by tracking real usage levels across your systems and enhancing effective 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 processors, 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) from 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 lead 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.

The latest chapter in the PBS story began when Veridian (the R&D contractor that developed PBS for NASA) released the Portable Batch System Professional Edition (PBS Pro), a complete workload management solution.

1.4 Why Use PBS Pro?

PBS Pro 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. 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 the resources they have access to.

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 complete customization.

Broad Platform Availability is achieved through support of Windows 2000 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, and interfacing with the global resource manager on the Cray T3e).

1.5 About Veridian

The PBS Pro product is brought to you by the same team that originally developed PBS for NASA over eight years ago. In addition to the core engineering team, the Veridian PBS Products department includes individuals who have supported PBS on computers all around the world, including the largest supercomputers in existence. The staff includes internationally-recognized experts in resource- 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 environment), co-architects of the Department of Defense MetaQueueing Project, co-architects of the NASA Information Power Grid, and co-chair of the Global Grid Forum's Scheduling Group. Veridian staff are routinely invited as speakers on a variety of information technology topics.

Veridian is an advanced information technology company delivering trusted solutions in the areas of national defense, critical infrastructure and essential business systems. A private company with annual revenues of $650 million, Veridian operates at more than 50 locations in the US and overseas, and employs nearly 5,000 computer scientists and software development engineers, systems analysts, information security and forensics specialists and other information technology professionals. The company is known for building strong, long-term relationships with a highly sophisticated customer base.

Chapter 2
Concepts and Terms

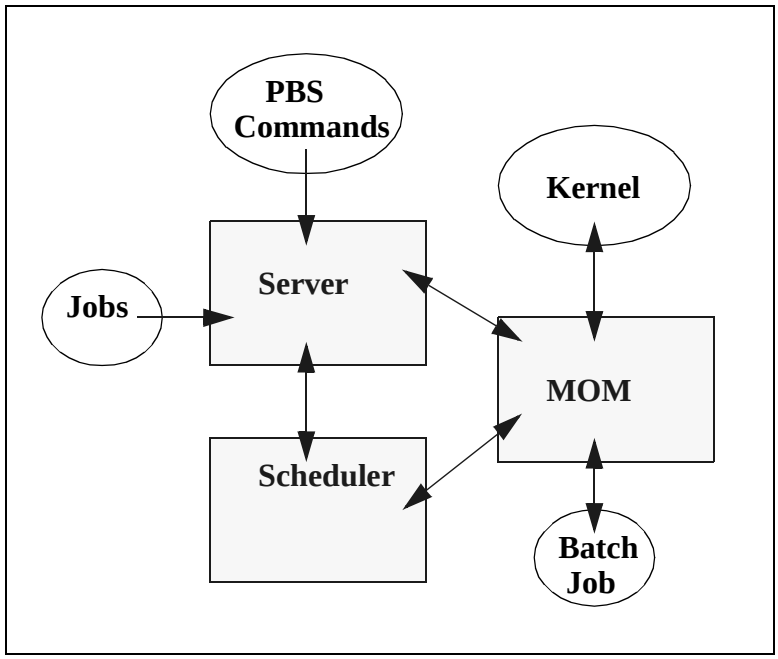
PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload management solutions like PBS include the features of traditional batch queueing but offer greater flexibility and control than first generation batch systems (such as the original UNIX batch system 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 covers both user-level and system-level monitoring as well as monitoring of the scheduling algorithms to see how well they are meeting the stated goals

2.1 PBS Components

PBS consist of two major component types: user-level commands and system daemons. 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 UNIX 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 require specific access privileges as discussed in chapter 11 of the **PBS Administrator Guide**.

Job Server The *Job Server* daemon 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 dae-

mons 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, protecting the job against system crashes, and running the job. Typically there is one Server managing a given set of resources.

Job Executor (MOM)

The *Job Executor* is the daemon which actually places the job into execution. This daemon, *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, 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 daemon 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 chapter 11 of the **PBS Administrator Guide**.

Job Scheduler

The *Job Scheduler* daemon, *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 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 processing units running under a single OS, is one node. Systems like the IBM SP and Linux clusters, which contain many computational units each with their own OS, are collections of many nodes. Nodes can be defined as either *cluster nodes* or *timeshared nodes*, as discussed below.

Nodes & Virtual Processors

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 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.

Cluster Node

A node whose purpose is geared toward running 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.

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.

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 old 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 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 choices for property names should be relayed to the users.

Portable Batch System

PBS consists of one Job Server (pbs_server), one or more Job Scheduler (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.

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 as a grouping for charging for the use of resources.

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 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 uniquely define a single queue at a single Server or it may map into many locations.

Destination Identifier This is a string that names the destination. It is composed 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.

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. Within UNIX and POSIX systems, membership in a group establishes one level of privilege. Group membership 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 the operator or administrator, and a third applied by the system itself or the PBS administrator.

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 in a POSIX session. (A session is a process group the member processes cannot leave.) A non-singleton job consists of multiple tasks of which each is a POSIX session. One *task* will run the job shell script.

Manager The *manager* is the person 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 owner is the user who submitted the job to PBS.

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.

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

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).

Virtual Processor (VP) See Cluster Node.

Chapter 3

Getting Started With PBS

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

3.1 New Features in PBS Pro 5.1

For users already familiar with PBS, the following is a list of new features and changes in PBS Pro release 5.1 which affect users. More detail is given in the indicated sections.

User Changes to node and resource specification syntax. (See "Node Specification Syntax" on page 39.)

User Enhancements to Advance Reservation feature. (See "Advance Reservation of Resources" on page 98.)

User Support for OpenMP jobs. (See "OpenMP Jobs with PBS" on page 109.)

Important: The full list of new features in this release of PBS Pro is given in the **PBS 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 by allowing you to specify the tasks you need run. 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 yours and run them when the resources are available.

With PBS you create a *batch job* which you then submit to PBS. A batch job is simply a shell script containing the set of commands you want run on the computers. It also contains directives which specify the resource requirements (such as 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 job:

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

./subrun
```

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

PBS also provides a special kind of batch job called *interactive-batch.* An interactive-batch job is treated just like a regular batch job (it is queued up, and must wait for resources to become available before it can run). But once it is started, the user's terminal input and output are connected to the job in what appears to be an `rlogin` session. It appears that the user is logged into one of the available computer systems, 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.

3.3 The Two Faces of PBS: CLI vs. GUI

PBS provides two user interfaces: a command line interface (CLI) and a graphical user interface (GUI). You can use either to interact with PBS: both interfaces have the same functionality. Some people prefer the command line, others prefer the GUI.

The command Line Interface (CLI) lets you type commands at the UNIX prompt. The Graphical User Interface (GUI) is a graphical point-and-click interface. The subsequent chapters will explain how to use both the CLI and the GUI to create, submit, and manipulate PBS jobs.

3.4 User's PBS Environment

In order to have your UNIX 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, the following are needed:

User must have access to the requested resources/hosts
User must have a valid group/account
User must have a non-zero allocation (most systems)
User must be able to transfer files between hosts (e.g. via `rcp` or `scp`)

3.4.1 Setting Up Your Own 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 activity should be skipped by placing a test of the environment variable **`PBS_ENVIRONMENT`** (or for NQS compatibility, **`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 PBS jobs run, the "exit status" of the last command executed in the job is reported by `csh` to PBS as the "exit status" of the job. (We will see later that this is important for job dependencies and job chaining.) However, if you have commands in your `.logout`, `csh` will report the exit status of the last of those commands instead. 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
```

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.

3.5 Environment Variables

While we're on the topic of the user's environment, we should mention that 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_”) indicated that the variable is from the job’s originating environment (i.e. the user’s). The Appendix 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/james
PBS_O_LOGNAME=james
PBS_O_PATH=/usr/new/bin:/usr/local/bin:/bin
PBS_O_SHELL=/sbin/csh
PBS_O_TZ=PST8PDT
PBS_O_HOST=cray1.pbspro.com
PBS_O_WORKDIR=/u/james
PBS_O_QUEUE=submit
PBS_JOBNAME=INTERACTIVE
PBS_JOBID=16386.cray1.pbspro.com
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. The environment variable **PBS_DPREFIX** determines the prefix string which identifies directives in the job script. The default prefix string is “#PBS”.

Chapter 3
Getting Started With PBS

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 A Sample PBS Job

As we saw in the previous chapter, a PBS job is a shell script containing the resource requirements of the job and the set of commands you wish to execute. Let's look at an example PBS job in detail:

```
1   #!/bin/sh
2   #PBS -l walltime=1:00:00
3   #PBS -l mem=400mb
4   #PBS -l ncpus=4
5   #PBS -j oe
6
7   ./subrun
```

The first line is standard for any shell script: it 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.

Lines 2-5 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 6-7 as being user commands.

We will see shortly that we use the `qsub` command to submit PBS jobs. Any option that you specific to the `qsub` command can also be provided as a PBS directive inside the PBS script. PBS directives come in two types: resource requirements and job behavior options.

In our example above, lines 2-4 specify the "`-l`" resource list option, followed by a specific resource request. Specifically, lines 2-4 request 1 hour of wall-clock time, 400 megabytes (MB) of memory, and 4 CPUs.

Line 5 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 line 7 is the command line for executing the program we wish to run: our example submarine simulation application, `subrun`. While only a single command is shown in this example (e.g. "`./subrun`"), you can specify as many programs, tasks, or job steps as you need.

4.2 Creating a PBS Job

There are several ways to create a PBS job. The most common are by using your favorite text editor, and by using the PBS graphical user interface (GUI). The rest of this chapter discusses creating and submitting jobs using the command line interface. The next chapter explains in detail how to use the xpbs GUI to create and submit your job.

4.3 Submitting a PBS Job

Let's assume the above example script is in a file called "mysubrun". We submit this script using the **`qsub`** command:

```
% qsub mysubrun
16387.cluster.pbspro.com
```

Notice that upon successful submission of a job, PBS returns a *job identifier* (e.g. "`16387.cluster.pbspro.com`" in the example above.) This identifier is a "handle" to the job. It's format will always 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.

In the previous example we simply submitted the job script to PBS, which in turn read the resource directive contained in the script. However, you can override resource attributes contained in job script by specifying them on the command line. In fact, any job submission option or directive that you can specify inside the job script, you can also specify on the `qsub` command line. This is particularly useful if you just want to submit a single instance of your job, but you don't want to edit the script. For example:

```
% qsub -l ncpus=16 -l walltime=4:00:00 mysubrun
16388.cluster.pbspro.com
```

In this example, the 16 CPUs and 4 hours of wallclock time will override the values specified in the job script.

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, thusly:

```
% qsub -l ncpus=16,walltime=4:00:00 mysubrun
16389.cluster.pbspro.com
```

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

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

./subrun
```

4.4 How PBS Parses a Job Script

The `qsub` command scans the lines of the script file for directives. An initial line in the script that begins with the characters "`#!`" or the character "`:`" will be ignored and scanning will start with the next line. 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 processed as if it had occurred on the command line.

4.5 Converting a NQS/NQE Script 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.

4.6 User Authorization

When the user submits a job from a system other than the one on which the PBS Server is running, the name under which the job is to be executed is selected according to the rules listed under the "`-u`" option to `qsub` (see "Specifying job userID" on page 36). The user submitting the job must be authorized to run the job under the execution user name. This authorization is provided if

(1) The host on which qsub is run is trusted by the execution host (see `/etc/hosts.equiv`),

(2) The execution user has an `.rhosts` file naming the submitting user on the submitting host.

4.7 PBS System Resources

You can request a variety of resources that can be allocated and used by your job, including CPUs, 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 and establishes a limit to the amount of resource that can be consumed. If not set for a generally available resource, such as CPU time, 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 units:

node_spec — specifies the number and type of nodes, processors per node, tasks per node, etc. See "Node Specification Syntax" on page 39 for a complete explanation of use.

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 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. Following this is a table of additional PBS resources that may be requested on computer systems running the Cray UNICOS operating system.

.

Table 1: PBS Resources Available on All Systems

Resource	Meaning	Units
arch	System architecture needed by job.	string
cput	Total amount of CPU time required by all processes in job.	time
file	Maximum disk space requirements for a single file to be created by job.	size
mem	Total amount of RAM memory required by job.	size
ncpus	Number of CPUs (processors) required by job.	unitary
nice	Requested “nice” (UNIX priority) value for job.	unitary
nodes	Number and/or type of nodes needed by job. (See also “Node Specification Syntax” on page 39.)	node_spec
pcput	Maximum amount of CPU time used by any single process in the job.	time
pmem	Maximum amount of physical memory (workingset) used by any single process of the job.	size
pvmem	Maximum amount of virtual memory used by any single process in the job.	size
software	Allows a user to specify software required by the job. The allowable values and effect on job placement is site dependent.	string
vmem	Maximum amount of virtual memory used by all concurrent processes in the job.	size
walltime	Maximum amount of real time during which the job can be in the running state.	time

On Cray systems running UNICOS 8 or later, there are additional resources that may be requested by PBS jobs, as shown below.

Table 2: PBS Resources on Cray UNICOS

Resource	Meaning	Units
mppe	The number of processing elements used by a single process in the job.	unitary
mppt	Maximum amount of wall clock time used on the MPP in the job.	time
mta, mtb...mth	Maximum number of magnetic tape drives required in the corresponding device class of a or b.	unitary
pf	Maximum number of file system blocks that can be used by all process in the job.	
pmppt	Maximum amount of wall clock time used on the MPP by a single process in the job.	time
pncpus	Maximum number of processors used by any single process in the job.	unitary
ppf	Maximum number of file system blocks that can be used by a single process in the job.	size
procs	Maximum number of processes in the job.	unitary
psds	Maximum number of data blocks on the SDS (secondary data storage) for any process in the job.	
sds	Maximum number of data blocks on the SDS (secondary data storage) for the job.	
srfs_big	Session Reservable File System (SRFS) space in BIG-DIR. Note, SRFS is not supported by Cray.	size
srfs_fast	SRFS space in FASTDIR	size
srfs_tmp	SRFS space in TMPDIR.	size
srfs_wrk	SRFS space in WRKDIR.	size

4.8 Job Submission Options

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

Table 3: Options to the `qsub` Command

Option	Function and Page Reference
`-A account_string`	“Specifying a local account” on page 36
`-a date_time`	“Deferring execution” on page 34
`-c interval`	“Specifying job checkpoint interval” on page 35
`-e path`	“Redirecting output and error files” on page 30
`-h`	“Holding a job (delaying execution)” on page 34
`-I`	“Interactive-batch jobs” on page 38
`-j join`	“Merging output and error files” on page 37
`-k keep`	“Retaining output and error files on execution host” on page 37
`-l nodespec`	“Node Specification Syntax” on page 39
`-M user_list`	“Setting e-mail recipient list” on page 32
`-m MailOptions`	“Specifying e-mail notification” on page 32
`-N name`	“Specifying a job name” on page 32
`-o path`	“Redirecting output and error files” on page 30
`-p priority`	“Setting a job’s priority” on page 34
`-q destination`	“Specifying Queue and/or Server” on page 30
`-r value`	“Marking a job as “rerunnable” or not” on page 33
`-S path_list`	“Specifying which shell to use” on page 33
`-u user_list`	“Specifying job userID” on page 36
`-V`	“Exporting environment variables” on page 31

Table 3: Options to the `qsub` Command

Option	Function and Page Reference
`-v variable_list`	"Expanding environment variables" on page 31
`-W depend=list`	"Specifying Job Dependencies" on page 87
`-W group_list=list`	"Specifying job groupID" on page 36
`-W stagein=list`	"Input/Output File Staging" on page 91
`-W stageout=list`	"Input/Output File Staging" on page 91
`-z`	"Suppressing job identifier" on page 38

4.8.1 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 server. See the discussion of **`PBS_DEFAULT`** in "Environment Variables" on page 18.

```
% qsub -q queueName@serverName mysubrun

% qsub -q queueName@serverName.domain.com mysubrun
```

```
% qsub -q queue mysubrun

% qsub -q @server mysubrun
```

```
#!/bin/sh
#PBS -q queueName
...
```

4.8.2 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. The following examples illustrate these various options.

```
#!/bin/sh
#PBS -o /u/james/myOutputFile
#PBS -e /u/james/myErrorFile
...
```

```
% qsub -o myOutputFile mysubrun
% qsub -o /u/james/myOutputFile mysubrun
% qsub -o myWorkstation:/u/james/myOutputFile mysubrun

% qsub -e myErrorFile mysubrun
% qsub -e /u/james/myErrorFile mysubrun
% qsub -e myWorkstation:/u/james/myErrorFile mysubrun
```

4.8.3 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 mysubrun
```

```
#!/bin/sh
#PBS -V
...
```

4.8.4 Expanding environment variables

The "`-v variable_list`" option to `qsub` expands the list of environment variables that are 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 mysubrun
```

4.8.5 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`":

`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 mysubrun
```

```
#!/bin/sh
#PBS -M ae
...
```

4.8.6 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 james@pbspro.com mysubrun
```

4.8.7 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 white space characters with the first character alphabetic. 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 mysubrun
```

```
#!/bin/sh
#PBS -N myName
...
```

4.8.8 Marking a job as "rerunnable" or not

The "`-r value`" option declares whether the job is rerunable. To rerun a job is to terminate the session leader of the job and return the job to the queued state 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 rerunable. If the argument is "`n`", the job is not rerunable. The default value is "`y`", rerunable.

```
% qsub -r n mysubrun
```

```
#!/bin/sh
#PBS -r n
...
```

4.8.9 Specifying which shell to use

The "`-S path_list`" option declares the shell that interprets 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 /bin/tcsh mysubrun

% qsub -S /bin/tcsh@mars,/usr/bin/tcsh@jupiter mysubrun
```

4.8.10 Setting a job's priority

The "`-p priority`" option defines the priority of the job. The *priority* argument must be a integer between -1024 and +1023 inclusive. The default is no priority which is equivalent to a priority of zero.

```
% qsub -p 120 mysubrun
```

```
#!/bin/sh
#PBS -p -300
...
```

4.8.11 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 mysubrun
```

```
#!/bin/sh
#PBS -a 10220700
...
```

4.8.12 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 79.)

```
% qsub -h mysubrun
```

```
#!/bin/sh
#PBS -h
...
```

4.8.13 Specifying job checkpoint interval

The "`-c interval`" option defines the interval at which the job will be checkpointed. 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. Unless otherwise stated, "u" is treated the same as "s".

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

```
% qsub -c s mysubrun
```

```
#!/bin/sh
#PBS -c s
...
```

4.8.14 Specifying job userID

The "`-u user_list`" option defines the user name under which the job is to run on the execution system. If unset, the *user_list* defaults to the user who is running qsub. The *user_list* argument is of the form: `user[@host][,user[@host],...]` Only one user name may be given per specified host, and only one of the user specifications may be supplied without the corresponding host specification. That user name will be used for execution on any host not named in the argument list. A named host refers to the host on which the job is queued for execution, not the actual execution host. Authorization must exist for the job owner to run as the specified user.

```
% qsub -u james@jupiter,barney@purpleplanet mysubrun
```

4.8.15 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 mysubrun
```

4.8.16 Specifying a local account

The "`-A account_string`" option defines the account string associated with the job. The *account_string* is an undefined string of characters and is interpreted by the server which executes the job. This value is often used by sites to track usage by locally defined account names.

```
% qsub -A acct# mysubrun
```

```
#!/bin/sh
#PBS -A accountNumber
...
```

4.8.17 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 mysubrun
```

```
#!/bin/sh
#PBS -j oe
...
```

4.8.18 Retaining output and error files on execution host

The "`-k keep`" option defines which (if either) of standard output or standard error will be retained on the execution host. If set for a stream, this option overrides the path name for that stream. If not set, neither stream 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.

e The standard error stream is to retained on the execution host. The stream 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.esequence` 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 stream is to retained on the execution host. The stream 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.osequence` where `job_name` is the name specified for the job, and sequence is the sequence number component of the job identifier.

eo Both the standard output and standard error streams will be retained.

oe Both the standard output and standard error streams will be retained.

n Neither stream is retained.

```
% qsub -k oe mysubrun
```

```
#!/bin/sh
#PBS -k oe
...
```

4.8.19 Suppressing job identifier

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

```
% qsub -z mysubrun
```

```
#!/bin/sh
#PBS -z
...
```

4.8.20 Interactive-batch jobs

The "`-I`" option declares that the job is to be run "interactively". The job will be queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected through qsub to the terminal session in which qsub is running. If the `-I` option is specified on the command line or in a script directive, or if the "interactive" job attribute declared true via the `-W` option, `-W interactive=true`, either on the command line or in a script directive, the job is an interactive job. The script will be processed for directives, but no executable commands will be included with the job. 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 response "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	Suspend the `qsub` program if running under the C shell. "susp" is the suspend character, usually CNTL-Z.
~asusp	Suspend the input half of `qsub` (terminal to job), but allow output to continue to be displayed. Only works under the C shell. "asusp" is the auxiliary suspend character, usually CNTL-Y.

4.9 Node Specification Syntax

With PBS Pro Release 5.1, there are a number of new features and capabilities associated with requesting nodes and controlling where jobs are run. This section summarizes these changes. Subsequent sections discuss the node specification in detail.

First, in an effort to reduce the differences between timeshared and cluster nodes, a job with a `-l nodes=nodespec` resource requirement may now run on a set of nodes that includes time-shared nodes and a job without a `-l nodes=nodespec` may now run on a cluster node. The differences between time-shared and cluster nodes in release 5.1 is discussed later in this section.

The new syntax for *node_spec* is any combination of the following separated by colons ':'.

number {if it appears, it must be first}
node name
property[:property...]
`ppn`=number
`ncpus`=number
number:any other of the above[:any other]

where *ppn* is the number of processes (tasks) per node (defaults to 1) and *ncpus* is the number of CPUs (threads) per process (also defaults to 1).

The total number of (virtual) processors allocated per node is the product of the number of processes (ppn) per node times the number of CPUs per process (NCPUs).

In addition to the existing global modifier suffix of "#shared", there is now "#excl" which means that the user is requesting exclusive access to the entire node, not just the allocated VPs. This allows a user to have exclusive access even if she wants to run on just one of the CPUs.

4.9.1 Processes (Tasks) vs CPUs

The node resource specification has been improved to allow separate requirements for the number of parallel tasks and the number of required CPUs. In release 5.0, the `:ppn=x` sub-specification was read as "processors per node". In release 5.1, `ppn` is read as "processes (or parallel tasks) per node". For example, the node specification.

```
-l nodes=4:brown:ppn=3:ncpus=2
```

requests: a total of four separate nodes, each having the property of "brown"; three parallel processes (tasks) should be run on each node, this means the node will appear in the `PBS_NODEFILE` (MPI hostfile) three times; two CPUs have been allocated to each process so that each process can run two threads, `OMP_NUM_THREADS` is set to 2. The above specification yields (4 nodes * 3 processes * 2 CPUs), for a total 24 CPUs. If `ppn` or `ncpus` is not specified, its value defaults to one.

4.9.2 Order of Nodes in the Node File

In PBS Pro 5.0, the node file named by the environment variable `PBS_NODEFILE` was only created for a job if the job has more than one node allocated. In 5.1, this file is always created.

In 5.1, the order of the hosts listed in the node file has changed when a node appears more than once. If the job only requests one process per node, now as before, 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:

```
-l nodes=A:ppn=3+B:ppn=2+C:ppn=1
```

then under PBS Pro 5.0 the `PBS_NODEFILE` would have contained: A, A, A, B, B, C. But now in PBS Pro 5.1 it will contain: A, B, C, A, B, A. This change allows the user to have a parallel job step that runs only one process on each node, 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.

4.9.3 Exclusive Access to Whole Node

It is now possible via the global suffix of `#excl` to request exclusive access to the entire node, without asking for all of the CPUs on the node. For example:

```
-lnodes=3:green+2:blue#excl
```

requests a total of five nodes, three "green" and two "blue". Exclusive access will be granted to the nodes, regardless of the number of CPUs on the nodes. No other job will be allocated those nodes. Exclusive access will not be granted to time-shared nodes.

4.9.4 NCPUS Request

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:

```
-lnodes=1:ncpus=#
```

4.9.5 Time-shared vs Cluster Nodes

For those already familiar with PBS Pro, in version 5.1 the difference between time-shared and cluster nodes have been reduced to:

1. Time-shared nodes are first choice for jobs that do not have a node specification.
2. Time-share nodes may not be requested exclusively with the `#excl` suffix.
3. More processes than CPUs can be run on time-shared nodes but not on cluster nodes.
4. If load balancing by "load average" is activated in the Job Scheduler, it applies only to time-shared nodes.
5. Allocation of cluster nodes remains based on the number of (virtual) processors.

Chapter 4
Submitting a PBS Job

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 runs under the X-Windows system and 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 User's xpbs Environment

In order to use xpbs, you need to first set up your environment as described in "User's PBS Environment" on page 17.

Next, make sure your X-Windows session is set 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.pbspro.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
```

Finally, launch `xpbs`:

```
% xpbs &
```

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

5.2 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

5.2.1 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	to update the information on hosts, queues, and jobs.
Auto Update	same as *Manual Update* except updating is done automatically every user-specified number of minutes.
Track Job	for periodically checking for returned output files of jobs.
Preferences	for setting certain 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.

5.2.2 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 mouse button> click, <shift key> plus <left mouse button> click for contiguous selection, or <cntrl key> plus <left mouse button> click for non-contiguous selection.

To the right of the Hosts Panel are a series of buttons that represent actions that can be per-

formed on selected hosts(s). (Use of these button will be explained in detail below.) The button are:

detail	for obtaining detailed 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 PBS servers on selected host(s). (-admin only)

Note that some buttons are only visible if `xpbs` is started with the "`-admin`" option. While any user can specify "`-admin`", the buttons will only function for authorized PBS managers and operators.

5.2.3 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 is meant to be selected (highlighted) via a single <left mouse button> click, <shift key> plus <left mouse button> click for contiguous selection, or <cntrl key> plus <lift mouse button> click for non-contiguous selection.

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	for obtaining detailed information about selected queue(s). This functionality can also be achieved by double clicking on a Queues 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)

5.2.4 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 is meant to be selected (highlighted) via a single <left mouse button> click, <shift key> plus <left mouse button> click for contiguous selection, or <cntrl key> plus <left mouse button> click for non-contiguous selection.

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.

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

detail	for obtaining detailed 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 the selected job(s).
move	for moving selected job(s) into some specified destination queue.
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)

5.2.5 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 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.

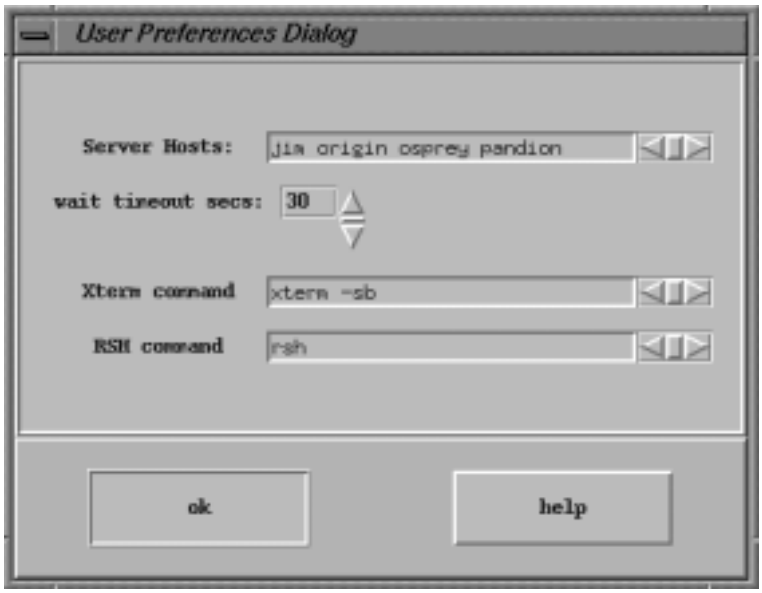
Tip 3. To contiguously select more than one entry: click <left mouse button> then drag the mouse across multiple entries.

Tip 4. To non-contiguously select more than one entry: hold the <cntrl key> while clicking the <left mouse button> 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 which xterm command to use
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 thru the GUI. Each "task" that you perform using `xpbs` is converted into the necessary PBS command and then run on your behalf.

Table 4: `xpbs` Buttons and PBS Commands

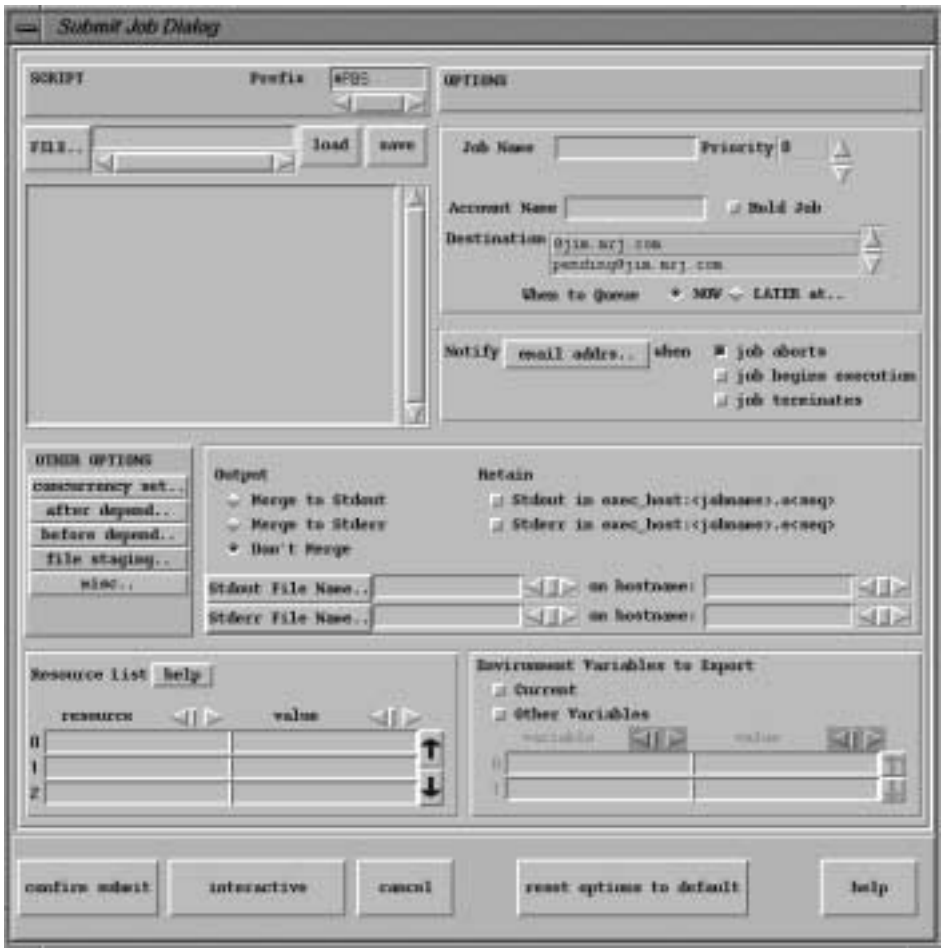
Command Button	PBS Command
detail (Hosts)	qstat -B -f <selected server_host(s)>
terminate	qterm <selected server_host(s)>
detail (Queues)	qstat -Q -f <selected queue(s)>
stop	qstop <selected queue(s)>
start	qstart <selected queue(s)>
enable	qenable <selected queue(s)>
disable	qdisable <selected queue(s)>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

5.6 How to Submit a Job Using xpbs

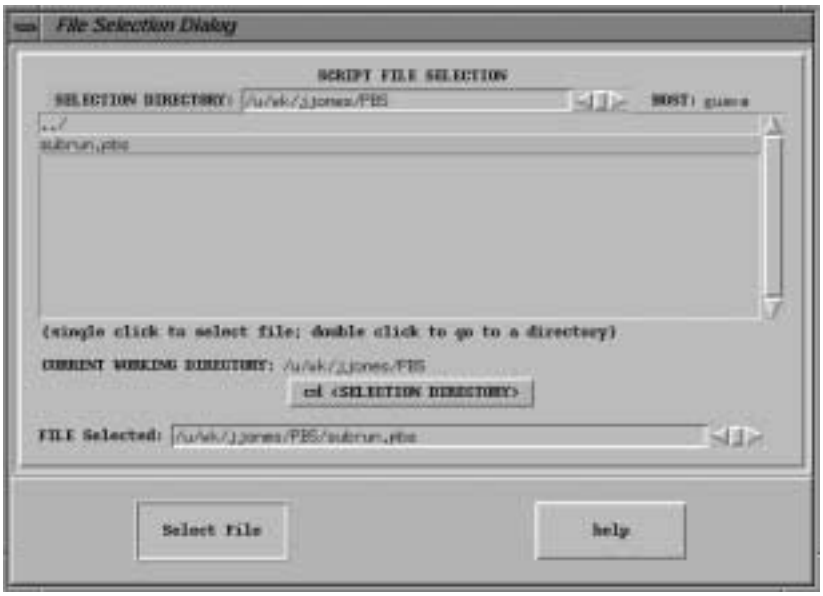
To submit a job using xpbs, perform the following steps:

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 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 a couple of 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 radial 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 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 radial 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:

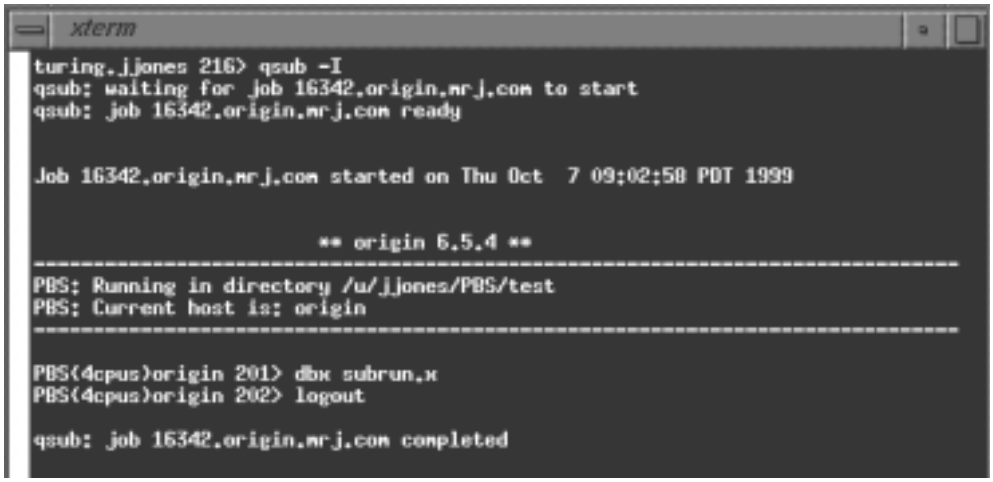
Reset options to default
Save the script to a file
Submit the job as a batch job
Submit the job as an interactive-batch job

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.

Alternatively, you can 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 a xterminal window (`xterm`) to be launched, and within that window a PBS interactive-batch job submitted. (For details and restrictions on use, see “Interactive-batch jobs” on page 38.)



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 `xpbs` configuration file, and will be used the next time you run `xpbs`.

5.8 The xpbs Configuration File

Upon exit, the `xpbs` state may be written to the user's `$HOME/.xpbsrc` file. 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 Widgets Used in xpbs

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:

a. single click with mouse button 1 to select/highlight an entry.
b. shift key + mouse button 1 to contiguously select more than one entry.
c. cntrl key + mouse button 1 to non-contiguously select more than one entry. NOTE: For systems running Tk < 4.0, the newly selected item is reshuffled to appear next to already selected items.
d. click the *Select All/Deselect All* button to select all entries or deselect all entries at once.
e. double clicking an entry usually activates some action that uses the selected entry as a parameter.

A *scrollbar* usually appears either vertically or horizontally and contains 5 distinct areas that are mouse clicked to achieve different effects:

top arrow	Causes the view in the associated widget to shift up by one unit (i.e. the object appears to move down one unit in its window). If the button is held down the action will auto-repeat.
top gap	Causes the view in the associated window to shift up by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very top of the window will now appear at the very bottom). If the button is held down the action will auto-repeat.
slider	Pressing button 1 in this area has no immediate effect except to cause the slider to appear sunken rather than raised. However, if the mouse is moved with the button down then the slider will be dragged, adjusting the view as the mouse is moved.
bottom gap	Causes the view in the associated window to shift down by one less than the number of units in the window (i.e. the portion of the object that used to appear at the very bottom of the window will now appear at the very top). If the button is held down the action will auto-repeat.
bottom arrow	Causes the view in the associated window to shift down by one unit (i.e. the object appears to move up one unit in its window). If the button is held down the action will auto-repeat.

An *entry* widget brought into focus with a click of the left mouse button. 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>, <Cntrl-f>, or <Cntrl-b> (move backwards) 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 click of the left mouse button. 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.

5.10 `xpbs` X-Windows Preferences

The resources that can be set in the X resources 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 *defServerFile.
*defServerFile	the file containing the name of the default server host. The content of this will be substituted for the **`PBS_DEFAULT_SERVER`** keyword in *serverHosts value.
*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 5
Using the xpbs GUI

Chapter 6

Checking Job / System Status

This chapter will introduce 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 request the status of jobs, queues, and the PBS server. The requested status is written to standard out. When requesting job status, any jobs for which the user does not have view privilege are not displayed. 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 18.)

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:

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
T	Job is in transition (being moved to a new location)
W	Job is waiting for its requested execution time to be reached
S	Job is suspended

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

```
% qstat
Job id    Name         User         Time Use S Queue
--------- ------------ ------------ -------- - -----
16.south  aims14       james               0 H workq
18.south  aims14       james               0 W workq
26.south  airfoil      barry        00:21:03 R workq
27.south  airfoil      barry        21:09:12 R workq
28.south  subrun       james               0 Q workq
29.south  tns3d        utley               0 Q workq
30.south  airfoil      barry               0 Q workq
31.south  seq_35_3     bayuca              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
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 Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south james  workq aims14   --   --   1  -- 0:01 H  --
18.south james  workq aims14   --   --   1  -- 0:01 W  --
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
52.south james  workq subrun   --   --   1  -- 0:10 Q  --
53.south utley  workq tns3d    --   --   1  -- 0:20 Q  --
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
55.south bayuca 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 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.pbspro.com
    server_state = Active
    scheduling = True
    total_jobs = 14
    state_count = Transit:0 Queued:13 Held:0 Waiting:0
                  Running:1 Exiting:0
    managers = james@fast.pbspro.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_5_1_1
```

6.1.3 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*.

```
% 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
```

6.1.4 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.

```
% qstat -f 89
Job Id: 89.south
    Job_Name = tns3d
    Job_Owner = utley@south.pbspro.com
    resources_used.cput = 00:00:00
    resources_used.mem = 2700kb
    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 2001
    Error_Path = south:/u/utley/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 2001
    Output_Path = south:/u/utley/tns3d.o89
    Priority = 0
    qtime = Thu Aug 23 10:11:09 2001
    Rerunable = True
    Resource_List.ncpus = 1
    Resource_List.walltime = 00:20:00
    session_id = 2083
    substate = 42
    Variable_List = PBS_O_HOME=/u/utley,PBS_O_LANG=en_US,
       PBS_O_LOGNAME=utley,PBS_O_PATH=/bin:/usr/bin,
       PBS_O_SHELL=/bin/csh,PBS_O_HOST=south,
       PBS_O_WORKDIR=/u/utley,PBS_O_SYSTEM=Linux,
       PBS_O_QUEUE=workq
    euser = utley
    egroup = mrj
    queue_rank = 88
    queue_type = E
    comment = Job run on node south - started at 10:41
    etime = Thu Aug 23 10:11:09 2001
```

6.1.5 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. "`*.pbspro.com`". *`user_name`* without a "`@host`" is equivalent to "`user_name@*`", that is at any host.

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

% qstat -u james,barry

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

6.1.6 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.

6.1.7 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).

6.1.8 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.

6.1.9 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.

6.1.10 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, and includes the node name and number of virtual processors assigned to the job. 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 james  workq aims14   --   --   1  -- 0:01 H  --
    --
18.south james  workq aims14   --   --   1  -- 0:01 W  --
    --
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
    south/0
52.south james  workq subrun   --   --   1  -- 0:10 Q  --
    --
```

6.1.11 Display Disk Reservation Information

The "**-R**" option to `qstat` is only applicable to systems running Session Reservable File System (SRFS) or similar disk reservation software. With this option PBS displays any disk space reservations associated with the jobs.

6.1.12Display Job Comment

The "**-s**" option to `qstat` displays the job comment, 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 james  workq aims14   --   --   1  -- 0:01 H  --
    Job held by james on Wed Aug 22 13:06:11 2001
18.south james  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 james  workq subrun   --   --   1  -- 0:10 Q  --
    Not Running: No available resources on nodes
```

6.1.13Display 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
```

6.2 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 canal 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.

Similarly, to view detailed information about 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 apples for jobs as well. 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 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 jobs 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 the *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 form of the *date_time* operand of the `touch(1)` command:

```
[[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 occur-

rences the single letter `n`, or one or more of the letters `u`, `o`, 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
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 *resource_list* is in the following format:

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

The relation operator op **must** be present.

-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 of 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 *Rerunable* 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`, `T`, and `W`. The characters in the *states* argument have the following interpretation:

Table 5: Job States Viewable by Users

State	Meaning
E	the Exiting state.
H	the Held state.
Q	the Queued state.
R	the Running state.
T	the Transiting state.
W	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. "`*.pbspro.com`". User_name without a "`@host`" is equivalent to "`user_name@*`", that is 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
```

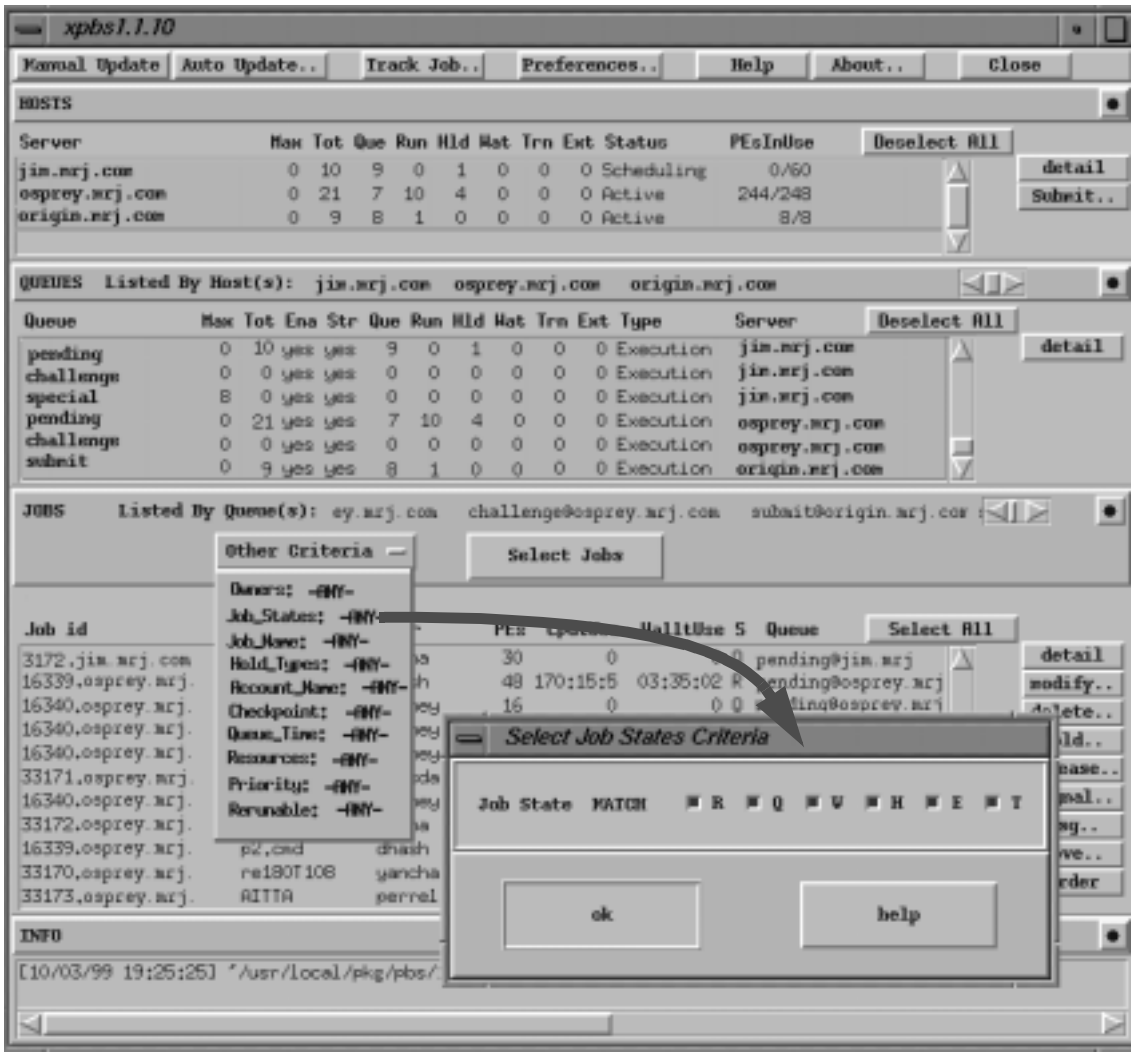
Notice that what is returned is the job identifiers of jobs that match the selection criteria. This may or may not be enough information for your purposes. Many users will use UNIX shell syntax, and pass the list of job identifiers directly into qstat for viewing purposes, as shown in the next example.

```
% 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
```

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.

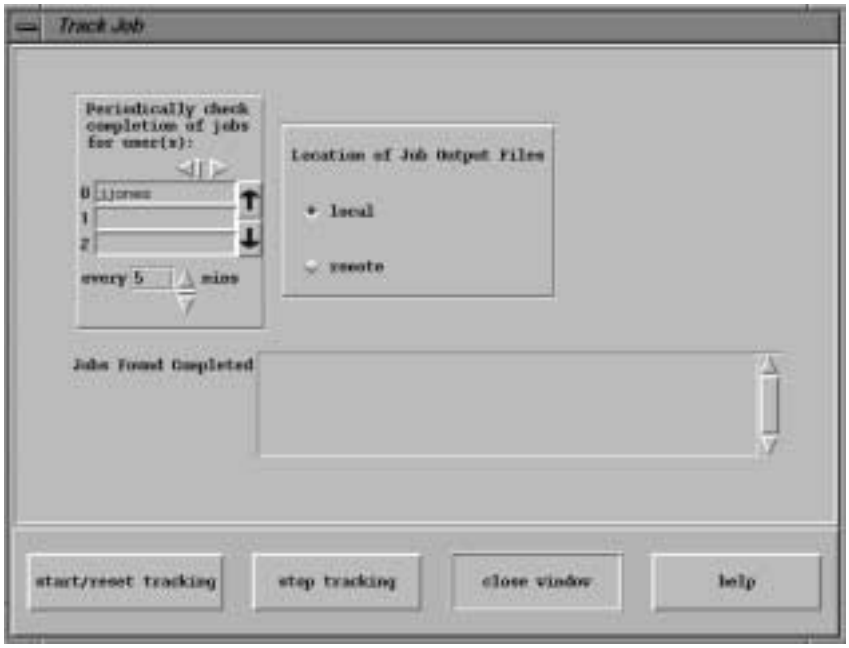


You may specify as many or as few selection criteria as you wish. When you have completed your selection, click on the *Select Jobs* button above the HOSTS panel to have `xpbs` refresh the display with the jobs that match your selection criteria. The selected criteria will remain in effect until you 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.

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 who 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.

6.6 Using the `qstat` TCL Interface

If `qstat` is compiled with an option to include a tcl interpreter, using the `-f` flag to get a full display causes a check to be made for a script file to use to output the requested information. The first location checked is `$HOME/.qstatrc`. If this does not exist, the next location checked is administrator configured. If one of these is found, a Tcl interpreter is started and the script file is passed to it along with three global variables.

The command line arguments are split into two variable named *flags* and *operands*. The status information is passed in a variable named *objects*. All of these variables are Tcl lists. The *flags* list contains the name of the command (usually "`qstat`") as its first element. Any other elements are command line option flags with any options they use, presented in the order given on the command line. They are broken up individually so that if two flags are given together on the command line, they are separated in the list. For example, if the user typed

```
qstat -QfWbigdisplay
```

the flags list would contain

```
qstat -Q -f -W bigdisplay
```

The operands list contains all other command line arguments following the flags. There will always be at least one element in *operands* because if no operands are typed by the user, the default destination or server name is used. The *objects* list contains all the information retrieved from the Server(s) so the Tcl interpreter can run once to format the entire output. This list has the same number of elements as the *operands* list. Each element is another list with two elements. The first element is a string giving the type of objects to be found in the second. The string can take the values "server", "queue", "job" or "error". The second element will be a list in which each element is a single batch status object of the type given by the string discussed above. In the case of "error", the list will be empty. Each object is again a list. The first element is the name of the object. The second is a list of attributes. The third element will be the object text. All three of these object elements correspond with fields in the structure batch_status which is described in detail for each type of object by the man pages for `pbs_statjob(3)`, `pbs_statque(3)`, and `pbs_statserver(3)`. Each attribute in the second element list whose elements correspond with the `attrl` structure. Each will be a list with two elements. The first will be the attribute name and the second will be the attribute value.

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, follwed by the `xpbs` method.

7.1 Modifying Job Attributes

There may come a time when you need to change an attribute on a job you have already 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 while the job is still queued. However, once a job begins execution, the resource limits cannot be changed. These include:

cputime
walltime
number of CPUs
memory

The usage syntax for `qalter` is:

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

The *`job-resources`* are same option and value pairs used on the `qsub` command line. (See “Submitting a PBS Job” on page 22.) 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 “twinkie”):

```
% 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 twinkie 54

% qstat -a 54
                                                  Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq twinkie  --   --   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 widow.

7.2 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, the batch operator, or the batch administrator.

```
% qdel 17
```

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

7.3 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 supported 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 ether 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`, `o`, or `s` in any combination, or the letter `n`. The hold type associated with each letter is:

Letter	Meaning
n	none
u	user
o	operator
s	system

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 (1) cause the job to be checkpointed, (2) the resources assigned to the job will be released, and (3) the job is 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 rerun with the `qrerun` command.

Similarly, the `qrls` command releases the 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 holds as exist for setting a hold.

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 twinkie  --   --   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 twinkie  --   --   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 twinkie  --   --   1  -- 0:20 Q  --
```

To hold (or release) a job using `xpbs`, first select the job(s) of interest, then click the *hold* (or *release*) 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 message can be written using the **`qmsg`** command.

The usage syntax of the `qmsg` command is:

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

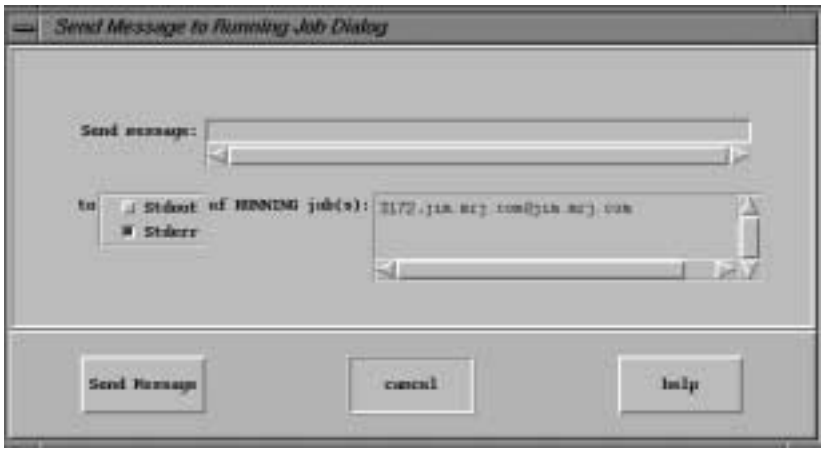
The "`-E`" option writes the message into the error file of the specified job(s). The "`-O`" option write the message into the output file of the specified job(s). If neither option is specified, the message will be written to the standard error file of the job.

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 following operands are *`job_identifiers`* which specify the jobs to receive the message string. For example:

```
% qmsg -E "hello to my error (.e) file" 54
% qmsg -O "hello to my output (.o) file" 54
% qmsg "this too will go to my error (.e) file" 54
```

Important: On most systems the command "`kill -l`" (that's 'minus ell') will list all the available signals. The UNIX manual page for `kill(1)` usually also lists the available signals.

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, as shown below. 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 a 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:

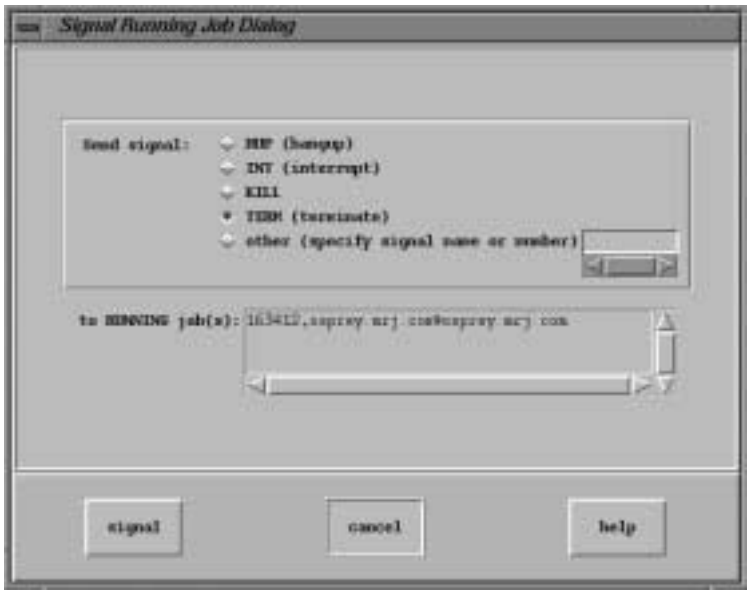
The user is not authorized to signal the job.
The job is not in the running state.
The requested signal is not supported by the system upon which the job is executing.

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
```

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, shown below.



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. No attribute of the job, such as priority is changed. The impact of interchanging the order with the queue(s) is dependent on local job scheduled policy, contact your systems administrator for details.

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

Usage of the `qorder` command is:

```
qorder job_identifier job_identifier
```

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.

7.7 Moving Jobs Between Queues

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

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.

Chapter 8
Advanced PBS Features

This chapter covers the lesser used and more complex topic which will add substantial functionality to your use of PBS. The reader is advised to have already read chapters 5 - 7 of this manual.

8.1 Using Job Comments

Users tend to want to know what is happening to their job. PBS provides a special job attribute, `comment` which is available to the operator, manager, or the Scheduler program. This attribute can be set to a string to pass information to the job owner. It might be used to display information about why the job is not being run or why a hold was placed on the job. Users are able to see this attribute when it is set by using the `-f and -s` option of the `qstat` command. (For details see “Display Job Comment” on page 67.) The Scheduler can set the comment attribute via the pbs_alterjob() API. Operators and managers may use the `-W` option of the `qalter` command, for example

```
qalter -W comment="some text" job_id
```

8.2 Job Exit Status

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 status, the user may either remove `.logout` or edit it as shown in this example:

```
set EXITVAL = $status
[previous contents remain unchanged]
exit $EXITVAL
```

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

8.3 Specifying Job Dependencies

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

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

The *argument* is either a numeric count or a PBS job id 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).

synccount:count
: This job is the first in a set of jobs to be executed at the same time. *`count`* is the number of additional jobs in the set.

syncwith:jobid
: This job is an additional member of a set of jobs to be executed at the same time. In the above and following dependency types, *`jobid`* is the job identifier of the first job in the set.

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

afterok:jobid[:jobid...]
: This job may be scheduled for execution only after jobs *`jobid`* have terminated with no errors. See the `csh` warning under "User's PBS Environment" on page 17.

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

afterany:jobid[:jobid...]
: This job may be scheduled for execution after jobs *jobid* have terminated, 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 forms, see below.

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, 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 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. Mail will be sent to the job submitter stating the error.

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

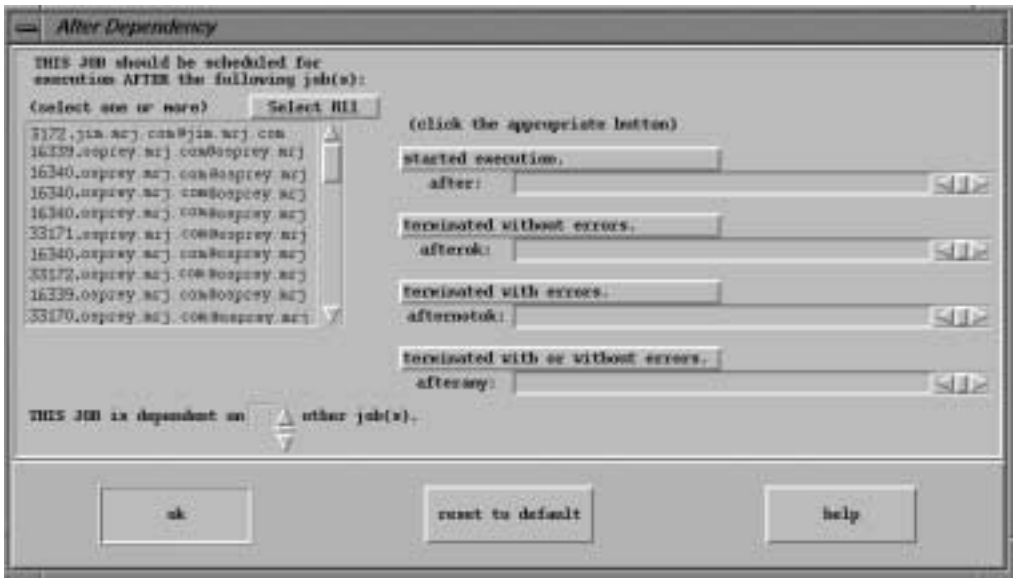
Suppose you have three jobs (job1, job2, and job3) and you want job3 to start *after* job1 and job2 have *ended*. The first example below illustrates the options you would use on the qsub command line to implement these job dependencies.

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

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.pbspro.com
% qsub -W depend=afteroky:16397 job2
16396.jupiter.pbspro.com
```

You can use `xpbs` to specify job dependencies as well. On the *Submit Job* window, in the miscellany 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. The *After Dependency* dialog box is shown below.



8.4 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`(1B). This version of `rcp` is provided because it, unlike some `rcp` implementation, 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.

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

1. If the user's `.cshrc` script outputs any characters to standard output, e.g. contains an echo command, `pbs_rcp` will fail.

2. The user must have permission to `rsh` to the remote host. Output is delivered to the remote destination host with the remote file owner's name being the job owner's name (job submitter). On the execution host, the file is owned by the user's execution name which may be different. For information, see the `-u user_list` option on the `qsub`(1) command.

If the two names are identical, permission to `rcp` may be granted at the system level by an entry in the destination host's `/etc/host.equiv` file naming the execution host. If the owner name and the execution name are different or if the destination host's `/etc/hosts.equiv` file does not contain an entry for the execution host, the user must have a `.rhosts` file in her home directory of the system to which the output files are being returned. The `.rhosts` must contain an entry for the system on which the job executed with the user name under which the job was executed. It is wise to have two lines, one with just the "base" host name and one with the full `host.domain.name`

If PBS is built to use the *Secure Copy Program* `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.

For delivery of output files on the local host, PBS uses the `/bin/cp` command. 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.

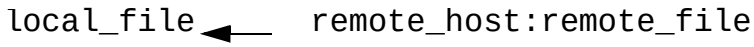
Additional information as to the cause of the delivery problem might be determined from MOM's log file. Each failure is logged.

8.5 Input/Output File Staging

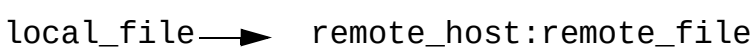
File staging is a way to specify which files should be copied onto the execution host before the job starts, and which should be copied off the execution host when it completes. (For file staging under Globus, see "PBS File Staging through GASS" on page 96.) The "`-W stagein=file_list`" and "`-W stageout=file_list`" 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. 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:



and for stage out, the direction of travel is:



Also 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 `grid.dat` located in the `/u/james` directory 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`.

```
#!/bin/sh
#PBS -W stagein=dat1@server:/u/jones/grid.dat mysubrun
#PBS -W stageout=dat2 mysubrun
...
```

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`* may be a directory name. Also as with `rcp`, the *`local_file`* specified in the stage in/out directive may name a directory. For stage-in, if *`remote_file`* is a directory, then *`local_file`* must also be a directory. For stage out, if *`local_file`* is a directory, then *`remote_file`* must also 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.

Stage-in presents another complication. Assume the user wishes to stage-in the contents of a single file named *stardust* and gives the following stage-in directive:

```
-W stagein=/tmp/foo@mars:stardust
```

If `/tmp/foo` is an existing directory, the local file becomes `/tmp/foo/stardust`. When the job exits, PBS will determined that `/tmp/foo` is a directory and append `/stardust` to it. Thus `/tmp/foo/stardust` will be deleted.

If however, the user wishes to stage-in the contents of a directory named `cat` and gives the following stage-in directive:

```
-W stagein=/tmp/dog/newcat@mars:cat
```

where `/tmp/dog` is an existing directory, then at job end, PBS will determine that

`/tmp/dog/newcat` is a directory and append `/cat` and then fail on the attempt to delete `/tmp/dog/newcat/cat`.

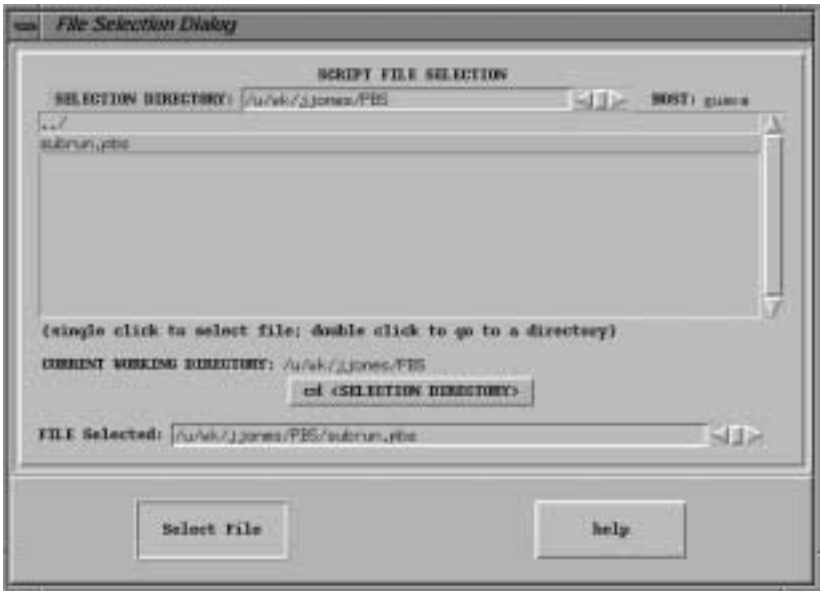
On stage-in when *`remote_file`* is a directory, the user should not specify a new directory as *`local_name`*. In the above case, the user should go with

```
-W stagein=/tmp/dog@mars:cat
```

which will produce `/tmp/dog/cat` which will match what PBS will try to delete at job's end.

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).

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. These will launch the *File Staging* dialog box in which you will be able to set up the file staging you wish, as shown below.



8.6 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, then jobs can be routed to Globus from PBS.

8.6.1 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 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`.

Be sure to have in your PBS Scheduler the ability to recognize a Globus job, and to run it immediately regardless of any scheduling parameters. A Globus job is the one with a resource specification of `site=globus:`.

Also, be sure to create a Globus proxy certificate by running the utility `grid-proxy-init` in order to submit jobs to Globus without a password. If 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.

8.6.2 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 6: `qsub` Options vs. Globus RSL

PBS Option	Globus RSL Mapping
-l site=globus:<globus_gatekeeper>	specifies the gatekeeper to contact
-l ncpus=yyy	count=yyy
-A <account_name>	project=<account_name>
-l {walltime=yyy,cput=yyy, pcput=yyy}	maxtime=yyy where yyy is in minutes
-o <output_path>	stdout=<local_output_path>
-e <error_path>	stderr=<local_error_path> NOTE: PBS will deliver from *local_*path* to user specified output_path and stderr_path
-v <variable_list>	environment=<variable_list>, jobtype=single

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

Table 7: PBS Job States vs. Globus States

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

8.6.3 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. Same with 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 mechanisms are used for transferring files to hosts that run Globus; otherwise, pbs_rcp or cp is used. This means that if the *host* as given in the argument, runs Globus, then Globus communication will be opened to that host.

8.6.4 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.

8.6.5 Examples

Here are some examples of using PBS with Globus:

Example 1: If you want to run a single processor job on globus gatekeeper mars.pbspro.com/jobmanager-fork then you could create a pbs script like the following example:

```
% cat job.script
#PBS -l site=globus:mars.pbspro.com/jobmanager-fork
echo "‘hostname‘:Hello world! Globus style."
```

Upon execution, this will give the sample output:

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

Example 2: If you want to run a multi-processor job on globus gatekeeper pluto.pbspro.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.pbspro.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.pbspro.com:progs/mpitest
/u/jill/mpitest &
/u/jill/mpitest &
/u/jill/mpitest &
/u/jill/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 2000
Process #3 of 4 on host: pluto at time: Mon Aug 29 17:39:01 2000
Process #1 of 4 on host: pluto at time: Mon Aug 29 17:39:01 2000
Process #0 of 4 on host: pluto at time: Mon Aug 29 17:39:01 2000
```

Example 3: Here is a more complicated example. If you want to run a SGI-specified MPI job on a host (e.g. "sgi.glaxey.com") which is running a different batch system via the Globus gatekeeper, with a cpu count of 4, and shipping the architecture compatible executable to the Globus host, and sending the output file back to the submitting host, then do:

```
 % cat job.script
#PBS -l site=globus:sgi.glaxey.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/bayucan/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 regular PBS job, a Globus job can be deleted, signaled, held, released, rerun, have text appended to its output/error files, and be moved from one location to another.

8.7 Advance Reservation of Resources

An *Advance Reservation* is a set of resources with availability limited to a specific user (or group of users), and a specific start time 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, a queue will be created for this reservation. The queue will have an user level access control list set to the user who created it 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 jobs in the queue will be started. Once the reservation is complete, any jobs left in the queue will be deleted.

The Scheduler will check to see if the reservation will conflict with work currently running on the machine (not queued work), other reservations, or dedicated time. If the Scheduler determines that the reservation can not be fulfilled, the reservation will be deleted and mail will be sent to the user.

8.7.1 Submitting a PBS Reservation

The **`pbs_rsub`** command is used to request a reservation of resources. If the request is granted, PBS provisions 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. (For details see "Submitting a PBS Job" on page 22.)

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 then 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. Similarly 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 as a total number of 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_points
: Specifies the set of events that cause the server to send mail messages to the specified list of users. This option takes a string consisting of any combination of "a", "b","c" or "e".

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

Default: "ac"

-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 (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 is a comma separated list of entries of the form: `[+|-]group@host`. 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 the 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. Jobs in the queue that request more of a resource than the queue limit for that resource are not allowed to run. Also, 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 you 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,walltime=30:00 -R 1130
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. The owner of the reservation can submit jobs against the reservation using the `qsub` command, naming the reservation 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 in the current release.

8.7.2 Showing Status of PBS Reservations

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

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

The short option (-S) will show all the reservations in a short concise form. The information provided is the identifier of the reservation, name of the queue belonging to 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  james  CO  Today 11:30 / 1800 /  Today 12:00
R302 R302  barry  CO  Today 15:50 / 1800 /  Today 16:20
R304 R304  james  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 = james@south
reserve_type = 2
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Fri Aug 24 11:30:00 2001
reserve_end = Fri Aug 24 12:00:00 2001
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 = james@south
server = south
ctime = Fri Aug 24 06:30:53 2001
mtime = Fri Aug 24 06:30:53 2001
Variable_List =
PBS_O_LOGNAME=james,PBS_O_HOST=south,PBS_O_MAIL=/var/spool/
mail/james
euser = james
egroup = pbs
```

8.7.3 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, a PBS operator, or PBS Manager.

```
% pbs_rdel R304
```

8.7.4 Reservation Job

A user can, if allowed, submit a job that requests the reservation of resources for a specified time period in the future. If the system can confirm the request, the scheduler will direct the server to run the job in that future window of time.

8.7.5 Identification and Status

When the user requests an advance resources reservation via the pbs_rsub command, an option (“-I n”) is available to wait for 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 use pbs_rstat or some other means to descern 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. After that period of time the server is to delete the request for resource reservation.

8.7.6 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 happens to belong to an advance reservation will have this fact show up in the job record.

8.7.7 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.

8.8 Running Jobs on Scyld Beowulf Clusters

This section contains information specific to running PBS jobs on Scyld Computing Corporation's Beowulf clusters. Users should use the `ncpus` resource when submitting their jobs. PBS will allocates nodes based on the number of CPUs requested by a job. For example:

```
qsub -lncpus=10 script
```

This will create a job which will be allocated enough nodes to have ten CPUs available.

A new environment variable is provided for each job: **`BEOWULF_JOB_MAP`**. Its value is a list of node numbers separated by colon ':' characters. If a node has more than one cpu, its node number will appear as many times as there are CPUs.

Chapter 9
Running Parallel Jobs

9.1 Parallel 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 parallel jobs. As discussed in section 3.5 “Multiple Execution Systems” on page 17, PBS can allocate nodes to one job at a time, called space-sharing. It is important to remember that the entire node is allocated to the job regardless of the number of processors or the amount of memory in the node. To have PBS allocate nodes to a user’s job, the user must specify how many of what type of nodes are required for the job. Then the user’s parallel job must execute tasks on the allocated nodes.

9.1.1 Requesting Nodes

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...]
```

where *`node_spec`* can be any of the following: `number`, `property[:property...]`, or `number:property[:property...]`. The *`node_spec`* may have an optional global modifier appended. This is of the form `#property`.

For example:

```
6+3:fat+2:fat:hippi+disk#prime
```

Where `fat`, `hippi`, and `disk` are examples of property names assigned by the administrator in the `/var/spool/PBS/server_priv/nodesfile`. The above example translates as the user requesting six plain nodes plus three "fat" nodes plus two nodes that are both "fat" and "hippi" plus one "disk" node, a total of 12 nodes. Where `#prime` is appended as a global modifier, the global property, "prime" is appended by the Server to each element of the node specification. It would be equivalent to

```
6:prime+3:fat:prime+2:fat:hippi:prime+disk:prime
```

A major use of the global modifier is to provide the `shared` keyword. This specifies that all the nodes are to be temporarily-shared nodes. The keyword shared is only recognized as such when used as a global modifier.

9.1.2 Parallel Jobs and Nodes

PBS provides the names of the nodes allocated to a particular job in a file in `/usr/spool/PBS/aux/`. The file is owned by root but world readable. The name of the file is passed to the job in the environment variable *PBS_NODEFILE*. For IBM SP systems, it is also in the variable *MP_HOSTFILE*.

A user may request multiple processes per node by adding the term `ppn=#` (for processor per node) to each node expression. For example, to request 2 VPs on each of 3 nodes and 4 VPs on 2 more nodes, the user can request

```
-l nodes=3:ppn=2+2:ppn=4
```

If a user specifies `-lnodes=A:ppn=3` then node A will be listed in the *PBS_NODEFILE* three times. If the user specifies `-lnodes=A: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.

If a user specifies `-lnodes=A:ppn=3:cpp=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 a user specifies `-lnodes=A:ppn=2+B:ppn=2` then both node A and node B will be listed in the *PBS_NODEFILE* twice. However the listing will have the new ordering of A, B, A, B; not A, A, B, B as before. If `-lnodes=A:ppn=2+B:ppn=3` is given, then the

ordering in the *PBS_NODEFILE* is A, B, A, B, B. 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.)

9.2 MPI Jobs with PBS

On a typical system, to execute an Message Passing Interface (MPI) program you would use the `mpirun` command. For example, here is a sample PBS script for a MPI job:

```
#!/bin/sh
#PBS -l nodes=32
#
mpirun -np 32 ./a.out
```

9.3 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.

9.4 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:

```
#!/bin/sh
#PBS -l nodes=32
#
pvmexec ./a.out -inputfile datain
```

9.5 POE Jobs with PBS

On a most IBM SP-series systems to run any parallel job, you need to launch the application to IBM Parallel Operating Environment (POE) via the `pbspoe` command. For example, here is a sample PBS script which executes a job via POE:

```
#!/bin/sh
#PBS -l nodes=32
#
pbspoe ./a.out
```

9.6 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.

Appendix A: PBS Environment Variables

Table 8: PBS Environment Variables

Variable	Meaning
PBS_O_HOME	Value of **HOME** from submission environment.
PBS_O_LANG	Value of **LANG** from submission environment
PBS_O_LOGNAME	Value of **LOGNAME** from submission environment
PBS_O_PATH	Value of **PATH** from submission environment
PBS_O_MAIL	Value of **MAIL** from submission environment
PBS_O_SHELL	Value of **SHELL** from submission environment
PBS_O_TZ	Value of **TZ** from submission environment
PBS_O_HOST	The host name upon which the `qsub` command is running.
PBS_O_QUEUE	The original queue name to which the job was submitted.
PBS_O_SYSTEM	The operating system name given by `uname -s` on the host on which qsub is running.
PBS_O_WORKDIR	The absolute path of the current working directory of the qsub command.
PBS_ENVIRONMENT	Set to indicate the job is a batch job, or to indicate the job is a PBS interactive job, see -I option.
PBS_JOBID	The job identifier assigned to the job by the batch system.
PBS_JOBNAME	The job name supplied by the user.
PBS_NODEFILE	The filename containing a list of nodes assigned to the job.
PBS_QUEUE	The name of the queue from which the job is executed.

Appendix A
PBS Environment Variables

Index