Altair®

# PBS Professional™ 8.0

# User's Guide

UNIX®, LINUX® and Windows®

# PBS Professional™ User's Guide

Altair® PBS Professional™ 8.0, Updated: October 28, 2006

Edited by Anne Urban

For more information, copies of documentation, and sales, contact Altair at:

Web: www.altair.com, www.pbspro.com

Email: sales@pbspro.com.

## Technical Support

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

# Table of Contents

# Preface

## Intended Audience

PBS Professional is the professional workload management system from Altair that provides a unified queuing and job management interface to a set of computing resources. This document provides the user with the information required to use the Pbs Professional, 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 Professional Quick Start Guide**: offers a short overview of the installation and use of PBS Professional.

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

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

## Ordering Software and Publications

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

## Document Conventions

PBS documentation uses the following typographic conventions.

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

# Acknowledgements

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

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

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

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

# Chapter 1 Introduction

This book, the **User’s Guide** to the PBS Professional is intended as your knowledgeable companion to the PBS Professional software. The information herein pertains to PBS in general, with specific information for PBS Professional 8.0.

## 1.1 Book organization

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

| | |
|---|---|
| Chapter 1 | gives an overview of this book, PBS, and the PBS team. |
| Chapter 2 | discusses the various components of PBS and how they interact, followed by definitions of terms used in PBS and in distributed workload management. |
| Chapter 3 | introduces PBS, describing both user interfaces and suggested settings to the user’s environment. |
| Chapter 4 | describes the structure and components of a PBS job, and explains how to create and submit a PBS job. |

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

## 1.2 Supported Platforms

For a list of supported platforms, see the Release Notes.

## 1.3 What is PBS Professional?

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

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

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

## 1.4 History of PBS

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

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

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

The PBS story continued when MRJ-Veridian (the R&D contractor that developed PBS for NASA) released the Portable Batch System Professional Edition (PBS Pro), a commercial, enterprise-ready, workload management solution. Three years later, the MRJ-Veridian PBS Products business unit was acquired by Altair Engineering, Inc. Altair set up the PBS Products unit as a subsidiary company named Altair Grid Technologies focused on PBS Professional and related Grid software.

## 1.5 About the PBS Team

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

## 1.6 About Altair Engineering

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

## 1.7 Why Use PBS?

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

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

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

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

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

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

*System Monitoring* includes a graphical user interface for system monitoring. Displays vnode 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, and execution conditioned on the success or failure of another specific job (or set of jobs).

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

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

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

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

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

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

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

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

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

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

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

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

Chapter 2
# Concepts and Terms

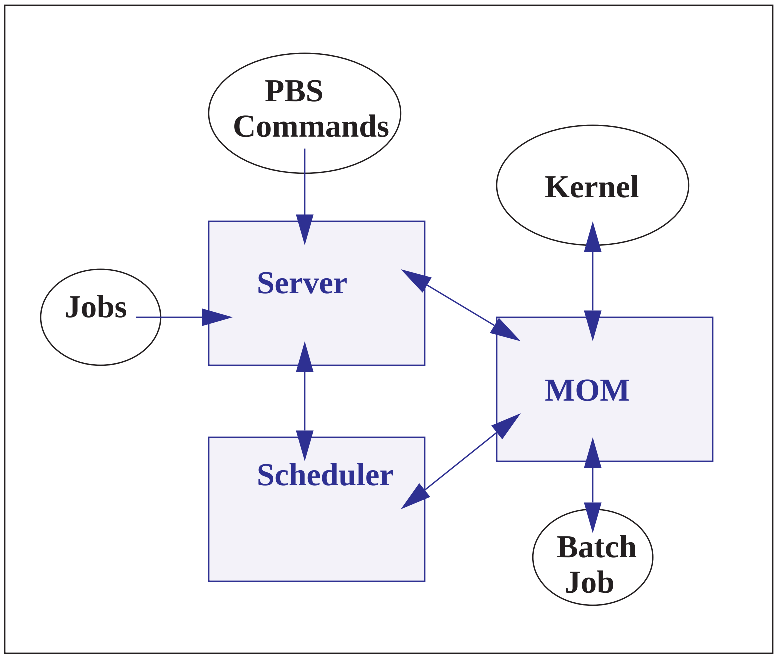
PBS is a distributed workload management system. As such, PBS handles the management and monitoring of the computational workload on a set of one or more computers. Modern workload management solutions like PBS Professional include the features of traditional batch queueing but offer greater flexibility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

Workload management systems have three primary roles:

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

## 2.1 PBS Components

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



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

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

**Server** The *Job Server* daemon/service is the central focus for PBS. Within this document, it is generally referred to as *the Server* or by the execution name *pbs_server.* All commands and the other daemons/services communicate with the Server via an *Internet Protocol* (IP) network. The Server's main function is to provide the basic batch services such as receiving/creating a batch job, modifying the job, and running the job. Normally, there is one Server managing a given set of resources. However if the Server Failover feature is enabled, there will be two Servers.

**Job Executor (MOM)**

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

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

**Scheduler**

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

## 2.2 Defining PBS Concepts and Terms

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

**Node** No longer used. A *node* to PBS is a computer system with a single *operating system* (OS) image, a unified virtual memory space, one or more CPUs and one or more IP addresses. Frequently, the term *execution host* is used for node. A computer such as the SGI Origin 3000, which contains multiple CPUs running under a single OS, is one node. Systems like the IBM SP and Linux clusters, which contain separate computational units each with their own OS, are collections of nodes.

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

**Vnode** A virtual node, or *vnode*, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. Each vnode in a complex must have a unique name. Vnodes can share resources, such as node-locked licenses.

**Chunk** A set of resources allocated as a unit to a job. Specified inside a selection directive. All pars of a chunk come from the same host. In a typical MPI job, there is one chunk per MPI process.

**Cluster** This is any collection of hosts 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.

| | |
|---|---|
| **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. 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*). |
| **Vnode Attribute** | Vnodes have attributes associated with them that provide control information. The attributes defined for vnodes are: state, the list of jobs to which the vnode is allocated, properties, `max_running`, `max_user_run`, `max_group_run`, and both assigned and available resources ("`resources_assigned`" and "`resources_available`"). |
| **Pbs Professional** | PBS consists of one Server (`pbs_server`), one Scheduler (`pbs_sched`), and one or more MOMs (`pbs_mom`). The PBS System can be set up to distribute the workload to one large system, multiple systems, a cluster of hosts, or any combination of these. |
| **Virtual Processor (VP)** | A vnode may be declared to consist of one or more *virtual processors (VPs)*. The term virtual is used because the number of VPs declared does not have to equal the number of real processors (CPUs) on the physical vnode. The default number of virtual processors on a vnode is the number of currently functioning physical processors; the PBS Manager can change the number of VPs as required by local policy. |

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

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

| | |
|---|---|
| **API** | PBS provides an *Application Programming Interface (API)* which is used by the commands to communicate with the Server. This API is described in the **PBS Professional External Reference Specification**. A site may make use of the API to implement new commands if so desired. |
| **Attribute** | An *attribute* is a data item whose value affects the operation or behavior of the object and can be set by the owner of the object. |
| **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 vnodes that are allocated to only one job at a time or of vnodes that have many jobs executing at once on each vnode or a combination of these two scenarios. |
| **Destination** | This is the location within PBS where a job is sent. A destination may be a single queue at a single Server or it may map into multiple possible locations, tried in turn until one accepts the job. |
| **Destination Identifier** | This is a string that names the destination. It is composed of two parts and has the format queue@server where server is the name of a PBS Server and queue is the string identifying a queue on that Server. |
| **Directive** | A means by which the user specifies to PBS the value of a variable such as number of CPUs, the name of a job, etc. The default start of a directive is "#PBS". PBS directives either specify resource requirements or attribute values. See page 57. |
| **File Staging** | *File staging* is the movement of files between a specified location and the execution host. See "Stage In" and "Stage Out" below. |
| **Group ID (GID)** | This unique number represents a specific group (see Group). |
| **Group** | *Group* refers to collection of system users (see Users). A user must be a member of a group and may be a member of more than one. Membership in a group establishes one level of privilege, and is also often used to control or limit access to |

|  |  |
|---|---|
|  | system resources. |
| **Hold** | A restriction which prevents a job from being selected for processing. There are three types of holds. One is applied by the job owner, another is applied by a PBS *Operator*, and a third applied by the system itself or the PBS *Manager*. (See also Operator and Manager in this glossary.) |
| **Job or Batch Job** | The basic execution object managed by the batch subsystem. A job is a collection of related processes which is managed as a whole. A job can often be thought of as a shell script running a set of tasks. |
| **Manager** | A *manager* is authorized to use all capabilities of PBS. The Manager may act upon the Server, queues, or jobs. The Manager is also called the administrator. |
| **Operator** | A person authorized to use some but not all of the restricted capabilities of PBS is an *operator*. |
| **Owner** | The user who submitted a specific job to PBS. |
| **PBS_HOME** | Refers to the path under which PBS was installed on the local system. Your local system administrator can provide the specific location. |
| **POSIX** | This acronym refers to the various standards developed by the "Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society" under standard P1003. |
| **Requeue** | The process of stopping a running (executing) job and putting it back into the queued ("Q") state. This includes placing the job as close as possible to its former position in that queue. |
| **Rerunnable** | If a PBS job can be terminated and its execution restarted from the beginning without harmful side effects, the job is rerunnable. |
| **Stage In** | This process refers to moving a file or files to the execution host prior to the PBS job beginning execution. |
| **Stage Out** | This process refers to moving a file or files off of the execution host after the PBS job completes execution. |

| | |
|---|---|
| **User** | Each system *user* is identified by a unique character string (the user name) and by a unique number (the user id). |
| **Task** | *Task* is a POSIX session started by MOM on behalf of a job. |
| **User ID (UID)** | Privilege to access system resources and services is typically established by the *user id*, which is a numeric identifier uniquely assigned to each user (see User). |
| **Job Array** | A collection of jobs submitted under a single job id. These jobs can be modified, queried and displayed as a set. |

# Chapter 3
# Getting Started With PBS

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

## 3.1 New Features in PBS Professional 8.0

PBS Professional has major new features: the enhancement to job resource requests and job placement on vnodes, and the use of vnodes instead of nodes. Several existing features are replaced by this new feature and are deprecated. All vnodes are now treated alike with respect to allocating jobs and specifying resources. There is no distinction between time-shared and cluster nodes; all nodes are treated as time-shared, but are referred to as vnodes.

The following is a list of new features and changes in PBS Professional release 8.0. More detail is given in the indicated sections.

Enhancement to job resource requests and placement. See "Major Changes" on page 16, "Requesting Resources" on page 32, and "Placing Jobs on Nodes" on page 36.

Improved integration Altix running ProPack 4 or 5. See section 10.5 "MPI Jobs on the Altix" on page 192.

Integration with several versions of MPI: MPICH-GM's mpirun with rsh/ssh and with MPD, MPICH-MX's mpirun with rsh/ssh and with MPD, MPICH2's mpirun and Intel MPI's mpirun. See section 10.4 "MPI Jobs with PBS" on page 174.

Support for IBM Blue Gene. See section 10.6 "Jobs on the IBM Blue Gene" on page 193.

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

## 3.2 Major Changes

The way in which jobs request resources and placement on vnodes has changed. All vnodes are now treated alike with respect to allocating jobs and specifying resources. Node properties are replaced. The nodes file is changed. Several commands have different usage. There may be incompatibilities with prior versions when converting from the deprecated form to the current form. The scheduler groups vnodes differently. The major changes and deprecations are listed below.

### 3.2.1 Resource Requests

Jobs now request resources in two new ways. They can use the *select statement* to define *chunks* and specify the quantity of each chunk. A chunk is a set of resources that are to be allocated as a unit. Jobs can also use a job-wide resource request, which uses `resource=value` pairs. The `-l nodes=` form is deprecated, and if it is used, it will be converted into a request for chunks and job-wide resources.

Do not mix new (“-lselect=”) and old (“-lnodes=”) syntax. This will produce an error.

The qsub, qalter and pbs_rsub commands are used to request resources.

Most existing jobs submitted with "-lnodes" will continue to work as expected. Existing jobs will be automatically converted to the new syntax. However, job tasks may execute in a different order, because vnodes may be assigned in a different order.

Previously-successful jobs submitted with old syntax can fail because a limit that was per-chunk is now job-wide. This is an example of a job submitted using -l nodes=X -lmem=M that would fail because the mem limit is now job-wide. If the following conditions are true:

a. PBS Professional 8.0 using standard MPICH
b. The job is submitted with qsub -lnodes=5 -lmem=10GB
c. The master process of this job tries to use more than 2GB

The job will be killed, where in <= 7.0 the master process could use 10GB before being killed. 10GB is now a job-wide limit, divided up into a 2GB limit per chunk.

For more information, see “Requesting Resources” on page 32, and the `qsub(1B)`, `qalter(1B)`, `pbs_rsub(1B)` and `pbs_resources(7B)` manual pages.

### 3.2.2 Vnodes Instead of Nodes

PBS now manages virtual nodes, or vnodes, rather than nodes. Jobs run on vnodes and resources are defined on vnodes. See "Vnodes: Virtual Nodes" on page 31.

### 3.2.3 Placing Jobs on Vnodes

All vnodes are treated alike with respect to allocating jobs to vnodes. Jobs are placed on vnodes using the *place* statement. This allows specification of whether the job should run on a single host, or be scattered across hosts, or be grouped by a resource, or whether it should run anywhere available. It also allows specification of whether the job should have exclusive use of its vnode(s).

For more information, see "Placing Jobs on Nodes" on page 36 and the `pbs_resources(7B)` manual page.

### 3.2.4 Node Types

What were called nodes are now called vnodes. All vnodes are now treated alike, and are treated the same as what were called "time-shared nodes". The types "time-shared" and "cluster" are deprecated. The **`:ts`** suffix is deprecated. It is silently ignored, and not preserved during rewrite. The vnode attribute `ntype` will only be used to distinguish between PBS and Globus vnodes. It is read-only.

### 3.2.5 Boolean Resources Replace Properties

What were properties are now *boolean resources*. These are treated like other resources except that they take on boolean values of "True" or "False". The term "property" is **deprecated**. It will be accepted for the time being in a nodespec.

### 3.2.6 New Resources

PBS has new resources: *mpiprocs*, *ompthreads* and *vnode*. There is also a new resource type, called multivalued string. See "PBS Resources" on page 32.

### 3.2.7 Nodes File

The server's node definition file has changed due to the use of boolean resources and the change in node types. The nodes file now lists vnodes. If an existing nodes file with deprecated forms is used, when it is written out, it will be in the new form. Properties will be written as boolean resources and any `:ts` suffixes will be dropped.

### 3.2.8 PBS_NODEFILE

The file containing the vnodes allocated to a job has changed in content and order from prior versions of PBS Professional. This file now lists vnode names. This file's name is given by the environment variable PBS_NODEFILE.

For jobs which requested a set of nodes via the -lnodes=nodespec option to qsub, each vnode allocated to the job will be listed N times, where N is the total number of CPUs allocated from the vnode divided by the number of threads requested. For example, qsub -lnodes=4:ncpus=3:ppn=2 will result in each of the four vnodes being written twice (6 CPUs divided by 3 from ncpus.) The file will contain the name of the first vnode twice, followed by the second vnode twice, etc.

For jobs which do not request vnodes via the -lnodes=nodespec option, the nodes file will contain the names of the allocated vnodes with each name repeated N times, where N is the total number of CPUs allocated from that vnode. For example, qsub -l select=3:ncpus=2 -lplace=scatter will result in this PBS_NODEFILE:

```
vnodeA
vnodeA
vnodeB
vnodeB
vnodeC
vnodeC
```

### 3.2.9 The qsub, qalter and pbs_rsub Commands

The `qsub, qalter and pbs_rsub` command usage has changed due to the change in resource requests and job placement. See the `qsub(1B), qalter(1B)` and `pbs_rsub(1B)` manual pages.

### 3.2.10 The qstat Command

The new exec_vnode attribute displayed via `qstat` shows the allocated resources on each vnode.

The exec_vnode line looks like:

```
exec_vnode = hostA:ncpus=1
```

For example, a job requesting

```
    -l select=2:ncpus=1:mem=1gb+1:ncpus=4:mem=2gb
```

would get an exec_vnode of

```
    exec_vnode =
            (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)
            +(VNC:ncpus=4:mem=2gb)
```

Note that the vnodes and resources required to satisfy a chunk are grouped by parentheses. In the example above, if two vnodes on a single host were required to satisfy the last chunk, the exec_vnode might be:

```
    exec_vnode =(VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)
            +(VNC1:ncpus=2:mem=1gb+VNC2:ncpus =2:mem=1gb)
```

### 3.2.11 Warning About Conversion from nodespec

When a nodespec is converted into a select statement, the job will have the environment variables NCPUS and OMP_NUM_THREADS set to the value of ncpus in the first piece of the nodespec. This may produce incompatibilities with prior versions when a complex node specification using different values of ncpus and ppn in different pieces is converted.

For detailed information on conversion from -l nodes=nodespec to -l select= and -l place=, see section 4.8 "Backward Compatibility" on page 54.

### 3.2.12 Node Grouping Changes

If a job requests grouping by a resource, i.e. *place=group=resource,* then the chunks are placed as requested and complex-wide node grouping is ignored.

If a job is to use node grouping but the required number of vnodes is not defined in any one group, grouping is ignored. This behavior is unchanged.

Node grouping is now part of a larger system called placement sets. Placement sets allow partitioning by multiple resources, so that a vnode may be in one set that share a value for one resource, and another set that share a different value for a different resource. See the **PBS Professional Administrator's Guide**.

### 3.2.13 Administrator Can Set Chunk Defaults

The administrator can set server and queue defaults for resources used in chunks. See "Server Configuration Attributes" on page 76, "Attributes for execution queues only" on page 92 and the pbs_server_attributes(7B) and pbs_queue_attributes(7B) manual pages.

### 3.2.14 New MPI Integrations

PBS is integrated with several versions of MPI: MPICH-GM's mpirun with rsh/ssh and with MPD, MPICH-MX's mpirun with rsh/ssh and with MPD, MPICH2's mpirun and Intel MPI's mpirun. section 10.4 "MPI Jobs with PBS" on page 174.

### 3.2.15 Deprecations

- The **-l nodes=nodespec** form is replaced by the -l select= and -l place= statements.
- The **nodes** resource is no longer used.
- The **-l resource=rescspec** form is replaced by the -l select= statement.
- The **time-shared** node type is no longer used, and
- the **:ts** suffix is obsolete.
- The **cluster** node type is no longer used.
- The resource **arch** is only used inside of a select statement.
- The resource **host** is only used inside of a select statement.
- The **nodect** resource is obsolete. The ncpus resource should be used instead. Sites which currently have default values or limits based on nodect should change them to be based on ncpus.
- The **neednodes** resource is obsolete.
- The **ssinodes** resource is obsolete.
- **Properties** are replaced by boolean resources.

## 3.3 Using PBS

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

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

UNIX:

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

Windows:

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

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

## 3.4 PBS Interfaces

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

**Table 1: PBS Professional User and Manager Commands**

| User Commands | | Administrator Commands | |
|---|---|---|---|
| **Command** | **Purpose** | **Command** | **Purpose** |
| `nqs2pbs` | Convert from NQS | `pbs-report` | Report job statistics |
| `pbs_rdel` | Delete Adv. Reservation | `pbs_hostid` | Report host identifier |
| `pbs_rstat` | Status Adv. Reservation | `pbs_hostn` | Report host name(s) |
| `pbs_password` | Update per user / per server password[1] | `pbs_migrate_users` | Migrate per user / per server passwords [1] |

**Table 1: PBS Professional User and Manager Commands**

| User Commands | | Administrator Commands | |
|---|---|---|---|
| `pbs_rsub` | Submit Adv.Reservation | `pbs_probe` | PBS diagnostic tool |
| `pbsdsh` | PBS distributed shell | `pbs_rcp` | File transfer tool |
| `qalter` | Alter job | `pbs_tclsh` | TCL with PBS API |
| `qdel` | Delete job | `pbsfs` | Show fairshare usage |
| `qhold` | Hold a job | `pbsnodes` | Vnode manipulation |
| `qmove` | Move job | `printjob` | Report job details |
| `qmsg` | Send message to job | `qdisable` | Disable a queue |
| `qorder` | Reorder jobs | `qenable` | Enable a queue |
| `qrls` | Release hold on job | `qmgr` | Manager interface |
| `qselect` | Select jobs by criteria | `qrerun` | Requeue running job |
| `qsig` | Send signal to job | `qrun` | Manually start a job |
| `qstat` | Status job, queue, Server | `qstart` | Start a queue |
| `qsub` | Submit a job | `qstop` | Stop a queue |
| `tracejob` | Report job history | `qterm` | Shutdown PBS |
| `xpbs` | Graphical User Interface | `xpbsmon` | GUI monitoring tool |

Notes:
1 Available on Windows only.

## 3.5 User's PBS Environment

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

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

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

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

## 3.6 Usernames Under PBS

By default PBS will use your login identifier as the username under which to run your job. This can be changed via the “`-u`” option to `qsub` (see section 4.10.14 “Specifying Job User ID” on page 52). The user submitting the job must be authorized to run the job under the execution user name (whether explicitly specified or not).

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

## 3.7 Setting Up Your UNIX/Linux Environment

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

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

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

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

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

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

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

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

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

An interactive job comes complete with a pseudotty suitable for running those commands that set terminal characteristics. But more importantly, it does not caution the user that starting something in the background that would persist after the user has exited from the interactive environment might cause trouble for some moms. They could believe that once the interactive session terminates, all the user's processes are gone with it. For example, applications like ssh-agent background themselves into a new session and would prevent a CPU set-enabled mom from deleting the CPU set for the job. This in turn might cause subsequent failed attempts to run new jobs, resulting in them being placed in a held state.

### 3.7.1 Setting MANPATH on SGI Systems

The PBS “man pages” (UNIX manual entries) are installed on SGI systems under `/usr/bsd`, or for the Altix, in `/usr/pbs/man`. In order to find the PBS man pages, users will need to ensure that `/usr/bsd` is set within their `MANPATH`. The following example illustrates this for the C shell:

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

## 3.8 Setting Up Your Windows Environment

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

### 3.8.1 Windows User's HOMEDIR

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

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

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

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

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

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

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

```
\Documents and Settings\username
\Documents and Settings\username.local-hostname
\Documents and Settings\username.local-hostname.00N
\Documents and Settings\username.domain-name
```

where *N* is a number (third path above)

### 3.8.2 Windows Usernames and Job Submission

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

```
qsub -u “Bob Jones” my_job
```

### 3.8.3 Windows rhosts File

The Windows `rhosts` file is located in the user's [`PROFILE_PATH`], for example: `\Documents and Settings\`*`username`*`\.rhosts`, with the format:

*hostname username*

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

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

```
wks031 susan
```

Furthermore, in order for Susan’s output files from her job to be returned to her automatically by PBS, she would need to add an entry to her `.rhosts` file on her workstation naming the execution host `Host1`.

```
Host1 susan
```

If instead, Susan has access to several execution hosts, she would need to add all of them to her `.rhosts` file:

```
Host1 susan
Host2 susan
Host3 susan
```

Note that Domain Name Service (DNS) on Windows may return different permutations for a full hostname, thus it is important to list all the names that a host may be known. For instance, if Host4 is known as "Host4", "Host4.<subdomain>", or "Host4.<subdomain>.<domain>" you should list all three in the `.rhosts` file.

```
Host4 susan
Host4.subdomain susan
Host4.subdomain.domain susan
```

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

```
Host5.subdomain.domain “Bob Jones”
```

## 3.8.4 Windows Mapped Drives and PBS

In Windows XP (unlike Windows 2000), when you map a drive, it is mapped "locally" to your session. The mapped drive cannot be seen by other processes outside of your session. A drive mapped on one session cannot be un-mapped in another session even if it's the same user. This has implications for running jobs under PBS. Specifically if you map a drive, `chdir` to it, and submit a job from that location, the vnode that executes the job may not be able to deliver the files back to the same location from which you issued `qsub`. The workaround is to use the “`-o`” or “`-e`” options to `qsub` and specify a local (non-mapped) directory location for the job output and error files. For details see section 4.10.2 “Redirecting output and error files” on page 46.

## 3.9 Environment Variables

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

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

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

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

The environment variable **`PBS_DPREFIX`** determines the prefix string which identifies directives in the job script. The default prefix string is "`#PBS`"; however the Windows user may wish to change this as discussed in section 4.7 "Changing the Job's PBS Directive" on page 39.

## 3.10 Temporary Scratch Space: TMPDIR

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

Under Windows, `TMP` will also be set to the value of `%TMPDIR%`. The temporary directory will be created under either `\winnt\temp` or `\windows\temp`, unless an alternative directory was specified by the administrator in the MOM configuration file.

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

UNIX:

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

Windows:

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

# Chapter 4
# Submitting a PBS Job

**The way in which jobs request resources and specify placement on vnodes has changed.** There are several associated new features and deprecations. The current method of requesting resources is deprecated. Please see "New Rules for Submitting Jobs" on page 36 and "Placing Jobs on Vnodes" on page 48, as well as the `pbs_resources(7B)` manual page.

## 4.1 Vnodes: Virtual Nodes

A virtual node, or vnode, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. PBS views hosts as being composed of one or more vnodes. Jobs run on one or more vnodes. See the pbs_node_attributes(7B) man page.

### 4.1.1 Relationship Between Hosts, Nodes, and Vnodes

A host is any computer. Execution hosts used to be called nodes. However, some machines such as the Altix can be treated as if they are made up of separate pieces containing CPUs, memory, or both. Each piece is called a vnode. Some hosts have a single vnode and some have multiple vnodes. PBS treats all vnodes alike in most respects. Chunks cannot be split across hosts, but they can be split across vnodes on the same host.

Resources that are defined at the host level are applied to vnodes. If you define a dynamic host-level resource, it will be shared among the vnodes on that host. This sharing is managed by the MOM. If you define a static host-level resource, you can set its value at each vnode, or you can set it on one vnode and make it indirect at other vnodes.

## 4.2 PBS Resources

Resources can be available on the server and queues, and on vnodes. Jobs can request resources.  Resources are allocated to jobs, and some resources such as memory are consumed by jobs. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs.

PBS provides built-in resources, and in addition, allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the server or queue level (e.g. floating licenses.) Vnodes can share resources. The administrator can also specify default arguments for qsub. These arguments can include resources. See the qsub(1B) man page.

Resources made available by defining them via resources_available at the server level are only used as job-wide resources. These resources (e.g. walltime, server_dyn_res) are requested using -l RESOURCE=VALUE. Resources made available at the host (vnode) level are only used as chunk resources, and can only be requested within chunks using -l select=RESOURCE=VALUE. Resources such as mem and ncpus can only be used at the vnode level.

Resources are allocated to jobs both by explicitly requesting them and by applying specified defaults. Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See the pbs_resources(7B) manual page.

Jobs are assigned limits on the amount of resources they can use. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources.

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are derived from both the amount of requested job-wide resources and the amount found when per-chunk consumable resources are summed. Job resource limits from sums of all chunks, including defaults, override those from job-wide defaults and resource requests. Limits include both explicitly requested resources and default resources.

Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.

A "consumable" resource is one that is reduced by being used, for example, ncpus, licenses, or mem. A "non-consumable" resource is not reduced through use, for example, walltime or a boolean resource.

Resources are tracked in server, queue, vnode and job attributes. Servers, queues and vnodes have two attributes, resources_available.RESOURCE and resources_assigned.RESOURCE. The resources_available.RESOURCE attribute tracks the total amount of the resource available at that server, queue or vnode, without regard to how much is in use. The resources_assigned.RESOURCE attribute tracks how much of that resource has been assigned to jobs at that server, queue or vnode. Jobs have an attribute called resources_used.RESOURCE which tracks the amount of that resource used by that job.

### 4.2.1 Resource Types

Resources have the following data types:

| | |
|---|---|
| boolean | Boolean-valued resource. Should be defined only at the vnode level, for manageability. Non-consumable. Name of resource is a string. Allowable values (case insensitive): True\|T\|Y\|1\|False\|F\|N\|0<br><br>A boolean resource named "RESOURCE" is defined in PBS_HOME/server_priv/resourcedef by putting in a line of the form:<br>RESOURCE type=boolean flag=h<br><br>or on vnode VNODE from qmgr:<br>Qmgr: set node <VNODE> |

resources_available.<RESOURCE>=<value>

| | |
|---|---|
| float | Float. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...] |
| long | Long integer. Allowable values: 0-9 [[0-9] ...] |
| size | Number of bytes (default) or words. It is expressed in the form `integer[suffix]`. The suffix is a multiplier defined in the following table. The size of a word is the word size on the execution host. |

| | |
|---|---|
| b or w | bytes or words. |
| kb or kw | Kilo ($2^{10}$, 1024) bytes or words. |
| mb or mw | Mega ($2^{20}$, 1,048,576) bytes or words. |
| gb or gw | Giga ($2^{30}$, 1,073,741,824) bytes or words. |
| tb or tw | Tera ($2^{40}$, or 1024 gigabytes) bytes or words. |
| pb or pw | Peta ($2^{50}$, or 1,048,576 gigabytes) bytes or words. |

| | |
|---|---|
| string | String. Non-consumable. Allowable values: [_a-zA-Z0-9][[-_a-zA-Z0-9[]#.] ...] (Leading underscore ("_"), alphabetic or numeric, followed by dash ("-"), underscore ("_"), alphabetic, numeric, left bracket ("["), right bracket ("]"), hash ("#") or period (".")) |
| string array | String-valued resource which can contain multiple values. Comma-separated list of strings. Non-consumable. Resource request will succeed if request matches one of the values. Resource request can contain only one string. |
| 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]
```

### 4.2.2 Built-in Resources

The table below lists the built-in resources that can be requested by PBS jobs on any system.

**Table 2: PBS Resources**

| Resource | Description |
|---|---|
| arch | System architecture. For use inside chunks only. One architecture can be defined for a vnode. One architecture can be requested per vnode. Allowable values and effect on job placement are site-dependent. Type: string. |
| cput | Amount of CPU time used by the job for all processes on all vnodes. Establishes a job resource limit. Non-consumable. Type: time. |
| file | Size of any single file that may be created by the job. Type: size. |
| host | Name of execution host. For use inside chunks only. Automatically set to the short form of the hostname in the Mom attribute. Cannot be changed. Site-dependent. Type: string. |
| mem | Amount of physical memory i.e. workingset allocated to the job, either job-wide or vnode-level. Consumable. Type: size. |
| mpiprocs | Number of MPI processes for this chunk. Defaults to 1 if ncpus > 0, 0 otherwise. For use inside chunks only. Type: integer.<br><br>The number of lines in PBS_NODEFILE is the sum of the values of mpiprocs for all chunks requested by the job. For each chunk with mpiprocs=P, the host name for that chunk is written to the PBS_NODEFILE P times. |
| ncpus | Number of processors requested. Cannot be shared across vnodes. Consumable. Type: integer. |
| nice | Nice value under which the job is to be run. Host-dependent. Type: integer. |

**Table 2: PBS Resources**

| Resource | Description |
|---|---|
| nodect | Number of chunks in resource request from selection directive, or number of hosts requested from node specification. Otherwise defaults to value of 1. Read-only. Type: integer. |
| ompthreads | Number of OpenMP threads for this chunk. Defaults to ncpus if not specified. For use inside chunks only. Type: integer.<br><br>For the MPI process with rank 0, the environment variables NCPUS and OMP_NUM_THREADS are set to the value of ompthreads. For other MPI processes, behavior is dependent on MPI implementation. |
| pcput | Amount of CPU time allocated to any single process in the job. Establishes a job resource limit. Non-consumable. Type: time. |
| pmem | Amount of physical memory (workingset) for use by any single process of the job. Establishes a job resource limit. Consumable. Type: size |
| pvmem | Amount of virtual memory for use by any single process in the job. Establishes a job resource limit. Consumable. Type: size. |
| software | Site-specific software specification. For use only in job-wide resource requests. Allowable values and effect on job placement are site-dependent. Type: string. |
| vmem | Amount of virtual memory for use by all concurrent processes in the job. Establishes a job resource limit, or when used within a chunk, establishes a per-chunk limit. Consumable. Type: size. |
| vnode | Name of virtual node (vnode) on which to execute. For use inside chunks only. Site-dependent. Type: string. See the pbs_node_attributes(7B) man page. |
| walltime | Amount of wall-clock time during which the job can run. Establishes a job resource limit. Non-consumable. Default: 5 years. Type: time. |

## 4.3 PBS Jobs

### 4.3.1 New Rules for Submitting Jobs

The new "place" specification cannot be used without the "select" specification. See

"Placing Jobs on Vnodes" on page 48.

A "select" specification cannot be used with a "nodes" specification.

A "select" specification cannot be used with -lncpus, -lmem, -lvmem, -larch, -lhost.

The built-in resource "software" is not a vnode-level resource. See "PBS Resources" on page 32.

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

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

### 4.3.2 PBS Job Script

A PBS job script consists of:

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

To submit a PBS job, the user can type

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

#### 4.3.2.1 Specifying the shell

| | |
|---|---|
| UNIX Users: | Since the job file under UNIX is a "shell script", the first line of the job file specifies which shell to use to execute the script. The Bourne shell (`sh`) is the default, but you can change this to your favorite shell. This first line can be omitted if it is acceptable for the job file to be interpreted using the Bourne shell. The remainder of the examples in this manual will assume these conditions are true. If this is not true for your site, simply add the shell specifier. |
| Windows Users: | Windows does not use a shell specification. This line will not appear for a Windows job. |

#### 4.3.2.2 PBS Directives

PBS directives are at the top of the script file. They are used to request resources or set attributes. A directive begins with the default string "#PBS". Attributes can also be set using options to the qsub command, which will override directives.

#### 4.3.2.3 The User's Tasks

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

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

```
More?
```

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

### 4.3.3 Setting Job Attributes

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

## 4.4 Submitting a PBS Job

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

### 4.4.1 Submitting a Job Script

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

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

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

```
sequence-number.servername
```

or, for a job array,

```
sequence-number[].servername.domain
```

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

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

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

The largest possible job ID is the 7-digit number 9,999,999. After this has been reached, job IDs start again at zero.

#### 4.4.1.1 Overriding Directives

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

```
qsub -N newjob my_job
```

#### 4.4.1.2 Submitting a Simple Job

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

```
qsub myjobscript <ret>
```

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

#### 4.4.1.3 Jobs Without a Job Script

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

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

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

## 4.5 Requesting Resources

PBS provides built-in resources, and allows the administrator to define custom resources. The administrator can specify which resources are available on a given vnode, as well as at the queue or server level (e.g. floating licenses.) See “PBS Resources” on page 32 for a listing of built-in resources.

Resources defined at the queue or server level apply to an entire job. If they are defined at the vnode level, they apply only to the part of the job running on that vnode.

Jobs request resources, which are allocated to the job, along with any defaults specified by the administrator.

Custom resources are used for application licenses, scratch space, etc., and are defined by the administrator. See “Customizing PBS Resources” on page 287 of the PBS Professional Administrator’s Guide. Custom resources are used the same way built-in resources are used.

Do not use an old-style resource or node specification (“-lnodes=”) with “-lselect” or “-lplace”. This will produce an error.

### 4.5.1 Allocation

Resources are allocated to jobs both because jobs explicitly request them and because specified default resources are applied to jobs. Jobs explicitly request resources either at the vnode level in *chunks* defined in a *selection statement*, or in *job-wide* resource requests. An explicit resource request can appear in the following, in order of precedence:

```
qalter
qsub
```
PBS job script directives

### 4.5.2 Requesting Resources in Chunks

A *chunk* declares the value of each resource in a set of resources which are to be allocated as a unit to a job. It is the smallest set of resources that will be allocated to a job. All of a chunk must be taken from a single host. A chunk request is a vnode-level request. Chunks are described in a selection statement, which specifies how many of each kind of chunk. A selection statement has this form:

```
 -l select=[N:]chunk[+[N:]chunk ...]
```

If N is not specified, it is taken to be 1.

A chunk is one or more resource_name=value statements separated by a colon, e.g.:

```
ncpus=2:mem=10GB:host=Host1
ncpus=1:mem=20GB:arch=linux
```

Example of multiple chunks in a selection statement:

```
-l select=2:ncpus=1:mem=10GB+3:ncpus=2:mem=8GB:arch=solaris
```

Each job submission can have only one `“-l select”` statement.

Host-level resources can only be requested as part of a chunk. Server or queue resources cannot be requested as part of a chunk.

### 4.5.3 Requesting Job-wide Resources

A *job-wide* resource request is for resource(s) at the server or queue level. Job-wide resources are requested outside of a selection statement, in this form:

```
-l keyword=value[,keyword=value ...]
```

where *keyword* identifies either a consumable resource or a time-based resource such as walltime.

Job-wide resources are used for requesting floating licenses or other resources not tied to specific vnodes, such as cput and walltime.

Job-wide resources can only be requested outside of chunks.

### 4.5.4 Boolean Resources

A resource request can specify whether a boolean resource should be true or false. For example, if some vnodes have green=true and some are red=true, a selection statement for two vnodes, each with one CPU, all green and no red, would be:

```
-l select=2:green=true:red=false:ncpus=1
```

The next example Windows script shows a job-wide request for walltime and a chunk request for ncpus and memory.

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

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

Keep in mind the difference between requesting a vnode-level boolean and a job-wide boolean.

**`qsub -l select=1:green=True`**

will request a vnode with green set to True. However,

**`qsub -l green=True`**

will request green set to True on the server and/or queue.

### 4.5.5 Default Resources

Jobs get default resources, both job-wide and per-chunk, with the following order of precedence, from

Default `qsub` arguments
Default queue resources
Default server resources

For each chunk in the job's selection statement, first queue chunk defaults are applied, then server chunk defaults are applied. If the chunk does not contain a resource defined in the defaults, the default is added. For a resource RESOURCE, a chunk default is called "default_chunk.RESOURCE".

For example, if the queue in which the job is enqueued has the following defaults defined:

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the default_chunk elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb.
```

In the above, mem=2gb and ncpus=1 are inherited from default_chunk.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults. If a default resource is defined which is not specified in the resource request, it is added to the resource request.

## 4.5.6 Requesting Application Licenses

Application licenses are set up as resources defined by the administrator. PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

### 4.5.6.1 Floating Licenses

PBS queries the license server to find out how many floating licenses are available at the beginning of each scheduling cycle. If you wish to request a site-wide floating license, it will typically have been set up as a server-level (job-wide) resource. To request an application license called AppF, use:

```
qsub -l AppF=<number of licenses> <other qsub arguments>
```

If only certain hosts can run the application, they will typically have a host-level boolean resource set to True. To request the application license and the vnodes on which to run the application, use:

```
qsub -l AppF=<number of licenses> <other qsub arguments>
     -l select=haveAppF=True
```

PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

#### 4.5.6.2 Node-locked Licenses

Per-host node-locked licenses are typically set up as either a boolean resource on the vnode(s) that are licensed for the application. The resource request should include one license for each host. To request a host with a per-host node-locked license for AppA in one chunk:

```
qsub -l select=1:runsAppA=1 <jobscript>
```

Per-use node-locked licenses are typically set up so that the host(s) that run the application have the number of licenses that can be used at one time. The number of licenses the job requests should be the same as the number of instances of the application that will be run. To request a host with a per-use node-locked license for AppB, where you’ll run one instance of AppB on two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppB=1
```

Per-CPU node-locked licenses are set up so that the host has one license for each licensed CPU. You must request one license for each CPU. To request a host with a node-locked license for AppC, where you’ll run a job using two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppC=2
```

### 4.5.7 Requesting Scratch Space

Scratch space on a machine is set up as a host-level dynamic resource. The resource will have a name such as “dynscratch”. To request 10MB of scratch space in one chunk, a resource request would include:

```
-l select=1:ncpus=N:dynscratch=10MB
```

### 4.5.8 Note About Submitting Jobs

The default for walltime is 5 years. The scheduler uses walltime to predict when resources will become available. Therefore it is useful to request a reasonable walltime for each job.

### 4.5.9 Submitting Jobs with Resource Specification (Old Syntax)

If neither a node specification nor a selection directive is specified, then a selection directive will be created requesting 1 chunk with resources specified by the job, and with those from the queue or server default resource list. These are: ncpus, mem, arch, host, and software, as well as any other default resources specified by the administrator.

For example, a job submitted with

```
qsub -l ncpus=4:mem=123mb:arch=linux
```

will have the following selection directive created:

```
select=1:ncpus=4:mem=123mb:arch=linux
```

Do not mix old style resource or node specification with the new select and place statements. Do not use one in a job script and the other on the command line. This will result in an error.

### 4.5.10 Moving Jobs From One Queue to Another

If the job is moved from the current queue to a new queue, any default resources in the job's resource list that were contributed by the current queue are removed. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Example:

Given the following set of queue and server default values:

Server
```
resources_default.ncpus=1
```

Queue QA
```
resources_default.ncpus=2
```

```
default_chunk.mem=2gb
```

Queue QB

```
default_chunk.mem=1gb
no default for ncpus
```

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

**`qsub -l ncpus=1 -lmem=4gb`**
In QA: `select=1:ncpus=1:mem=4gb` - no defaults need be applied
In QB: `select=1:ncpus=1:mem=4gb` - no defaults need be applied

**`qsub -l ncpus=1`**
In QA: `select=1:ncpus=1:mem=2gb`
In QB: `select=1:ncpus=1:mem=1gb`

**`qsub -lmem=4gb`**
In QA: `select=1:ncpus=2:mem=4gb`
In QB: `select=1:ncpus=1:mem=4gb`

**`qsub -l nodes=4`**
In QA: `select=4:ncpus=1:mem=2gb`
In QB: `select=4:mem=1gb`

**`qsub -l mem=16gb -l nodes=4`**
In QA: `select=4:ncpus=1:mem=4gb`
In QB: `select=4:ncpus=1:mem=4gb`

### 4.5.11 Jobs Submitted with Undefined Resources

Any job submitted with undefined resources, specified either with "-l select" or with "-l nodes", will not be rejected at submission. The job will be aborted upon being enqueued in an execution queue if the resources are still undefined. This preserves backward compatibility.

### 4.5.12 Limits on Resource Usage

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are established by per-chunk resources, both from explicit requests and from defaults.

Job resource limits set a limit for per-job resource usage. Job resource limits are established both by requesting job-wide resources and when per-chunk consumable resources are summed. Job resource limits from sums of all chunks, including defaults, override those from job-wide defaults and resource requests. Limits include both explicitly requested resources and default resources.

If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated. See The **PBS Professional Administrator's Guide**.

If both job resource limits and a selection directive are specified when a job is submitted, the sum of the resources in the directive must not exceed the specified limits.

For example,

```
qsub -l ncpus=4:mem=200mb-lselect=2:ncpus=2:mem=100mb
```

is accepted because neither the sum of the number of CPUs nor the sum of the requested memory exceeds the specified limits.

However,

```
qsub -l ncpus=2 -lselect=1:ncpus=3
```

will be rejected because the requested number of CPUs, 3, is greater than the specified limit of 2.

If a select directive is supplied and the corresponding job limits are not specified, then job limits are created from the directive for each consumable resource.

For example,

```
qsub -lselect=2:ncpus=3:mem=4gb:arch=linux
```

will have the following job limits set:

ncpus=6 and mem=8gb

## 4.6 Placing Jobs on Vnodes

The *place statement* controls how the job is placed on the vnodes from which resources may be allocated for the job. The place statement can be specified, in order of precedence, via:

Explicit placement request in qalter
Explicit placement request in qsub
Explicit placement request in PBS job script directives
Default `qsub` place statement
Queue default placement rules
Server default placement rules
Built-in default conversion and placement rules

The place statement may be not be used without the select statement.

The place statement has this form:

-l place=[ *arrangement* ][: *sharing* ][: *grouping*]

where

*arrangement* is one of `free | pack | scatter`
*sharing* is one of `excl | shared`
*grouping* can have only one instance of `group=resource`

and where

**Table 3: Placement Modifiers**

| Modifier | Meaning |
|---|---|
| `free` | Place job on any vnode(s). |
| `pack` | All chunks will be taken from one host. |
| `scatter` | Only one chunk with any MPI processes will be taken from a host. A chunk with no MPI processes may be taken from the same vnode as another chunk. |
| `exclusive` | Only this job uses the vnodes chosen. |
| `shared` | This job can share the vnodes chosen. |

**Table 3: Placement Modifiers**

| Modifier | Meaning |
|---|---|
| `group=resource` | Chunks will be grouped according to a resource. All vnodes in the group must have a common value for the resource, which can be either the built-in resource host or a site-defined vnode-level resource. |

Note that vnodes can have sharing attributes that override job placement requests. See the `pbs_node_attributes(7B)` man page.

Grouping by resource name will override `node_group_key`. To run a job on a single host, use "-lplace=pack".

### 4.6.1 Vnodes Allocated to a Job

The nodes file contains the names of the vnodes allocated to a job. The nodes file's name is given by the environment variable PBS_NODEFILE. The order in which hosts appear in the file is the order in which chunks are specified in the selection directive. The order in which hostnames appear in the file is hostA X times, hostB Y times, where X is the number of MPI processes on hostA, Y is the number of MPI processes on hostB, etc. See the definition of the resources "mpiprocs" and "ompthreads" in "PBS Resources" on page 32. See also "The mpiprocs Resource" on page 172.

## 4.7 Examples of Submitting Jobs Using Select and Place

Unless otherwise specified, the vnodes allocated to the job will be allocated as shared or exclusive based on the setting of the vnode's sharing attribute. Each of the following shows how you would use -l select= and -l place=.

1. A job that will fit in a single host such as an Altix but not in any of the vnodes, packed into the fewest vnodes:

```
-l select=1:ncpus=10:mem=20gb
-l place=pack
```

In earlier versions, this would have been:

```
-lncpus=10,mem=20gb
```

2. Request four chunks, each with 1 CPU and 4GB of memory taken from anywhere.

```
-l select=4:ncpus=1:mem=4GB
-l place=free
```

3. Allocate 4 chunks, each with 1 CPU and 2GB of memory from between one and four vnodes which have an arch of “linux”.

```
-l select=4:ncpus=1:mem=2GB:arch=linux -l place=free
```

4. Allocate four chunks on 1 to 4 vnodes where each vnode must have 1 CPU, 3GB of memory and 1 node-locked dyna license available for each chunk.

```
-l select=4:dyna=1:ncpus=1:mem=3GB -l place=free
```

5. Allocate four chunks on 1 to 4 vnodes, and 4 floating dyna licenses. This assumes “dyna” is specified as a server dynamic resource.

```
-l dyna=4 -l select=4:ncpus=1:mem=3GB -l place=free
```

6. This selects exactly 4 vnodes where the arch is linux, and each vnode will be on a separate host. Each vnode will have 1 CPU and 2GB of memory allocated to the job.

```
-lselect=4:mem=2GB:ncpus=1:arch=linux -lplace=scatter
```

7. This will allocate 3 chunks, each with 1 CPU and 10GB of memory. This will also reserve 100mb of scratch space if scratch is to be accounted . Scratch is assumed to be on a file system common to all hosts. The value of “place” depends on the default which is “place=free”.

```
-l scratch=100mb -l select=3:ncpus=1:mem=10GB
```

8. This will allocate 2 CPUs and 50GB of memory on a host named zooland. The value of “place” depends on the default which defaults to “place=free”:

```
-l select=1:ncpus=2:mem=50gb:host=zooland
```

9. This will allocate 1 CPU and 6GB of memory and one host-locked swlicense from each of two hosts:

```
-l select=2:ncpus=1:mem=6gb:swlicense=1
-lplace=scatter
```

10. Request free placement of 10 CPUs across hosts:

```
-l select=10:ncpus=1
-l place=free
```

11. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We request an odd number of CPUs that are not shared, so they must be “rounded up”:

```
-l select=1:ncpus=3:mem=6gb
-l place=pack:excl
```

12. Here is an odd-sized job that will fit on a single Altix, but not on any one node-board. We are asking for small number of CPUs but a large amount of memory:

```
-l select=1:ncpus=1:mem=25gb
-l place=pack:excl
```

13. Here is a job that may be run across multiple Altix systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb
-l place=free
```

14. Submit a job that must be run across multiple Altix systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb
-l place=scatter
```

15. Request free placement across nodeboards within a single host:

```
-l select=1:ncpus=10:mem=10gb
-l place=group=host
```

16. Request free placement across vnodes on multiple Altixes:

```
-l select=10:ncpus=1:mem=1gb
-l place=free
```

17. Here is a small job that uses a shared cpuset:

```
-l select=1:ncpus=1:mem=512kb
-l place=pack:shared
```

18. Request a special resource available on a limited set of nodeboards, such as a graphics card:

```
-l select=1:ncpus=2:mem=2gb:graphics=True
     + 1:ncpus=20:mem=20gb:graphics=False
-l place=pack:excl
```

19. Align SMP jobs on c-brick boundaries:

```
-l select=1:ncpus=4:mem=6gb
-l place=pack:group=cbrick
```

20. Align a large job within one router, if it fits within a router:

```
-l select=1:ncpus=100:mem=200gb
-l place=pack:group=router
```

21. Fit large jobs that do not fit within a single router into as few available routers as possible. Here, RES is the resource used for node grouping:

```
-l select=1:ncpus=300:mem=300gb
-l place=pack:group=<RES>
```

#### 4.7.0.1 Examples Using Old Syntax

22. Request CPUs and memory on a single host using old syntax:

```
-l ncpus=5,mem=10gb
```

will be converted into the equivalent:

```
-l select=1:ncpus=5:mem=10gb
-l place=pack
```

23. Request CPUs and memory on a named host along with custom resources including a floating license using old syntax:

```
-l ncpus=1,mem=5mb,host=origin3,opti=1,platform=IRIX64
```

is converted to the equivalent:

```
-l select=1:ncpus=1:mem=5gb:host=origin3:platform=IRIX64
-l place=pack
-l opti=1
```

24. Request one host with a certain property using old syntax:

```
-lnodes=1:property
```

is converted to the equivalent:

```
-l select=1:ncpus=1:property=True
-l place=scatter
```

25. Request 2 CPUs on each of four hosts with a given property using old syntax:

```
-lnodes=4:property:ncpus=2
```

is converted to the equivalent:

```
-l select=4: ncpus=2:property=True
-l place=scatter
```

26. Request 1 CPU on each of 14 hosts asking for certain software, licenses and a job limit amount of memory using old syntax:

```
-lnodes=14:mpi-fluent:ncpus=1 -lfluent=1,fluent-all=1,fluent-par=13
-l mem=280mb
```

is converted to the equivalent:

```
-l select=14:ncpus=1:mem=20mb:mpi_fluent=True
-l place=scatter
-l fluent=1,fluent-all=1,fluent-par=13
```

27. Requesting licenses using old syntax:

```
-lnodes=3:dyna-mpi-Linux:ncpus=2 -ldyna=6,mem=100mb,software=dyna
```

is converted to the equivalent:

```
-l select=3:ncpus=2:mem=33mb: dyna-mpi-Linux=True
-l place=scatter
-l software=dyna
-l dyna=6
```

28. Requesting licenses using old syntax:

```
 -l ncpus=2,app_lic=6,mem=200mb -l software=app
```

is converted to the equivalent:

```
-l select=1:ncpus=2:mem=200mb
-l place=pack
-l software=app
-l app_lic=6
```

29. Additional example using old syntax:

```
-lnodes=1:fserver+15:noserver
```

is converted to the equivalent:

```
-l select=1:ncpus=1:fserver=True + 15:ncpus=1:noserver=True
-l place=scatter
```

but could also be more easily specified with something like:

```
-l select=1:ncpus=1:fserver=True + 15:ncpus=1:fserver=False
-l place=scatter
```

30. Allocate 4 vnodes, each with 6 CPUs with 3 MPI processes per vnode, with each vnode on a separate host. The memory allocated would be one-fourth of the memory specified by the queue or server default if one existed. This results in a different placement of the job from version 5.4:

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

is converted to:

```
-l select=4:ncpus=6:mpiprocs=3 -l place=scatter
```

31. Allocate 4 vnodes, from 4 separate hosts, with the property blue. The amount of memory allocated from each vnode is 2560MB ( = 10GB / 4) rather than 10GB from each vnode.

-l nodes=4:blue:ncpus=2 -l mem=10GB

is converted to:

-l select=4:blue=True:ncpus=2:mem=2560mb \
-lplace=scatter

## 4.8 Backward Compatibility

For backward compatibility, a legal node specification or resource specification will be converted into selection and placement directives. Specifying “cpp” is part of the old syntax, and should be replaced with “ncpus”. Do not mix old style resource or node specification syntax with select and place statements. If a job is submitted using -l select on the command line, and it contains an old-style specification in the job script, that will result in an error.

### 4.8.1 Node Specification Conversion

Node specification format:

`-lnodes=[N:`*spec_list* | *spec_list*`]`
`[[+N:`*spec_list* | +*spec_list*`] ...]`
`[`*#suffix* `...][-lncpus=Z]`

where:

*spec_list* has syntax: *spec*[:*spec* ...]
*spec* is any of: hostname | property | ncpus=X | cpp=X | ppn=P
*suffix* is any of: property | excl | shared
N and P are positive integers
X and Z are non-negative integers

The node specification is converted into selection and placement directives as follows:

Each *spec_list* is converted into one chunk, so that N:*spec_list* is converted into N chunks.

*If spec is hostname :*

The chunk will include host=hostname

*If spec matches any vnode's resources_available.host value:*

The chunk will include host=hostname

*If spec is property :*

The chunk will include property=true
Property must be a site-defined vnode-level boolean resource.

*If spec is ncpus=X or cpp=X :*

The chunk will include ncpus=X

*If no spec is ncpus=X and no spec is cpp=X :*

The chunk will include ncpus=1

*If spec is ppn=P :*

The chunk will include mpiprocs=P

Example:

```
-lnodes=4:ppn=2
```

is converted into

```
-lselect=4:ncpus=2:mpiprocs=2
```

*If -lncpus=Z is specified and no spec contains ncpus=X and no spec is cpp=X :*

Every chunk will include ncpus=W,
where W is Z divided by the total number of chunks.
(Note: W must be an integer; Z must be evenly divisible by the number of chunks.)

*If property is a suffix :*

All chunks will include property=true

*If excl is a suffix :*

The placement directive will be -lplace=scatter:excl

*If shared is a suffix :*

The placement directive will be -lplace=scatter:shared

*If neither excl nor shared is a suffix :*

The placement directive will be -lplace=scatter

Example:

```
-l nodes=3:green:ncpus=2:ppn=2+2:red
```

is converted to:

```
-l select=3:green=true:ncpus=4:mpiprocs=2+2 \
:red=true:ncpus=1
-l place=scatter
```

Node specification syntax for requesting properties is deprecated. The new boolean resource syntax "property=true" is only accepted in a selection directive. It is erroneous to mix old and new syntax.

### 4.8.2 Resource Specification Conversion

The resource specification is converted to select and place statements after any defaults have been applied.

Resource specification format:

`-l`*resource*=*value*[:*resource*=*value* ...]

The resource specification is converted to:

```
-lselect=1[:resource=value ...]
-lplace=pack
```

with one instance of *resource=value* for each of the following vnode-level resources in the resource request:

built-in resources: ncpus | mem | vmem | arch | host

site-defined vnode-level resources l

## 4.9 How PBS Parses a Job Script

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

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

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

## 4.10 A Sample PBS Job

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

```
UNIX                              Windows

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

On line one in the example above Windows does not show a shell directive. (The default on Windows is the batch command language.) Also note that it is possible under both Windows and UNIX to specify to PBS the scripting language to use to interpret the job script (see the "-S" option to `qsub` in section 4.13.9 "Specifying Scripting Language to Use" on page 67). The Windows script will be a .exe or .bat file.

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

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

The section “Job Submission Options” on page 62 describes how to use the `qsub` command to submit PBS jobs. Any option that you specify to the `qsub` command line (except “`-I`”) can also be provided as a PBS directive inside the PBS script. PBS directives come in two types: resource requirements and attribute settings.

In our example above, lines 2-3 specify the “`-l`” resource list option, followed by a specific resource request. Specifically, lines 2-3 request 1 hour of wall-clock time as a job-wide request, and 400 megabytes (MB) of memory in a chunk. .

Line 4 requests that PBS *join* the `stdout` and `stderr` output streams of the job into a single stream.

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

## 4.11 Changing the Job’s PBS Directive

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

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

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

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

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

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

or

```
qsub -C “REM PBS” my_job_script
```

For additional details on the “`-C`” option to `qsub`, see section 4.13 “Job Submission Options” on page 62.

## 4.12 Windows Jobs

### 4.12.1 Submitting Windows Jobs

Any .bat files that are to be executed within a PBS job script have to be prefixed with "call" as in:

```
---[job_b.bat]----------
@echo off
call E:\step1.bat
call E:\step2.bat
-----------------------
```

Without the "call", only the first .bat file gets executed and it doesn't return control to the calling interpreter.

An example:

A job script that contains:

```
--[job_a.bat]---------
@echo off
E:\step1.bat
E:\step2.bat
```

should now be:

```
--[job_a.bat]---------
@echo off
call E:\step1.bat
```

call E:\step2.bat

### 4.12.2 Passwords

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

#### 4.12.2.1 Single-Signon Password Method

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

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

When no options are given to pbs_password, the password credential on the default PBS server for the current user, i.e. the user who executes the command, is updated to the prompted password. Any user jobs previously held due to an invalid password are not released.

The available options to `pbs_password` are:

| | |
|---|---|
| -r | Any user jobs previously held due to an invalid password are released. |
| -s server | Allows user to specify server where password will be changed. |
| -d | Deletes the password. |
| user | The password credential of user user is updated to the prompted password. If user is not the current user, this action is only allowed if:<br><br>1. The current user is root or admin.<br><br>2. User user has given the current user explicit access via the ruserok() mechanism: |

a. The hostname of the machine from which the current user is logged in appears in the server's hosts.equiv file, or

b. The current user has an entry in user's HOMEDIR\.rhosts file.

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

#### 4.12.2.2 Per-job Password Method

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

```
qsub -Wpwd=”<password>” job.script
```

The password specified will be shown on screen and will be passed onto the program, which will then encrypt it and save it securely for use by the job. The password should be enclosed in double quotes. If you only type the pair of double quotes, you will be prompted for the password.

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

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

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

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

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

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

## 4.13 Job Submission Options

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

**Table 4: Options to the `qsub` Command**

| Option | Function and Page Reference |
|---|---|
| `-A account_string` | “Specifying a local account” on page 72 |
| `-a date_time` | “Deferring Execution” on page 68 |
| `-C “DPREFIX”` | “Changing the Job’s PBS Directive” on page 58 |
| `-c interval` | “Specifying Job Checkpoint Interval” on page 69 |
| `-e path` | “Redirecting Output and Error Files” on page 64 |
| `-h` | “Holding a job (Delaying Execution)” on page 68 |
| `-I` | “Interactive-batch Jobs” on page 73 |
| `-J X-Y[:Z]` | “Job Array” on page 153 |
| `-j` *`join`* | “Merging Output and Error Files” on page 72 |
| `-k` *`keep`* | “Retaining Output and Error Files on Execution Host” on page 72 |
| `-l resource_list` | “New Rules for Submitting Jobs” on page 36 |
| `-M user_list` | “Setting Email Recipient List” on page 66 |
| `-m MailOptions` | “Specifying Email Notification” on page 65 |
| `-N name` | “Specifying a Job Name” on page 66 |
| `-o path` | “Redirecting Output and Error Files” on page 64 |

**Table 4: Options to the `qsub` Command**

| Option | Function and Page Reference |
|---|---|
| `-p priority` | "Setting a job's priority" on page 68 |
| `-q destination` | "Specifying Queue and/or Server" on page 63 |
| `-r value` | "Marking a Job as "Rerunnable" or Not" on page 67 |
| `-S path_list` | "Specifying Scripting Language to Use" on page 67 |
| `-u user_list` | "Specifying Job User ID" on page 69 |
| `-V` | "Exporting Environment Variables" on page 65 |
| `-v variable_list` | "Expanding Environment Variables" on page 65 |
| `-W depend=list` | "Specifying Job Dependencies" on page 129 |
| `-W group_list=list` | "Specifying Job Group ID" on page 71 |
| `-W stagein=list` | "Input/Output File Staging" on page 132 |
| `-W stageout=list` | "Input/Output File Staging" on page 132 |
| `-W cred=dce` | "Running PBS in a UNIX DCE Environment" on page 150 |
| `-W block=opt` | "Requesting qsub Wait for Job Completion" on page 128 |
| `-W pwd="password"` | "Per-job Password Method" on page 61 and "Running PBS in a UNIX DCE Environment" on page 150 |
| `-W umask=nnn` | "Changing UNIX Job umask" on page 128 |
| `-z` | "Suppressing Job Identifier" on page 73 |

### 4.13.1 Specifying Queue and/or Server

The "`-q destination`" option to `qsub` allows you to specify a particular destination to which you want the job submitted. The *destination* names a queue, a Server, or a queue at a Server. The `qsub` command will submit the script to the Server defined by the *destination* argument. If the *destination* is a routing queue, the job may be routed by the Server to a new destination. If the `-q` option is not specified, the `qsub` command will submit the script to the default queue at the default Server. (See also the discussion of **PBS_DEFAULT** in “Environment Variables” on page 28.) The destination specification takes the following form:

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

```
qsub -q queue my_job

qsub -q @server my_job
```

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

```
qsub -q queueName@serverName my_job

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

### 4.13.2 Redirecting Output and Error Files

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

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

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

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

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

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

### 4.13.3 Exporting Environment Variables

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

```
qsub -V my_job
```

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

### 4.13.4 Expanding Environment Variables

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

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

### 4.13.5 Specifying Email Notification

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

| | |
|---|---|
| `a` | send mail when job is *aborted* by batch system |
| `b` | send mail when job *begins* execution |
| `e` | send mail when job *ends* execution |

`n` *do not* send mail

```
qsub -m ae my_job
```

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

### 4.13.6 Setting Email Recipient List

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

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

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

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

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

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

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

### 4.13.7 Specifying a Job Name

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

```
qsub -N myName my_job
```

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

### 4.13.8 Marking a Job as “Rerunnable” or Not

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

```
qsub -r n my_job
```

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

### 4.13.9 Specifying Scripting Language to Use

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

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

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

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

### 4.13.10 Setting a job's priority

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

This option allows the user to specify a priority for their jobs. However, this option is dependant upon the local scheduling policy. By default the "sort jobs by job-priority" feature is disabled. If your local PBS administrator has enabled it, then all queued jobs will be sorted based on the user-specified priority. (If you need an absolute ordering of your own jobs, see "Specifying Job Dependencies" on page 129.)

```
qsub -p 120 my_job
```

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

### 4.13.11 Deferring Execution

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

```
qsub -a 0700 my_job
```

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

### 4.13.12 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 118.)

```
qsub -h my_job
```

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

### 4.13.13 Specifying Job Checkpoint Interval

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

| | |
|---|---|
| `n` | No checkpointing is to be performed. |
| `s` | Checkpointing is to be performed only when the Server executing the job is shutdown. |
| `c` | Checkpointing is to be performed at the default minimum time for the Server executing the job. |
| `c`=minutes | Checkpointing is to be performed at an interval of *minutes*, which is the integer number of minutes of CPU time used by the job. This value must be greater than zero. |
| `u` | Checkpointing is unspecified, thus resulting in the same behavior as "s". |

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

```
qsub -c c my_job
```

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

Checkpointing is not supported for job arrays.

### 4.13.14 Specifying Job User ID

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

For example, our user is UserS on the submission host HostS, UserA on server ServerA, and UserB on server ServerB, and is UserC everywhere else. Note that this user must be UserA on all ExecutionA and UserB on all ExecutionB machines. Then our user can use "qsub -u UserA@ServerA,UserB@ServerB,UserC" for the job. The job owner will always be UserS.

#### 4.13.14.1 qsub -u: User ID with UNIX

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

**Table 5: UNIX User ID and flatuid**

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

Note that if different names are listed via the `-u` option, then they are checked regardless of the value of `flatuid`.

#### 4.13.14.2 qsub -u: User ID with Windows

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

**Table 6: Requirements for Admin User to Submit Job**

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

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

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

### 4.13.15 Specifying Job Group ID

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. Under Windows, the primary group is the first group found for the user by PBS when querying the accounts database.

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

### 4.13.16 Specifying a local account

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

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

```
qsub -A Math312 my_job
```

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

### 4.13.17 Merging Output and Error Files

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

```
qsub -j oe my_job
```

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

### 4.13.18 Retaining Output and Error Files on Execution Host

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

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

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

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

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

`n` Neither file is retained.

```
qsub -k oe my_job
```

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

### 4.13.19 Suppressing Job Identifier

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

```
qsub -z my_job
```

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

### 4.13.20 Interactive-batch Jobs

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

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

**Important:** Interactive-batch jobs do not support job arrays.

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

When an interactive job is submitted, the `qsub` command will not terminate when the job is submitted. `qsub` will remain running until the job terminates, is aborted, or the user interrupts `qsub` with a SIGINT (the control-C key). If `qsub` is interrupted prior to job start, it will query if the user wishes to exit. If the user responds “yes”, `qsub` exits and the job is aborted.

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

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

## 4.14 Job Attributes

A PBS job has the following attributes, which may be set by the various options to `qsub` (for details see section 4.13 “Job Submission Options” on page 62).

| | |
|---|---|
| `Account_Name` | Reserved for local site accounting. If specified (using the `-A` option to `qsub`) this value is carried within the job for its duration, and is included in the job accounting records. |
| `block` | When true, specifies that `qsub` will wait for the job to complete, and return the exit value of the job. Default: false. Set |

via the *-W block* option to qsub. If `qsub` receives one of the signals: SIGHUP, SIGINT, SIGQUIT or SIGTERM, it will print the following message on stderr: `qsub: wait for job <jobid> interrupted by signal <signal>`

| | |
|---|---|
| `Checkpoint` | If supported by the Server implementation and the host operating system, the checkpoint attribute determines when checkpointing will be performed by PBS on behalf of the job. The legal values for checkpoint are described under the `qalter` and `qsub` commands. |
| `depend` | The type of inter-job dependencies specified by the job owner. |
| `Error_Path` | The final path name for the file containing the job's standard error stream. See the `qsub` and `qalter` command description for more detail. |
| `Execution_Time` | The time after which the job may execute. The time is maintained in seconds since Epoch. If this time has not yet been reached, the job will not be scheduled for execution and the job is said to be in *wait* state. |
| `group_list` | A list of *group_names@hosts* which determines the group under which the job is run on a given host. When a job is to be placed into execution, the Server will select a group name according to the rules specified for use of the `qsub` command. |
| `Hold_Types` | The set of holds currently applied to the job. If the set is not null, the job will not be scheduled for execution and is said to be in the *hold* state. Note, the *hold* state takes precedence over the *`wait`* state. |
| `Job_Name` | The name assigned to the job by the `qsub` or `qalter` command. |
| `Join_Path` | If the `Join_Path` attribute is `oe`, then the job's standard error stream will be merged, inter-mixed, with the job's standard output stream and placed in the file determined by the |

| | |
|---|---|
| | `Output_Path` attribute. The `Error_Path` attribute is maintained, but ignored. However, if the `Join_Path` attribute is `eo`, then the job's standard output stream will be merged, intermixed, with the job's standard error stream and placed in the file determined by the `Error_Path` attribute, and the `Output_Path` attribute will be ignored. |
| `Keep_Files` | If `Keep_Files` contains the values "`o`" `KEEP_OUTPUT` and/or "`e`" `KEEP_ERROR` the corresponding streams of the batch job will be retained on the execution host upon job termination. `Keep_Files` overrides the `Output_Path` and `Error_Path` attributes. |
| `Mail_Points` | Identifies when the Server will send email about the job. |
| `Mail_Users` | The set of users to whom mail may be sent when the job makes certain state changes. |
| `no_stdio_sockets` | Flag to indicate whether a multi-host job should have the standard output and standard error streams of tasks running on other hosts returned to mother superior via sockets. These sockets may cause a job to be not checkpointable. Default: false (sockets are created.) |
| `Output_Path` | The final path name for the file containing the job's standard output stream. See the `qsub` and `qalter` command description for more detail. |
| `Priority` | The job scheduling priority assigned by the user. |
| `Rerunnable` | The rerunnable flag given by the user. |
| `Resource_List` | The resource list is a set of resources required by the job. The value also establishes the limit of usage of that resource. If not set, the value for a resource may be determined by a queue or Server default established by the administrator. |
| `Shell_Path_List` | A set of absolute paths of the program to process the job's script file. |

| | |
|---|---|
| `stagein` | The list of files to be staged in prior to job execution. |
| `stageout` | The list of files to be staged out after job execution. |
| `umask` | The initial umask of the job is set to the value of this attribute when the job is created. This may be changed by umask commands in the shell initialization files such as .profile or .cshrc. Default value: 077 |
| `User_List` | The list of *user@host* which determines the username under which the job is run on a given host. |
| `Variable_List` | This is the list of environment variables passed with the *Queue Job* batch request. |
| `comment` | An attribute for displaying comments about the job from the system. Visible to any client. |

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

| | |
|---|---|
| `accounting_id` | Accounting ID for tracking accounting data not produced by PBS. |
| `alt_id` | For a few systems, such as Irix 6.x running Array Services, the session id is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it will also be recorded in the end-of-job accounting record. For IRIX 6.x running Array Services, the `alt_id` attribute is set to the Array Session Handle (ASH) assigned to the job. |
| `array` | boolean; true if applied to a job array |
| `array_id` | string; applies to subjob; job array identifier for given subjob |
| `array_index` | string; applies to subjob; index number of given subjob |
| `array_indices_remaining` | string; applies to job array; list of indices of subjobs still |

| | |
|---|---|
| | queued. Range or list of ranges |
| `array_indices_submitted` | string; applies to job array; complete list of indices of subjobs given at submission time. Given as a range. |
| `array_state_count` | string; applies to job array; lists number of subjobs in each state |
| `ctime` | The time that the job was created. |
| `etime` | The time that the job became eligible to run, i.e. in a queued state while residing in an execution queue. |
| `exec_host` | If the job is running, string set to the name of each vnode on which the job is executing, along with the vnode-level, consumable resources allocated from that vnode.<br>Format:<br>"(vnode:ncpus=N:mem=M+vnode:ncpus=N:mem=M[+...])", where *vnode* is the name of a vnode, *N* is the number of CPUs on that vnode allocated to the job and *M* is the amount of memory on that vnode allocated to the job. Other resources may show up as well. |
| `egroup` | If the job is queued in an execution queue, this attribute is set to the group name under which the job is to be run. [This attribute is available only to the batch administrator.] |
| `euser` | If the job is queued in an execution queue, this attribute is set to the user name under which the job is to be run. [This attribute is available only to the batch administrator.] |
| `hashname` | The name used as a basename for various files, such as the job file, script file, and the standard output and error of the job. [This attribute is available only to the batch administrator.] |
| `interactive` | True if the job is an interactive PBS job. |
| `Job_Owner` | The login name on the submitting host of the user who submitted the batch job. |

| | |
|---|---|
| `job_state` | The state of the job. |
| `mtime` | The time that the job was last modified, changed state, or changed locations. |
| `qtime` | The time that the job entered the current queue. |
| `queue` | The name of the queue in which the job currently resides. |
| `queue_rank` | The job's position in the queue. Set by server. Read-only. Requires operator or administrator privilege to view. Integer. |
| `resources_used` | The amount of resources used by the job. This is provided as part of job status information if the job is running. |
| `run_count` | The number of times the server has run the job. Format: integer. |
| `schedselect` | This is set to the union of the "select" resource of the job and the queue and server defaults for resources in a chunk. |
| `server` | The name of the server which is currently managing the job. |
| `session_id` | If the job is running, this is set to the session id of the first executing task. |

# Chapter 5
# Using the xpbs GUI

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

## 5.1 Starting xpbs

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

UNIX:

```
xpbs &
```

Windows:

```
xpbs.exe
```

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

### 5.1.1 Running `xpbs` Under **`UNIX`**

Before running xpbs for the first time under UNIX, you may need to configure your workstation for it. Depending on how PBS is installed at your site, you may need to allow `xpbs` to be displayed on your workstation. However, if the PBS client commands are installed locally on your workstation, you can skip this step. (Ask your PBS administrator if you are unsure.)

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

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

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

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

```
setenv DISPLAY myWorkstation:0.0
```

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

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

## 5.2 Using xpbs: Definitions of Terms

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

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

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

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

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

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

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

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

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

## 5.3 Introducing the xpbs Main Display

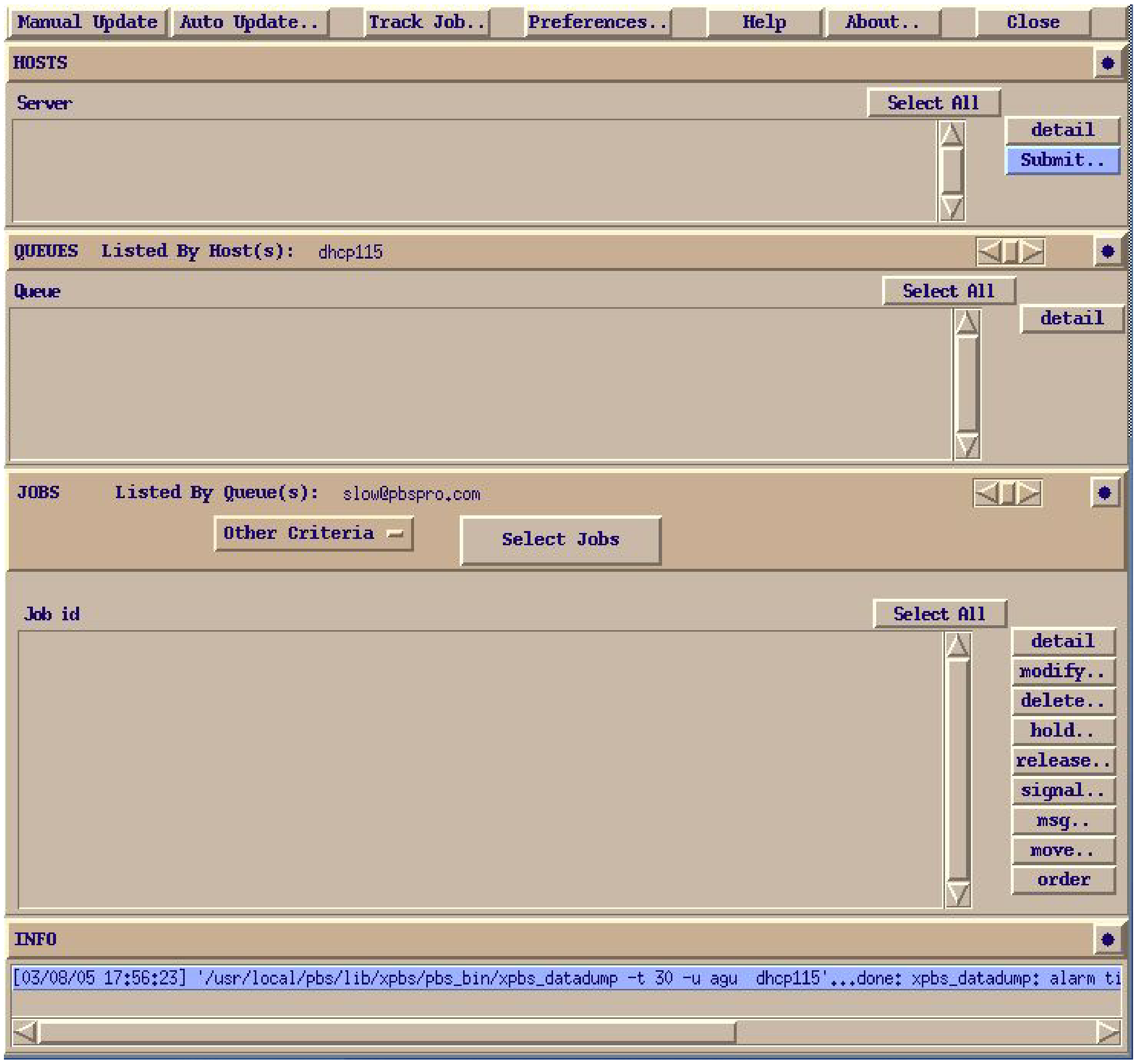
The main window or display of `xpbs` is comprised of five collapsible subwindows or *panels*. Each panel contains specific information. Top to bottom, these panels are: the Menu Bar, Hosts panel, Queues panel, Jobs panel, and the Info panel.

### 5.3.1 xpbs Menu Bar

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

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

.



```
[03/08/05 17:56:23] '/usr/local/pbs/lib/xpbs/pbs_bin/xpbs_datadump -t 30 -u agu  dhcp115'...done: xpbs_datadump: alarm ti
```

### 5.3.2 xpbs Hosts Panel

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

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

| | |
|---|---|
| detail | Provides information about selected Server host(s). This functionality can also be achieved by double clicking on an entry in the Hosts listbox. |
| submit | For submitting a job to any of the queues managed by the selected host(s). |
| terminate | For terminating (shutting down) PBS Servers on selected host(s). (Visible via the "`-admin`" option only.) |

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

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

**Table 8: xpbs Server Column Headings**

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

**Table 8: xpbs Server Column Headings**

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

### 5.3.3 xpbs Queues Panel

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

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

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

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

**Table 9: xpbs Queue Column Headings**

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

**Table 9: xpbs Queue Column Headings**

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

### 5.3.4 xpbs Jobs Panel

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

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

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

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

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

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

**Table 10: xpbs Job Column Headings**

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

### 5.3.5 xpbs Info Panel

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

### 5.3.6 xpbs Keyboard Tips

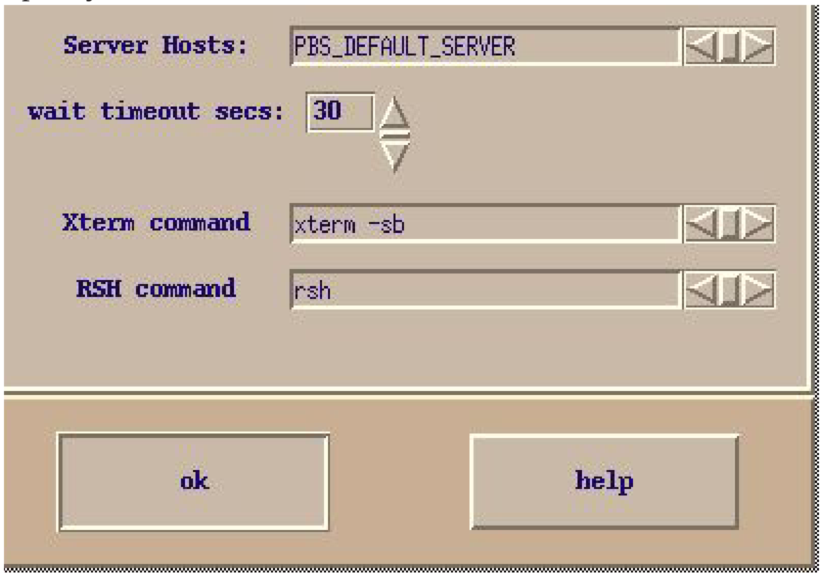
There are a number of shortcuts and key sequences that can be used to speed up using `xpbs`. These include:

Tip 1. All buttons which appear to be depressed in the dialog box/sub-window can be activated by pressing the return/enter key.
Tip 2. Pressing the tab key will move the blinking cursor from one text field to another.
Tip 3. To contiguously select more than one entry: left-click then drag the mouse across multiple entries.
Tip 4. To non-contiguously select more than one entry: hold the con-trol-left-click on the desired entries.

## 5.4 Setting xpbs Preferences

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

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


Server Hosts: PBS_DEFAULT_SERVER
wait timeout secs: 30
Xterm command xterm -sb
RSH command rsh

## 5.5 Relationship Between PBS and `xpbs`

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

**Table 11: `xpbs` Buttons and PBS Commands**

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

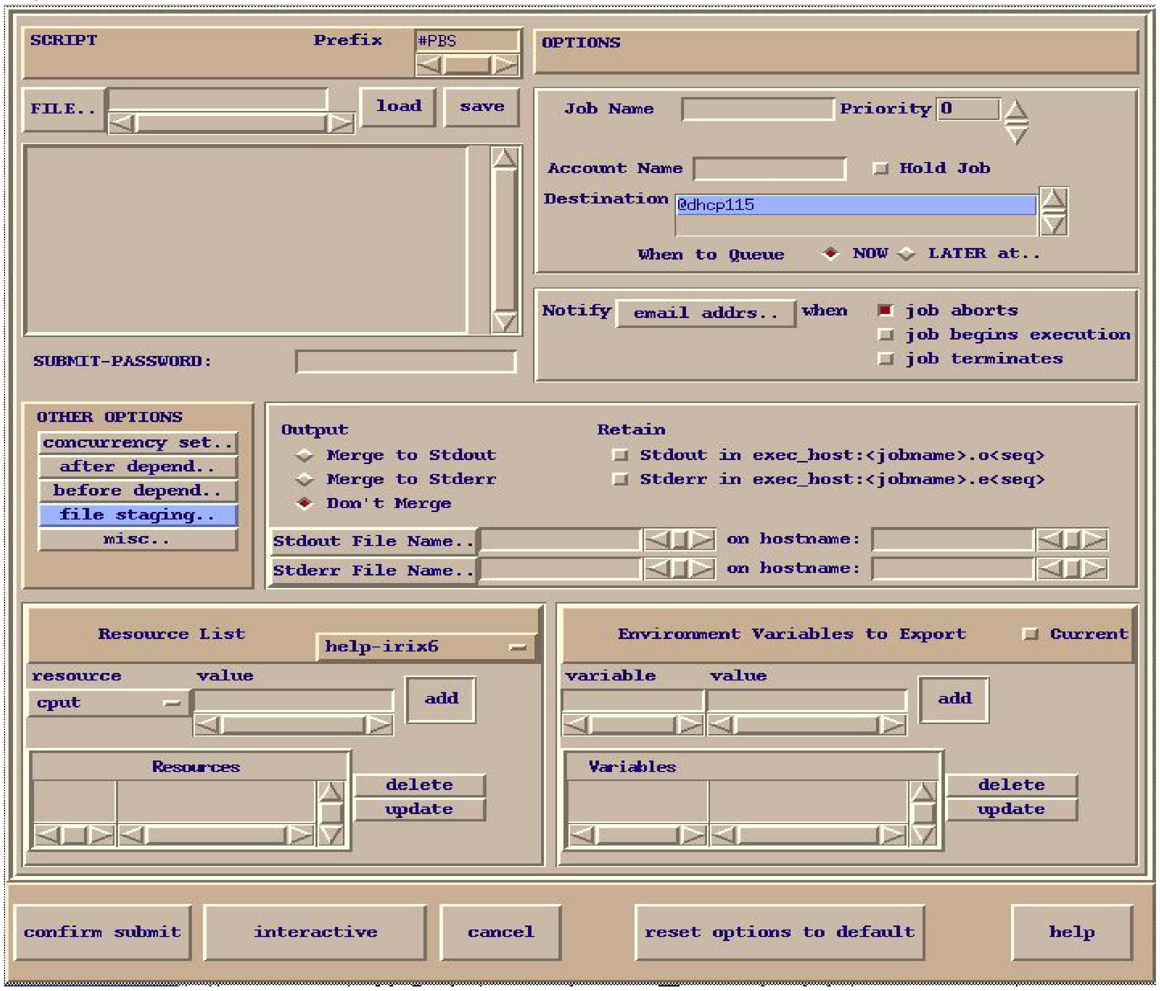
* Indicates command button is visible only if `xpbs` is started with the “`-admin`” option.

## 5.6 How to Submit a Job Using `xpbs`

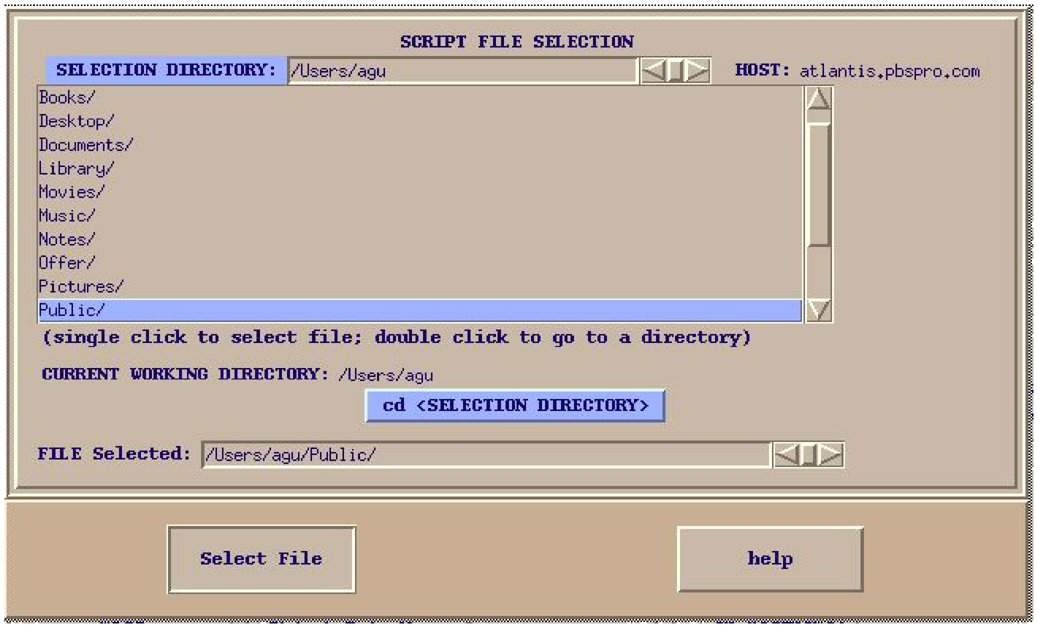
To submit a job using `xpbs`, perform the following steps:

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

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



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


SCRIPT FILE SELECTION
SELECTION DIRECTORY: /Users/agu
HOST: atlantis.pbspro.com
Books/
Desktop/
Documents/
Library/
Movies/
Music/
Notes/
Offer/
Pictures/
Public/
(single click to select file; double click to go to a directory)
CURRENT WORKING DIRECTORY: /Users/agu
cd <SELECTION DIRECTORY>
FILE Selected: /Users/agu/Public/
Select File
help

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 shows 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.

The resources specified in the "Resource List" section will be job-wide resources only. In order to specify chunks or job placement, use a script.

To run an array job, use a script. You will not be able to query individual subjobs or the whole job array using xpbs. Type the script into the "File: entry" box. Do not click the "Load" button. Instead, use the "Submit" button.

Finally, review the optional settings to see if any should apply to this job.

For example:

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

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

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

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

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

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

On UNIX systems (not Windows) you can alternatively submit the job as an interactive-batch job, by clicking the *Interactive* button at the bottom of the Submit Job window. Doing so will cause an X-terminal window (`xterm`) to be launched, and within that window a PBS interactive-batch job submitted. The path for the xterm command can be set via the preferences, as discussed above in section 5.4 “Setting xpbs Preferences” on page 89. For further details on usage, and restrictions, see “Interactive-batch Jobs” on page 73.)

## 5.7 Exiting xpbs

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

## 5.8 The xpbs Configuration File

Upon exit, the `xpbs` state may be written to the `.xpbsrc` file in the user’s home directory. (See also section 3.8.1 “Windows User's HOMEDIR” on page 25.) Information saved includes: the selected host(s), queue(s), and job(s); the different jobs listing criteria; the view states (i.e. minimized/maximized) of the Hosts, Queues, Jobs, and INFO regions; and all settings in the Preferences section. In addition, there is a system-wide `xpbs` configuration file, maintained by the PBS Administrator, which is used in the absence of a user’s personal `.xpbsrc` file.

## 5.9 `xpbs` Preferences

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

| | |
|---|---|
| *serverHosts | List of Server hosts (space separated) to query by `xpbs`. A special keyword **`PBS_DEFAULT_SERVER`** can be used which will be used as a placeholder for the value obtained from the `/etc/pbs.conf` file (UNIX) or “`[PBS Destination Folder]\pbs.conf`” file (Windows). |
| *timeoutSecs | Specify the number of seconds before timing out waiting for a connection to a PBS host. |
| *xtermCmd | The xterm command to run driving an interactive PBS session. |
| *labelFont | Font applied to text appearing in labels. |
| *fixlabelFont | Font applied to text that label fixed-width widgets such as listbox labels. This must be a fixed-width font. |

| | |
|---|---|
| *textFont | Font applied to a text widget. Keep this as fixed-width font. |
| *backgroundColor | The color applied to background of frames, buttons, entries, scrollbar handles. |
| *foregroundColor | The color applied to text in any context. |
| *activeColor | The color applied to the background of a selection, a selected command button, or a selected scroll bar handle. |
| *disabledColor | Color applied to a disabled widget. |
| *signalColor | Color applied to buttons that signal something to the user about a change of state. For example, the color of the *Track Job* button when returned output files are detected. |
| *shadingColor | A color shading applied to some of the frames to emphasize focus as well as decoration. |
| *selectorColor | The color applied to the selector box of a radiobutton or check-button. |
| *selectHosts | List of hosts (space separated) to automatically select/highlight in the HOSTS listbox. |
| *selectQueues | List of queues (space separated) to automatically select/high-light 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 "Own-ers: <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 limit-ing 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 lim-iting 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:<br>{ <arch-type1> resname1 resname2 ... resnameN }<br>{ <arch-type2> resname1 resname2 ... resnameN }<br>{ <arch-typeN> resname1 resname2 ... resnameN } |

# Chapter 6
# Checking Job / System Status

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

## 6.1 The `qstat` Command

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

### 6.1.1 Checking Job Status

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

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

The job state
The queue in which the job resides

The job state is abbreviated to a single character:

| State | Description |
|---|---|
| B | Job arrays only: job array has started |
| E | Job is exiting after having run |
| H | Job is held. A job is put into a held state by the server or by a user or administrator. A job stays in a held state until it is released by a user or administrator. |
| Q | Job is queued, eligible to run or be routed |
| R | Job is running |
| S | Job is suspended by server. A job is put into the suspended state when a higher priority job needs the resources. |
| T | Job is in transition (being moved to a new location) |
| U | Job is suspended due to workstation becoming busy |
| W | Job is waiting for its requested execution time to be reached or job specified a stage-in request which failed for some reason. |
| X | Subjobs only; subjob is finished (expired.) |

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

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

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

Session ID
Number of nodes requested
Number of parallel tasks (or CPUs)
Requested amount of memory
Requested amount of wallclock time
Walltime or CPU time, whichever submitter specified, if job is running.

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

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

### 6.1.2 Viewing Specific Information

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

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

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

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

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

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

**Important:** If a PBS Server is not specified on the `qstat` command line, the default Server will be used. (See discussion of **`PBS_DEFAULT`** in “Environment Variables” on page 28.)

### 6.1.3 Checking Server Status

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

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

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

```
qstat -Bf
Server: fast.mydomain.com
    server_state = Active
    scheduling = True
    total_jobs = 14
    state_count = Transit:0 Queued:13 Held:0 Waiting:0
                  Running:1 Exiting:0
    managers = user1@fast.mydomain.com
    default_queue = workq
    log_events = 511
    mail_from = adm
    query_other_jobs = True
    resources_available.mem = 64mb
    resources_available.ncpus = 2
    resources_default.ncpus = 1
    resources_assigned.ncpus = 1
    resources_assigned.nodect = 1
    scheduler_iteration = 600
    pbs_version = PBSPro_8.0.41640
```

### 6.1.4 Checking Queue Status

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

```
qstat -Q

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

The full display for a queue provides additional information:

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

### 6.1.5 Viewing Job Information

We saw above that the "`-f`" option could be used to display full or long information for queues and Servers. The same applies to jobs. By specifying the "`-f`" option and a job identifier, PBS will print all information known about the job (e.g. resources requested, resource limits, owner, source, destination, queue, etc.) as shown in the following example. (See "Job Attributes" on page 74 for a description of attribute.)

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

### 6.1.6 List User-Specific Jobs

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

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

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

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

qstat -u user1,barry

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

### 6.1.7 List Running Jobs

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

### 6.1.8 List Non-Running Jobs

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

### 6.1.9 Display Size in Gigabytes

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

### 6.1.10 Display Size in Megawords

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

### 6.1.11 List Nodes Assigned to Jobs

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

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

### 6.1.12 Display Job Comments

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

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

### 6.1.13 Display Queue Limits

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

```
qstat -q
server: south

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

### 6.1.14 Show State of Job, Job Array or Subjob

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

### 6.1.15 Show state of Job Arrays

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

### 6.1.16 Print Job Array Percentage Completed

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

## 6.2 Viewing Job / System Status with xpbs

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

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

## 6.3 The `qselect` Command

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

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

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

The available options to `qselect` are:

-a [op]date_time

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

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

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

-A account_string

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

-c [ op ] interval — Restricts selection to jobs whose *Checkpoint* interval attribute matches the specified relationship. The values of the *Checkpoint* attribute are defined to have the following ordered relationship:

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

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

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

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

-J — Shows only job array identifiers

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

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

| | |
|---|---|
| -N name | Restricts selection of jobs to those with a specific name. |
| -p [op]priority | Restricts selection of jobs to those with a priority that matches the specified relationship. If `op` is not specified, jobs are selected for which the job Priority attribute is equal to the priority. |
| -q destination | Restricts selection to those jobs residing at the specified destination. The destination may be one of the following three forms: `queue` `@server` `queue@server` If the `-q` option is not specified, jobs will be selected from the default Server. If the destination describes only a queue, only jobs in that queue on the default batch Server will be selected. If the destination describes only a Server, then jobs in all queues on that Server will be selected. If the destination describes both a queue and a Server, then only jobs in the named queue on the named Server will be selected. |
| -r rerun | Restricts selection of jobs to those with the specified *Rerunnable* attribute. The option argument must be a single character. The following two characters are supported by PBS: `y` and `n`. |
| -s states | Restricts job selection to those in the specified states. The *states* argument is a character string which consists of any combination of the characters: `B`,`E`, `H`, `Q`, `R`, `S`, `T`, `U`, `W` and `X`. The characters in the *states* argument have the following interpretation: |
| -t | Shows job, job array and subjob identifiers |

**Table 12: Job States**

| **State** | **Meaning** |
|---|---|
| B | Job array has started execution. |
| E | Job is in the process of Exiting. |
| H | Job has been placed on Hold. |
| Q | Job is in the Queued state. |
| R | Job is in the Running state. |
| S | Job has been Suspended. |
| T | Job is Transiting between states. |
| U | Job suspended due to workstation user activity |
| W | Job is in the Waiting state. |
| X | Subjob has completed execution or been deleted. |

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

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

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

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

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

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

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

UNIX:

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

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

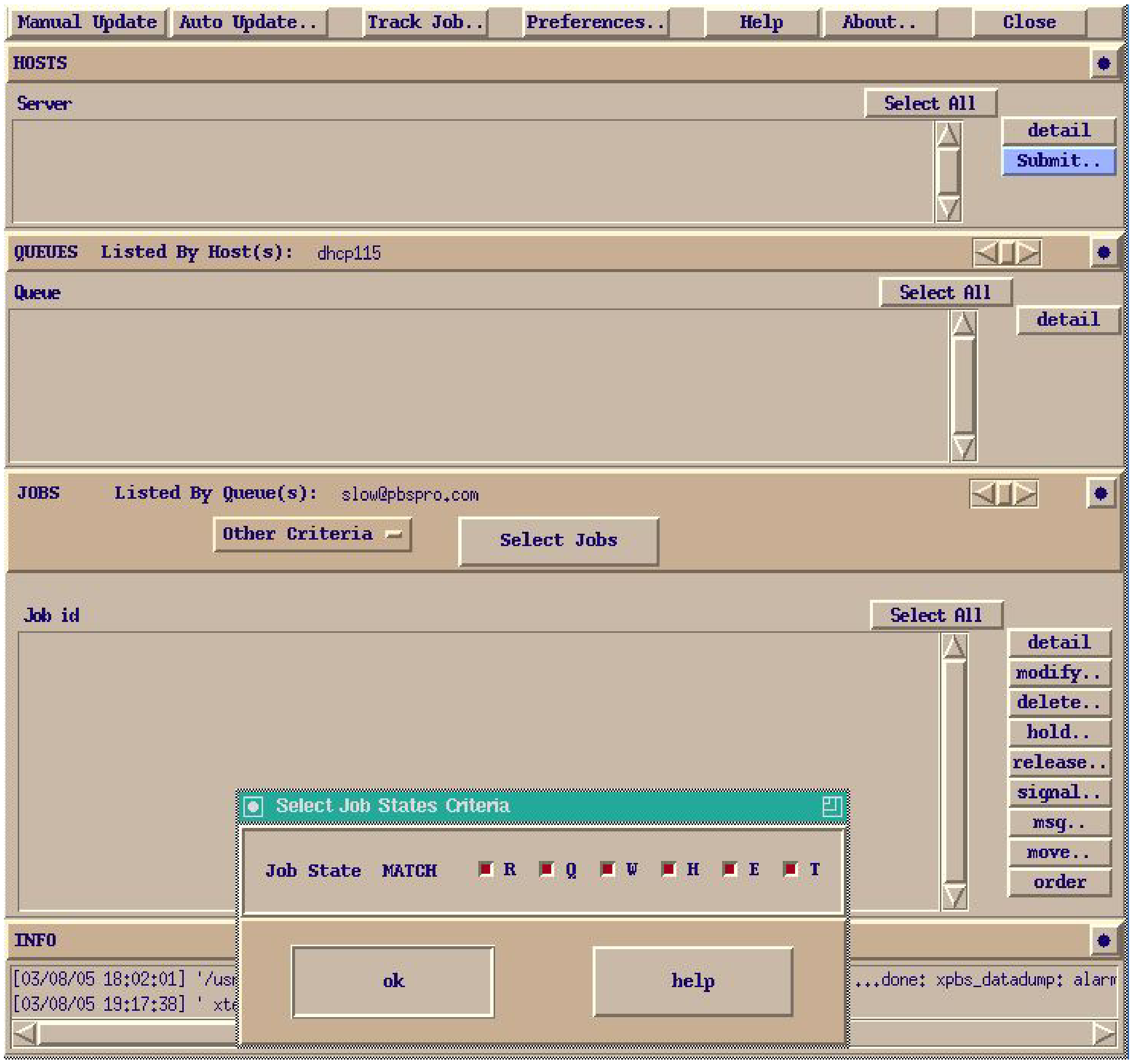
```
for /F "usebackq" %j in (`qselect -u barry -l ncpus.gt.16`)  do
( qstat -a %j )
121.south
133.south
154.south
```

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

## 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.



```
[03/08/05 18:02:01] '/usr ...done: xpbs_datadump: alarm
[03/08/05 19:17:38] ' xte
```

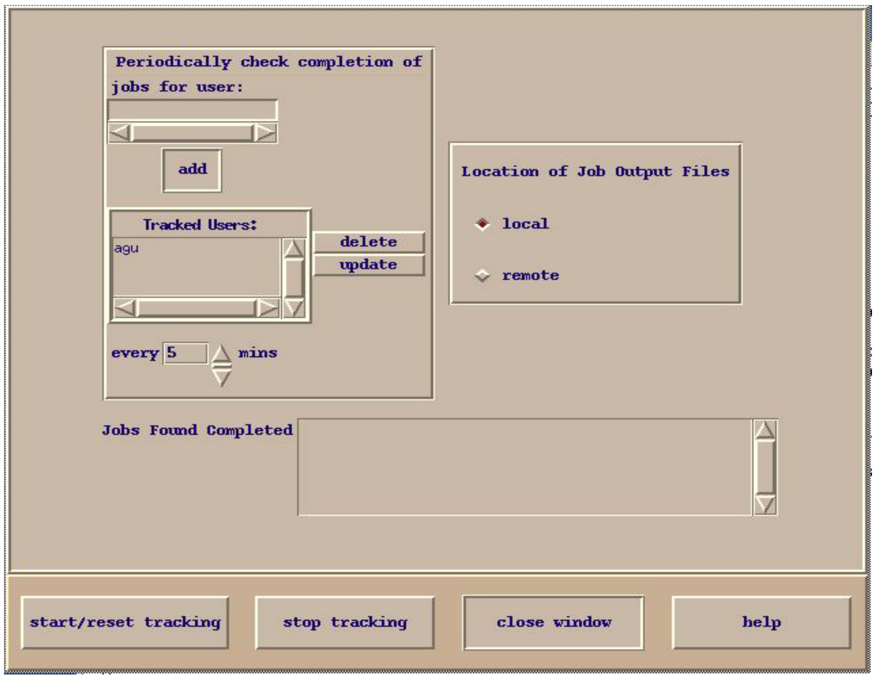
You may specify as many or as few selection criteria as you wish. When you have completed your selection, click on the *Select Jobs* button above the HOSTS panel to have `xpbs` refresh the display with the jobs that match your selection criteria. The selected criteria will remain in effect until you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

## 6.5 Using `xpbs` TrackJob Feature

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

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

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



Tracked Users: agu

every 5 mins

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

# Chapter 7
# Working With PBS Jobs

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

## 7.1 Modifying Job Attributes

Most attributes can be changed by the owner of the job (or a manager or operator) while the job is still queued. However, once a job begins execution, the only resources that can be modified are `cputime` and `walltime`. These can only be reduced.

When the qalter "-l" option is used to alter the resource list of a queued job, it is important to understand the interactions between altering the select directive and job limits.

If the job was submitted with an explicit "-l select=", then vnode-level resources must be qaltered using the "-l select=" form. In this case a vnode level resource RES cannot be qaltered with the "-l RES" form.

For example:

Submit the job:

```
% qsub -l select=1:ncpus=2:mem=512mb jobscript
```
Job’s ID is 230

qalter the job using "-l RES" form:
```
% qalter -l ncpus=4 230
```

Error reported by qalter:
```
qalter: Resource must only appear in "select"
specification when select is used: ncpus 230
```

qalter the job using the "-l select=" form:
```
% qalter -l select=1:ncpus=4:mem=512mb 230
```

No error reported by qalter:
```
%
```

### 7.1.1 Changing the Selection Directive

If the selection directive is altered, the job limits for any consumable resource in the directive are also modified.

For example, if a job is queued with the following resource list:

select=2:ncpus=1:mem=5gb, ncpus=2, mem=10gb

and the selection directive is altered to request

select=3:ncpus=2:mem=6gb

then the job limits are reset to ncpus=6 and mem=18gb

### 7.1.2 Changing the Job-wide Limit

However, if the job-wide limit is modified, the corresponding resources in the selection directive are not modified. It would be impossible to determine where to apply the changes in a compound directive.

Reducing a job-wide limit to a new value less than the sum of the resource in the directive is strongly discouraged. This may produce a situation where the job is aborted during execution for exceeding its limits. The actual effect of such a modification is not specified.

If a job is queued, requested modifications must still fit within the queue's and server's job resource limits. If a requested modification to a resource would exceed the queue's or server's job resource limits, the resource request will be rejected.

Resources are modified by using the `-l` option, either in chunks inside of selection statements, or in job-wide modifications using `resource_name=value` pairs. The selection statement is of the form:

```
-l select=[N:]chunk[+[N:]chunk ...]
```

where N specifies how many of that chunk, and a chunk is of the form:

```
resource_name=value[:resource_name=value ...]
```

Job-wide resource_name=value modifications are of the form:

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

It is an error to use a boolean resource as a job-wide limit.

Placement of jobs on vnodes is changed using the place statement:

```
-l place=modifier[:modifier]
```

where modifier is any combination of group, excl, and/or one of free|pack|scatter.

The usage syntax for `qalter` is:

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

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 20 to 25 minutes, and changing the job name from “airfoil” to “engine”):

```
qstat -u barry
                                                      Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
51.south barry  workq airfoil  930  --   1  -- 0:16 R 0:01
54.south barry  workq airfoil  --   --   1  -- 0:20 Q  --

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

qstat -a 54
                                                      Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq engine   --   --   1  -- 0:25 Q  --
```

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

The qalter command can be used on job arrays, but not on subjobs or ranges of subjobs. When used with job arrays, any job array identifiers must be enclosed in double quotes, e.g.:

```
qalter –l walltime=25:OO “1234[].south”
```

For more information, see the `qalter(1B)` manual page.

## 7.2 Holding and Releasing Jobs

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

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

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

Note that for a job array the `job_identifier` must be enclosed in double quotes.

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

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

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

If the job is running, then the following additional action is taken to interrupt the execution of the job. If checkpoint/restart is supported by the host system, requesting a hold on a running job will cause (1) the job to be checkpointed, (2) the resources assigned to the job to be released, and (3) the job to be placed in the held state in the execution queue. If checkpoint / restart is not supported, `qhold` will only set the requested hold attribute. This will have no effect unless the job is requeued with the `qrerun` command.
The qhold command can be used on job arrays, but not on subjobs or ranges of subjobs. On job arrays, the qhold command can be applied only in the 'Q', 'B' or 'W' states. This will put the job array in the 'H', held, state. If any subjobs are running, they will run to completion. Job arrays cannot be moved in the 'H' state if any subjobs are running.

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

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

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

The usage syntax of the qrls command is:

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

For job arrays, the job_identifier must be enclosed in double quotes.

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

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

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

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

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

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

## 7.3 Deleting Jobs

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

Example:

```
qdel 51
qdel 1234[].server
```

Job array identifiers must be enclosed in double quotes.

Mail is sent for each job deleted unless you specify otherwise. Use the following option to `qdel` to prevent more email than you want from being sent:

```
-W suppress_mail=<N>
```

`N` must be a non-negative integer. Make `N` the largest number of emails you wish to receive. If the number of jobs being deleted exceeds `N`, no mail is sent. Note that a job array is one job, so deleting a job array results in one email being sent.

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

## 7.4 Sending Messages to Jobs

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

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

The usage syntax of the `qmsg` command is:

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

Example:

```
qmsg -O "output file message" 54
qmsg -O "output file message" "1234[].server"
```

Job array identifiers must be enclosed in double quotes.

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

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

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

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

## 7.5 Sending Signals to Jobs

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

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

Job array `job_identifiers` must be enclosed in double quotes.

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

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

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

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

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

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

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

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

## 7.6 Changing Order of Jobs Within Queue

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

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

Usage of the `qorder` command is:

```
qorder job_identifier1 job_identifier2
```

Job array identifiers must be enclosed in double quotes.

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

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

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

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

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

## 7.7 Moving Jobs Between Queues

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

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

The usage syntax of the `qmove` command is:

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

Job array `job_identifiers` must be enclosed in double quotes.

The first operand is the new destination for

*queue*
*@server*
*queue@server*

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

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

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

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

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

## 7.8 Converting a Job into a Reservation Job

The `pbs_rsub` command can be used to convert a normal job into a reservation job that will run as soon as possible. PBS creates a reservation queue and a reservation, and moves the job into the queue. Other jobs can also be moved into that queue via `qmove(1B)` or submitted to that queue via `qsub(1B)`.

The format for converting a normal job into a reservation job is:

```
pbs_rsub [-l walltime=time] -W qmove=job_identifier
```

Example:

```
pbs_rsub -W qmove=54
pbs_rsub -W qmove=”1234[].server”
```

The `-R` and `-E` options to `pbs_rsub` are disabled when using the `-W qmove` option.

For more information, see “Advance Reservation of Resources” on page 137, and the `pbs_rsub(1B)`, `qsub(1B)` and `qmove(1B)` manual pages.

A job’s default walltime is 5 years. Therefore an ASAP reservation’s start time can be in 5 years, if all the jobs in the system have the default walltime.

# Chapter 8
# Advanced PBS Features

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

## 8.1 UNIX Job Exit Status

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

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

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

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

It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the ‘E’ accounting log record of that subjob. See “Job Array Exit Status” on page 169.

## 8.2 Changing UNIX Job umask

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

**Important:** This feature does not apply to Windows.

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

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

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

## 8.3 Requesting qsub Wait for Job Completion

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

If `qsub` receives one of the signals: `SIGHUP`, `SIGINT`, or `SIGTERM`, it will print a message and then exit with the exit status 2. If the job is deleted before running to completion, or an internal PBS error occurs, an error message describing the situation will be printed to this error stream and `qsub` will exit with an exit status of 3. Signals `SIGQUIT` and `SIGKILL` are not trapped and thus will immediately terminate the `qsub` process, leaving the associated job either running or queued. If the job runs to completion, `qsub` will exit with the exit status of the job. (See also section 8.1 “UNIX Job Exit Status” on page 127 for further discussion of the job exit status.)

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

## 8.4 Specifying Job Dependencies

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

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

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

```
type:arg_list[,type:arg_list ...]
```

where except for the `on` type, the `arg_list` is one or more PBS job IDs in the form:

```
jobid[:jobid ...]
```

There are several types:

`after:arg_list`

This job may be scheduled for execution at any point after all jobs in arg_list have started execution.

`afterok:arg_list`

This job may be scheduled for execution only after all jobs in arg_list have terminated with no errors. See "Warning about exit status with csh" in EXIT STATUS.

`afternotok:arg_list`

This job may be scheduled for execution only after all jobs in arg_list have terminated with errors. See "Warning about exit status with csh" in EXIT STATUS.

`afterany:arg_list`

This job may be scheduled for execution after all jobs in arg_list have terminated, with or without errors.

`before:arg_list`

Jobs in arg_list may begin execution once this job has begun execution.

`beforeok:arg_list`

Jobs in arg_list may begin execution once this job terminates without errors. See "Warning about exit status with csh" in EXIT STATUS.

`beforenotok:arg_list`

If this job terminates execution with errors, the jobs in arg_list may begin. See "Warning about exit status with csh" in EXIT STATUS.

`beforeany:arg_list`

Jobs in arg_list may begin execution once this job terminates execution, with or without errors.

`on:count`

This job may be scheduled for execution after `count` dependencies on other jobs have been satisfied. This type is used in conjunction with one of the `before` types listed. `count` is an integer greater than 0.

Job IDs in the arg_list of `before` types must have been submitted with a type of `on`.

To use the `before` types, the user must have the authority to alter the jobs in `arg_list`. Otherwise, the dependency is rejected and the new job aborted.

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.

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 specify these job dependencies.

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

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

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

Similarly, you can use before dependencies, as the following example exhibits. Note that unlike after dependencies, before dependencies require the use of the on dependency.

```
qsub -W depend=on:2 job1
16397.jupiter
qsub -W depend=beforeany:16397 job2
16398.jupiter
qsub -W depend=beforeany:16397 job3
16399.jupiter
```

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

## 8.5 Delivery of Output Files

To transfer output files or to transfer staged-in or staged-out files to/from a remote destination, PBS uses either `rcp` or `scp` depending on the configuration options. The version of `rcp` used by PBS always exits with a non-zero exit status for any error. Thus MOM knows if the file was delivered or not. The secure copy program, `scp`, is also based on this version of `rcp` and exits with the proper status code.

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

1. The user lacks authorization to access the specified system. (See discussion in “User’s PBS Environment” on page 22.)
2. Under UNIX, if the user’s `.cshrc` outputs any characters to standard output, e.g. contains an echo command, `pbs_rcp` will fail.

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

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

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

## 8.6 Input/Output File Staging

File staging is a way to specify which files should be copied onto the execution host before the job starts, and which should be copied off the execution host when it completes. (For file staging under Globus, see “PBS File Staging through GASS” on page 148.) The “`-W stagein=file_list`” and “`-W stageout=file_list`” options to `qsub` specify which files are staged (copied) in before the job starts or staged out after the job completes execution. On completion of the job, all staged-in and staged-out files are removed from the execution system. The *file_list* is in the form:

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

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

```
local_file ◄——   remote_host:remote_file
```

and for stage out, the direction of travel is:

```
local_file ——►   remote_host:remote_file
```

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

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

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

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

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

Example of a stagein:

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

Example of a stageout :

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

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

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

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

When stageout encounters an error, there are three retries. PBS waits 1 second and tries again, then waits 11 seconds and tries a third time, then finally waits another 21 seconds and tries a fourth time.

File staging is supported for job arrays.  See “File Staging” on page 158.

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

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

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

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

### 8.6.1 File Staging Between UNIX and Windows

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

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

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

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

### 8.6.2 Stage-in Failure

When stage-in fails, the job is placed in a 30-minute wait to allow the user time to fix the problem. Typically this is a missing file or a network outage. Email is sent to the job owner when the problem is detected. Once the problem has been resolved, the job owner or the Operator may remove the wait by resetting the time after which the job is eligible to be run via the `-a` option to `qalter`. The server will update the job’s comment with information about why the job was put in the wait state. When the job is eligible to run, it may run on different vnodes.

## 8.7 The `pbsdsh` Command

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

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

Usage of the `pbsdsh` command is:

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

Note that the double dash must come after the options and before the program and arguments. The double dash is only required for Linux.

The available options are:

| | |
|---|---|
| `-c N` | The program is spawned on the first *N* vnodes allocated. If the value of *N* is greater than the number of vnodes, it will “wrap” around, running multiple copies on the vnodes. This option is mutually exclusive with `-n`. |
| `-n N` | The program is spawned on a single vnode which is the *N*-th vnode allocated. This option is mutually exclusive with `-c`. |
| `-o` | The program will not wait for the tasks to finish. |
| `-s` | If this option is given, the program is run sequentially on each vnode, one after the other. |
| `-v` | Verbose output about error messages and task exit status is produced. |

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

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

```
#!/bin/sh
#PBS -l select=4:ncpus=1
```

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

pbsdsh ./myapp app-arg1
```

The `pbsdsh` command runs one task for each line in the PBS_NODEFILE. Each MPI rank will get a single line in the PBS_NODEFILE, so if you are running multiple MPI ranks on the same host, you will still get multiple `pbsdsh` tasks on that host.

## 8.8 Advance Reservation of Resources

An *Advance Reservation* is a set of resources with availability limited to a specific user (or group of users), a specific start time, and a specified duration. The user submits an advance reservation with the `pbs_rsub` command. PBS will then confirm that the reservation can be met, or else reject the request. Once the scheduler has confirmed the reservation, the queue that was created to support this reservation will be enabled, allowing jobs to be submitted to it. The queue will have a user level access control list set to the user who submitted the reservation and any other users the owner specified. The queue will accept jobs in the same manner as normal queues. When the reservation start time is reached, the queue will be started. Once the reservation is complete, any jobs remaining in the queue (running or not) will be deleted, and the reservation removed from the Server.

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

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

### 8.8.1 Submitting a PBS Reservation

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

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

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

| | |
|---|---|
| -R datetime | Specifies reservation starting time. If the reservation's end time and duration are the only times specified, this start time is calculated. The datetime argument adhers to the POSIX time specification: `[[[[CC]YY]MM]DD]hhmm[.SS]` If the day, `DD`, is not specified, it will default to today if the time `hhmm` is in the future. Otherwise, the day will be set to tomorrow. For example, if you submit a reservation having a specification `-R 1110` at 11:15am, it will be interpreted as being for 11:10am tomorrow. If the month portion, `MM`, is not specified, it defaults to the current month provided that the specified day `DD`, is in the future. Otherwise, the month will be set to next month. Similar comments apply to the two other optional, left hand components. |
| -E datetime | Specifies the reservation end time. See the `-R` flag for a description of the datetime string. If start time and duration are the only times specified, the end time value is calculated. |
| -D timestring | Specifies reservation duration. Timestring can either be expressed in seconds of walltime or it can be expressed as a colon delimited timestring e.g. HH:MM:SS or MM:SS. If the start time and end time are the only times specified, this duration time is calculated. |
| -q destination | Specifies the destination server to which to submit the reserva- |

tion. The default server is used if this option is not specified.

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

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

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

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

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

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

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

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

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

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

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

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

The following attribute is supported:

`qmove=jobid`

Converts the normal job with job ID jobid into a reservation job that will run as soon as possible. Creates the reservation and reservation queue and places the job in the queue. Uses the resources requested by the job to create the reservation.

In creating the reservation, resources requested through the `pbs_rsub` command override existing job resources. Therefore, if the existing job resources are greater than those requested for the reservation, the job will be rejected by the reservation.

The `-R` and `-E` options to `pbs_rsub` are disabled when using the `qmove=jobid` attribute.

See “Converting a Job into a Reservation Job” on page 125.

The following example shows the submission of a reservation asking for 1 vnode, 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 -R 1130 -D 30:00:00
R226.south UNCONFIRMED
```

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

### 8.8.2 Waiting for Confirmation

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

### 8.8.3 Showing Reservation Status

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

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

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

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

```
pbs_rstat -f R226
Name: R226.south
Reserve_Owner = user1@south
reserve_type = 2
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Fri Aug 24 11:30:00 2004
reserve_end = Fri Aug 24 12:00:00 2004
reserve_duration = 1800
queue = R226
Resource_List.ncpus = 1
Resource_List.mem = 500kb
Resource_List.nodes = 1
Resource_List.walltime = 00:30:00
Authorized_Users = user1@south
server = south
ctime = Fri Aug 24 06:30:53 2004
mtime = Fri Aug 24 06:30:53 2004
Variable_List = PBS_O_LOGNAME=user1,PBS_O_HOST=south
euser = user1
egroup = group1
```

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

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

### 8.8.4 Delete PBS Reservations

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

```
pbs_rdel R304
```

### 8.8.5 Accounting

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

### 8.8.6 Access Control

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

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

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

## 8.9 Dedicated Time

*Dedicated time* is one or more specific time periods defined by the administrator. These are not repeating time periods. Each one is individually defined.

During dedicated time, the only jobs PBS starts are those in special dedicated time queues. PBS schedules non-dedicated jobs so that they will not run over into dedicated time. Jobs in dedicated time queues are also scheduled so that they will not run over into non-dedicated time. PBS will attempt to backfill around the dedicated-non-dedicated time borders.

PBS uses walltime to schedule within and around dedicated time. If a job is submitted without a walltime to a non-dedicated-time queue, it will not be started until all dedicated time periods are over. If a job is submitted to a dedicated-time queue without a walltime, it will never run.

To submit a job to be run during dedicated time, use the -q <queue name> option to qsub and give the name of the dedicated-time queue you wish to use as the queue name. Queues are created by the administrator; see your administrator for queue name(s).

## 8.10 Using Comprehensive System Accounting

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

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

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

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

On the command line:

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

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

If the user creates a file foo with these commands:

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

The user could run this script via qsub:

```
qsub foo
```

This does the same thing, via the script “foo”.

## 8.11 Globus Support

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

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

### 8.11.1 Running Globus jobs

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

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

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

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

### 8.11.2 PBS and Globusrun

If you’re familiar with the `globusrun` utility, the following mappings of options from

PBS to an RSL string may be of use to you:

**Table 13: qsub Option vs. Globus RSL**

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

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

**Table 14: PBS Job States vs. Globus States**

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

### 8.11.3 PBS File Staging through GASS

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

*`localfile@host:inputfile`*

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

The same process is used for a stageout directive:

*`localfile@host:outputfile`*

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

### 8.11.4 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.

### 8.11.5 Examples

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

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

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

Upon execution, this will give the sample output:

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

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

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

mpirun -n=4 ~/mpitest

wait
```

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

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

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

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

Upon execution, the sample output is:

```
Done it
```

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

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

## 8.12 Running PBS in a UNIX DCE Environment

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

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

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

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

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

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

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

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

## 8.13 Running PBS in a UNIX Kerberos Environment

PBS Professional includes optional support for Kerberos-only (i.e. no DCE) environment. (By optional, we mean that the customer may acquire a copy of PBS Professional with the standard security and authentication module replaced with the KRB5 module.) This is not supported under Windows.

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

# Chapter 9
# Job Arrays

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

## 9.1 Definitions

| | |
|---|---|
| **Subjob** | Individual entity within a job array (e.g. **1234[7]**, where **1234[]** is the job array itself, and **7** is the index) which has many properties of a job as well as additional semantics (defined below.) |
| **Sequence_number** | The numeric part of a job or job array identifier, e.g. **1234.** |
| **Subjob index** | The unique index which differentiates one subjob from another. This must be a non-negative integer. |
| **Job array identifier** | The identifier returned upon success when submitting a job array. The format is **sequence_number[]** or **sequence_number[].server.domain.com.** |

| | |
|---|---|
| **Job array range** | A set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices. |

### 9.1.1 Description

A job array is a compact representation of one or more jobs, called subjobs when part of a Job array, which have the same job script, and have the same values for all attributes and resources, with the following exceptions:

- each subjob has a unique index
- Job Identifiers of subjobs only differ by their indices
- the state of subjobs can differ

All subjobs within a job array have the same scheduling priority.

A job array is submitted through a single command which returns, on success, a "job array identifier" with a server-unique sequence number. Subjob indices are specified at submission time. These can be:

- a contiguous range, e.g. 1 through 100
- a range with a stepping factor, e.g. every second entry in 1 through 100 (1, 3, 5, ... 99)

A job array identifier can be used

- by itself to represent the set of all subjobs of the job array
- with a single index (a "job array identifier") to represent a single subjob
- with a range (a "job array range") to represent the subjobs designated by the range

### 9.1.2 Identifier Syntax

Job arrays have three identifier syntaxes:

- The job array object itself : 1234[].server or 1234[]
- A single subjob of a job array with index M: 1234[M].server or 1234[M]
- A range of subjobs of a job array: 1234[X-Y:Z].server or 1234[X-Y:Z]

**Examples**:

1234[].server.domain.com  Full job array identifier

| | |
|---|---|
| 1234[] | Short job array identifier |
| 1234[73] | Subjob identifier of the 73rd index of job array 1234[] |
| 1234 | Error, if 1234[] is a job array |
| 1234.server.domain.com | Error, if 1234[].server.domain.com is a job array |

The sequence number (1234 in 1234[].server) is unique, so that jobs and job arrays cannot share a sequence number.

**Note**: Since some shells, for example csh and tcsh, read "[" and "]" as shell metacharacters, job array names and subjob names will need to be enclosed in double quotes for all PBS commands.

**Example**:

```
qdel "1234.myhost[5]"
qdel "1234.myhost[]"
```

Single quotes will work, except where you are using shell variable substitution.

## 9.2 qsub: Submitting a Job Array

To submit a job array, qsub is used with the option **-J range**, where **range** is of the form **X-Y[:Z]**. **X** is the starting index, **Y** is the ending index, and **Z** is the optional **stepping factor**. X and Y must be whole numbers, and Z must be a positive integer. Y must be greater than X. If Y is not a multiple of the stepping factor above X, (i.e. it won't be used as an index value) the highest index used will be the next below Y. For example, 1-100:2 gives 1, 3, 5, ... 99.

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

Interactive submission of job arrays is not allowed.

**Examples:**

To submit a job array of 10,000 subjobs, with indices 1, 2, 3, ... 10000:

```
$ qsub -J 1-10000 job.scr
1234[].server.domain.com
```

To submit a job array of 500 subjobs, with indices 500, 501, 502, ... 1000:

```
$ qsub -J 500-1000 job.scr
1235[].server.domain.com
```

To submit a job array with indices 1, 3, 5 ... 999:

```
$ qsub -J 1-1000:2 job.scr
1236[].server.domain.com
```

### 9.2.1 Interactive Job Submission

Job arrays do not support interactive submission.

## 9.3 Job Array Attributes

Job arrays and subjobs have all of the attributes of a job. In addition, they have the following when appropriate. These attributes are read-only.

**Table 15: Job Array Attributes**

| Name | Type | Applies To | Value |
|---|---|---|---|
| array | boolean | job array | True if item is job array |
| array_id | string | subjob | Subjob's job array identifier |
| array_index | string | subjob | Subjob's index number |
| array_state_count | string | job array | Similar to state_count attribute for server and queue objects. Lists number of subjobs in each state. |
| array_indices_remaining | string | job array | List of indices of subjobs still queued. Range or list of ranges, e.g. 500, 552, 596-1000 |

**Table 15: Job Array Attributes**

| **Name** | **Type** | **Applies To** | **Value** |
|---|---|---|---|
| array_indices_submitted | string | job array | Complete list of indices of subjobs given at submission time. Given as range, e.g. 1-100 |

## 9.4 Job Array States

Job array states map closely to job states except for the ‘B’ state. The ‘B’ state applies to job arrays and indicates that at least one subjob has left the queued state and is running or has run, but not all subjobs have run. Job arrays will never be in the ‘R’, ‘S’ or ‘U’ states.

**Table 16: Job Array States**

| State | Indication |
|---|---|
| B | The job array has started |
| W | The job array has a wait time in the future |
| H | The job array is held |
| T | The job array is in transit between servers |
| Q | The  job array is queued, or has been qrerun |
| E | All subjobs are finished and the server is cleaning up the job array |

### 9.4.1 Subjob States

Subjobs can be in one of six states, listed here.

**Table 17: Subjob States**

| State | Indication |
|---|---|
| Q | Queued |
| R | Running |
| E | Ending |
| X | Expired or deleted; subjob has completed execution or been deleted |
| S | Suspended |
| U | Suspended by keyboard activity |

## 9.5 PBS Environmental Variables

**Table 18: PBS Environmental Variables**

| Environment Variable Name | Used For | Description |
|---|---|---|
| $PBS_ARRAY_INDEX | subjobs | Subjob index in job array, e.g. 7 |
| $PBS_ARRAY_ID | subjobs | Identifier for a job array. Sequence number of job array, e.g. 1234 |
| $PBS_JOBID | Jobs, subjobs | Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 1234[7] |

## 9.6 File Staging

File staging for job arrays is like that for jobs, with an added variable to specify the subjob index. This variable is **^array_index^**. This is the name of the variable that will be used for the actual array index. The stdout and stderr files follow the naming convention for jobs, but include the identifier of the job array, which includes the subscripted index. As with jobs, the stagein and stageout keywords require the -W option to qsub.

### 9.6.1 Job Array File Staging Syntax on UNIX

stagein = local_path@host:remote_path

stageout = local_path@host:remote_path

"Local_path" is the path on the execution host.
Local_path is either relative to the user's home directory, or an absolute path.
"Host" is the machine where the data normally resides.
"Remote_path" is the path on the machine where the data normally resides.

**Examples**:

Remote_path: /film
Data files used as input: frame1, frame2, frame3
Host: store
Local_path: /tmp
Executable: a.out

For this example, `a.out` produces `frame2.out` from `frame2.`

```
#PBS -W stagein=/tmp/in/frame^array_index^@store:/film/frame^array_index^
#PBS- W stageout=/tmp/out/frame^array_index^.out \
                @store:/film/frame^array_index^.out
#PBS -J 1-3
a.out frame$PBS_ARRAY_INDEX  /tmp/in /tmp/out
```

Note that the stageout statement is all one line, broken here for readability.

The result will be that the user's directory named "film" contains the original files frame1, frame2, frame3, plus the new files frame1.out, frame2.out and frame3.out.

#### 9.6.1.1 Scripts

**Example 1**
In this example, we have a script named ArrayScript which calls scriptlet1 and scriptlet2. All three scripts are located in `/homedir/testdir`.

```
#!/bin/sh
#PBS -N ArrayExample
```

```
#PBS -J 1-2
echo "Main script: index " $PBS_ARRAY_INDEX
/homedir/testdir/scriptlet$PBS_ARRAY_INDEX
```

In our example, scriptlet1 and scriptlet2 simply echo their names. We run ArrayScript using the qsub command:

```
qsub ArrayScript
```

**Example 2**

In this example, we have a script called StageScript. It takes two input files, `dataX` and `extraX`, and makes an output file, `newdataX`, as well as echoing which iteration it is on. The `dataX` and `extraX` files will be staged from `inputs` to `work`, then `new-dataX` will be staged from `work` to `outputs`.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein=/homedir/work/data^array_index^
   @host1:/homedir/inputs/data^array_index^
#PBS -W stagein=/homedir/work/extra^array_index^
   @host1:/homedir/inputs/extra^array_index^
#PBS -W stageout=/homedir/work/newdata^array_index^
   @host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cd /homedir/work
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX
   >> newdata$PBS_ARRAY_INDEX
```

Local path (execution directory): `/homedir/work`
Remote path for inputs (original data files `dataX` and `extraX`): `/homedir/inputs`
Remote path for results (output of computation `newdataX`): `/homedir/outputs`
Host (data storage host): host1

`StageScript` resides in `/homedir/testdir`. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in `/homedir`, our home directory, which is why the line "`cd /homedir/work`" is in the script.

#### 9.6.1.2 Output Filenames

The name of the job array will default to the script name if no name is given via qsub -N. For example, if the sequence number were 1234,

```
    #PBS -N fixgamma
```

would give stdout for index number 7 the name fixgamma.o1234.7 and stderr the name fixgamma.e1234.7.
The name of the job array can also be given through stdin.

### 9.6.2 Job Array Staging Syntax on Windows

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

Example of a stagein:

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

Example of a stageout:

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

## 9.7 PBS Commands

### 9.7.1 PBS Commands Taking Job Arrays as Arguments

**Note**: Some shells such as csh and tcsh use the square bracket (“[“, “]”) as a metacharacter. When using one of these shells, and a PBS command taking subjobs, job arrays or job array ranges as arguments, the subjob, job array or job array range must be enclosed in double quotes.

The following table shows PBS commands that take job arrays, subjobs or ranges as arguments. The cells in the table indicate which objects are acted upon. In the table,

| | |
|---|---|
| Array[] = | the job array object |
| Array[Range] = | the set of subjobs of the job array with indices in range given |
| Array[Index] = | the individual subjob of the job array with the index given |

| | |
|---|---|
| Array[RUNNING] = | the set of subjobs of the job array which are currently running |
| Array[QUEUED] = | the set of subjobs of the job array which are currently queued |
| Array[REMAINING] = | the set of subjobs of the job array which are queued or running |
| Array[DONE]= | the set of subjobs of the job array which have finished running |

**Table 19: PBS Commands Taking Job Arrays as Arguments**

| Command | Argument to Command | | |
|---|---|---|---|
| | Array[] | Array[Range] | Array[Index] |
| qstat | Array[] | Array[Range] | Array[Index] |
| qdel | Array[] & Array[REMAINING] | Array[Range] where Array[REMAINING] | Array[Index] |
| qalter | Array[] | erroneous | erroneous |
| qorder | Array[] | erroneous | erroneous |
| qmove | Array[] & Array[QUEUED] | erroneous | erroneous |
| qhold | Array[] & Array[QUEUED] | erroneous | erroneous |
| qrls | Array[] & Array[QUEUED] | erroneous | erroneous |
| qrerun | Array[RUNNING] & Array[DONE] | Array[Range] where Array[RUNNING] | Array[Index] |
| qrun | erroneous | Array[Range] where Array[QUEUED] | Array[Index] |
| tracejob | erroneous | erroneous | Array[Index] |
| qsig | Array[RUNNING] | Array[Range] where Array[RUNNING] | Array[Index] |
| qmsg | erroneous | erroneous | erroneous |

### 9.7.2 qstat: Status of a Job Array

The qstat command is used to query the status of a Job Array. The default output is to list the Job Array in a single line, showing the Job Array Identifier. Options can be combined. To show the state of all running subjobs, use -t -r. To show the state only of subjobs, not job arrays, use -t -J.

**Table 20: Job Array and Subjob Options to qstat**

| Option | Result |
|---|---|
| -t | Shows state of job array object and subjobs.<br>Will also show state of jobs. |
| -J | Shows state only of job arrays. |
| -p | Prints the default display, with column for Percentage Completed.<br>For a job array, this is the number of subjobs completed or deleted divided by the total number of subjobs. For a job, it is time used divided by time requested. |

**Examples**:
We run an example job and an example job array, on a machine with 2 processors:
demoscript:

```
#!/bin/sh
#PBS -N JobExample
sleep 60
```

arrayscript:

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-5
sleep 60
```

We run these scripts using qsub.

```
qsub arrayscript
1235[].host
qsub demoscript
1236.host
```

Then:

**qstat**

```
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
1236.host   JobExample   user1              0 Q workq
```

**qstat -J**

```
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
```

**qstat -p**

```
Job id      Name         User        % done  S Queue
----------- ------------ ----------  ------- - -----
1235[].host ArrayExample user1            0  B workq
1236.host   JobExample   user1           --  Q workq
```

**qstat -t**

```
Job id       Name         User        Time Use S Queue
-----------  ------------ ----------  -------- - -----
1235[].host  ArrayExample user1              0 B workq
1235[1].host ArrayExample user1       00:00:00 R workq
1235[2].host ArrayExample user1       00:00:00 R workq
1235[3].host ArrayExample user1              0 Q workq
1235[4].host ArrayExample user1              0 Q workq
1235[5].host ArrayExample user1              0 Q workq
1236.host    JobExample   user1              0 Q workq
```

**qstat -Jt**

```
Job id       Name         User   Time Use S Queue
------------ ------------ -----  -------- - -----
1235[1].host ArrayExample user1  00:00:00 R workq
1235[2].host ArrayExample user1  00:00:00 R workq
1235[3].host ArrayExample user1         0 Q workq
1235[4].host ArrayExample user1         0 Q workq
1235[5].host ArrayExample user1         0 Q workq
```

After the first two subjobs finish:

```
     qstat -Jtp
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq

     qstat -pt
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[].host  ArrayExample user1      40 B workq
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
1236.host    JobExample   user1      -- Q workq
```

Now if we wait until only the last subjob is still running:

```
      qstat -rt
host:
                                                     Req'd  Req'd   Elap
Job ID     Username Queue Jobname    SessID NDS TSK Memory Time  S Time
----------- ------  ----- ---------  ------ --- --- ------ ----- - -----
1235[5].host user1  workq ArrayExamp 3048    --   1    --    --  R 00:00
1236.host    user1  workq JobExample 3042    --   1    --    --  R 00:00

      qstat -Jrt
host:
                                                     Req'd  Req'd   Elap
Job ID      Username Queue Jobname   SessID NDS TSK Memory Time  S Time
----------- -------- ----- --------- ------ --- --- ------ ----- - -----
1235[5].host   user1 workq ArrayExamp  048   --   1    --    --  R 00:01
```

### 9.7.3 qdel: Deleting a Job Array

The `qdel` command will take a job array identifier, subjob identifier or job array range. The indicated object(s) are deleted, including any currently running subjobs. Running subjobs are treated like running jobs. Subjobs not running will be deleted and never run. Only one email is sent per deleted job array, so deleting a job array of 5000 subjobs results in one email being sent.

### 9.7.4 qalter: Altering a Job Array

The qalter command can only be used on a job array object, not on subjobs or ranges. Job array attributes are the same as for jobs.

### 9.7.5 qorder: Ordering Job Arrays in the Queue

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

### 9.7.6 qmove: Moving a Job Array

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

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

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

### 9.7.7 qhold: Holding a Job Array

The qhold command can only be used with job array objects, not with subjobs or ranges. A hold can be applied to a job array only from the ‘Q’, ‘B’ or ‘W’ states. This will put the job array in the ‘H’, held, state. If any subjobs are running, they will run to completion. No queued subjobs will be started while in the ‘H’ state.

### 9.7.8 qrls: Releasing a Job Array

The qrls command can only be used with job array objects, not with subjobs or ranges. If the job array was in the 'Q' or 'B' state, it will be returned to that state. If it was in the 'W' state, it will be returned to that state unless its waiting time was reached, it will go to the 'Q' state.

### 9.7.9 qrerun: Requeueing a Job Array

The qrerun command will take a job array identifier, subjob identifier or job array range. If a job array identifier is given as an argument, it is returned to its initial state at submission time, or to its altered state if it has been qaltered. All of that job array's subjobs are requeued, which includes those that are currently running, and completed and deleted. If a subjob or range is given, those subjobs are requeued as jobs would be.

### 9.7.10 qrun: Running a Job Array

The qrun command takes a subjob or a range of subjobs, not a job array object. If a single subjob is given as the argument, it is run as a job would be. If a range of subjobs is given as the argument, the non-running subjobs within that range will be run.

### 9.7.11 tracejob on Job Arrays

The tracejob command can be run on job arrays and individual subjobs. When tracejob is run on a job array or a subjob, the same information is displayed as for a job, with additional information for a job array. Note that subjobs do not exist until they are running, so tracejob will not show any information until they are. When tracejob is run on a job array, the information displayed is only that for the job array object, not the subjobs. Job arrays themselves do not produce any MOM log information. Running tracejob on a job array will give information about why a subjob did not start.

### 9.7.12 qsig: Signaling a Job Array

If a job array object, subjob or job array range is given to qsig, all currently running subjobs within the specified set will be sent the signal.

### 9.7.13 qmsg: Sending Messages

The qmsg command is not supported by job arrays.

## 9.8 Other PBS Commands Supported for Job Arrays

### 9.8.1 qselect: Selection of Job Arrays

The default behavior of qselect is to return the job array identifier, without returning sub-job identifiers.

Note: qselect will not return any job arrays when the state selection (-s) option restricts the set to ‘R’, ‘S’, ‘T’ or ‘U’, because a job array will never be in any of these states. However, qselect can be used to return a list of subjobs by using the -t option.

Options to qselect can be combined. For example, to restrict the selection to subjobs, use both the -J and the -T options. To select only running subjobs, use -J -T -sR.

**Table 21: Options to qselect for Job Arrays**

| Option | Selects | Result |
|---|---|---|
| (none) | jobs,<br>job arrays | Shows job and job array identifiers |
| -J | job arrays | Shows only job array identifiers |
| -T | jobs,<br>subjobs | Shows job and subjob identifiers |

## 9.9 Job Arrays and xpbs

xpbs does not support job arrays.

## 9.10 More on Job Arrays

### 9.10.1 Job Array Run Limits

Jobs and subjobs are treated the same way by job run limits. For example, if *max_user_run* is set to 5, a user can have a maximum of 5 subjobs and/or jobs running.

### 9.10.2 Starving

A job array's starving status is based on the queued portion of the array. This means that if there is a queued subjob which is starving, the job array is starving. A running subjob retains its starving status when it was started.

### 9.10.3 Job Array Dependencies

Job dependencies are supported:

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

Note: Job dependencies are not supported for subjobs or ranges of subjobs.

### 9.10.4 Accounting

Job accounting records for job arrays and subjobs are the same as for jobs. When a job array has been moved from one server to another, the subjob accounting records are split between the two servers, except that there will be no 'Q' records for subjobs.

### 9.10.5 Checkpointing

Checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed, instead they run to completion.

### 9.10.6 Prologues and Epilogues

If defined, prologues and epilogues will run at the beginning and end of each subjob, but not for job arrays.

### 9.10.7 Job Array Exit Status

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob.

| Exit Status | Meaning |
|---|---|
| 0 | All subjobs of the job array returned an exit status of 0. No PBS error occurred. Deleted subjobs are not considered |
| 1 | At least 1 subjob returned a non-zero exit status. No PBS error occurred. |
| 2 | A PBS error occurred. |

### 9.10.8 Scheduling Job Arrays

All subjobs within a job array have the same scheduling priority.

#### 9.10.8.1 Preemption

Individual subjobs may be preempted by higher priority work.

#### 9.10.8.2 Peer Scheduling

Peer scheduling does not support job arrays.

#### 9.10.8.3 Fairshare

Subjobs are treated like jobs with respect to fairshare ordering, fairshare accounting and fairshare limits. If running enough subjobs of a job array causes the priority of the owning entity to change, additional subjobs from that job array may not be the next to start.

#### 9.10.8.4 Placement sets and Node Grouping

All nodes associated with a single subjob should belong to the same placement set or node group. Different subjobs can be put on different placement sets or node groups.

# Chapter 10
# Multiprocessor Jobs

PBS has two new resources, mpiprocs and ompthreads. See section 4.2.2 “Built-in Resources” on page 35.

## 10.1 Submitting SMP Jobs

To submit a job which should run on one host and which requires a certain number of cpus and amount of memory, submit the job with:

```
qsub -l select=ncpus=N:mem=M -l place=group=host
```

When the job is run, the PBS_NODEFILE will contain one entry, the name of the selected execution host. Generally this is ignored for SMP jobs as all processes in the job are run on the host where the job script is run. The job will have two environment variables, NCPUS and OMP_NUM_THREADS, set to N, the number of CPUs allocated.

## 10.2 Submitting MPI Jobs

If you have a cluster of small systems with for example 2 CPUs each, and you wish to submit an MPI job that will run on four separate hosts, then submit:

```
qsub -l select=4:ncpus=1 -l place=scatter
```

The PBS_NODEFILE file will contain one entry for each of the hosts allocated to the job. In the example above, it would contain 4 lines. The variables NCPUS and OMP_NUM_THREADS will be set to one.

If you do not care where the four MPI processes are run, you may submit:

```
qsub -l select=4:ncpus=1 -l place=free
```

and the job will run on 2, 3, or 4 hosts depending on what is available.

For this example, PBS_NODEFILE will contain 4 entries, either four separate hosts, or 3 hosts one of which is repeated once, or 2 hosts, etc. NCPUS and OMP_NUM_THREADS will be set 1 or 2 depending on the number of cpus allocated from the first listed host.

#### 10.2.0.1 The mpiprocs Resource

The number of MPI processes for a job is controlled by the value of the resource mpiprocs. The mpiprocs resource controls the contents of the PBS_NODEFILE on the host which executes the top PBS task for the PBS job (the one executing the PBS job script.) See “Built-in Resources” on page 35. The PBS_NODEFILE contains one line per MPI process with the name of the host on which that process should execute. The number of lines in PBS_NODEFILE is equal to the sum of the values of mpiprocs over all chunks requested by the job. For each chunk with mpiprocs=P, (where P > 0), the host name (the value of the allocated vnode's resources_available.host) is written to the PBS_NODEFILE exactly P times.

If a user wishes to run two MPI processes on each of 3 hosts and have them "share" a single processor on each host, the user would request

```
-lselect=3:ncpus=1:mpiproces=2
```

## 10.3 Submitting Multi-threaded Jobs

The number of OpenMP threads for a job is controlled by the value of the resource ompthreads. The ompthreads resource controls the values of the NCPUS and OMP_NUM_THREADS environment variables for every PBS task (including the top PBS task).

If a chunk requests ncpus=N, with N > 1, PBS will only create one MPI process for that chunk, but set the number of OpenMP threads to N.

So to request 16 MPI processes each with 2 threads on machines with 2 processors:

```
qsub -l select=16:ncpus=2
```

To request two chunks, each with 8 CPUs and 8 MPI tasks and four threads:

```
qsub -l select=2:ncpus=8:mpiprocs=8:ompthreads=4
```

Example: if "-l select=3:ncpus=2+1:ncpus=1" is satisfied by 4 CPUs from VnodeA, 2 from VnodeB and 1 from VnodeC, the following would be written to the PBS_NODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeC
```

The OpenMP environment variables would be set (for the 4 PBS tasks corresponding to the 4 MPI processes) as follows:

```
For PBS task #1 on VnodeA:  OMP_NUM_THREADS=2  NCPUS=2
For PBS task #2 on VnodeA:  OMP_NUM_THREADS=2  NCPUS=2
For PBS task #3 on VnodeB:  OMP_NUM_THREADS=2  NCPUS=2
For PBS task #4 on VnodeC:  OMP_NUM_THREADS=1  NCPUS=1
```

Example: if "-l select=3:ncpus=2:mpiprocs=2:ompthreads=1" is satisfied by 2 CPUs from each of three vnodes (VnodeA, VnodeB, and VnodeC), the following would be written to the PBS_VNODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

The OpenMP environment variables would be set (for the 6 PBS tasks corresponding to the 6 MPI processes) as follows:

```
For PBS task #1 on VnodeA:  OMP_NUM_THREADS=1  NCPUS=1
For PBS task #2 on VnodeA:  OMP_NUM_THREADS=1  NCPUS=1
For PBS task #3 on VnodeB:  OMP_NUM_THREADS=1  NCPUS=1
For PBS task #4 on VnodeB:  OMP_NUM_THREADS=1  NCPUS=1
```

For PBS task #5 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1
For PBS task #6 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1

If the user is running an OpenMP application on a host and wishes to run more threads than the number of CPUs requested (because each thread is I/O bound perhaps), the user would request

```
-l select=1:ncpus=8:ompthreads=16
```

Most multi-threaded jobs will request all CPUs on the same host. Should you have a job that is both MPI and multi-threaded, you will need to be careful in determining the number of CPUs and vnodes allocated. The total number of entries in the PBS_NODEFILE will be the same as the total number of CPUs allocated. NCPUS and OMP_NUM_THREADS will be set to the number of CPUs allocated from the first host. In order to run only one MPI process on each host, it is necessary to process the PBS_NODEFILE to eliminate duplicate entries.

## 10.4 MPI Jobs with PBS

For most implementations of the Message Passing Interface (MPI), you would use the `mpirun` command to launch your application. For example, here is a sample PBS script for an MPI job:

```
#PBS -l select=arch=linux
#
mpirun -np 32 -machinefile $PBS_NODEFILE a.out
```

### 10.4.1 MPICH Jobs With PBS

For users of PBS with MPICH on Linux, the `mpirun` command has been changed slightly. The syntax and arguments are the same except for one option, which should not be set by the user:

| | |
|---|---|
| -machinefile file | PBS supplies the machinefile. If the user tries to specify it, PBS will print a warning that it is replacing the machinefile. |

```
#PBS -l select=arch=linux
#
mpirun a.out
```

Under Windows the `-localroot` option to `MPICH`'s `mpirun` command may be needed in order to allow the job's processes to run more efficiently.

### 10.4.2 MPI Jobs Using LAM MPI

The `pbs_mpilam` command follows the convention of LAM's `mpirun`. The "nodes" here are LAM nodes. LAM's `mpirun` has two syntax forms:

```
pbs_mpilam/mpirun [global_options] [<where>] <program>
[--args]

pbs_mpilam/mpirun [global_options] <schema file>
```

Where
`<where>` is a set of node and/or CPU identifiers indicating where to start <`program`>:

Nodes: `n<list>`, e.g., `n0-3,5`
CPUS: `c<list>`, e.g., `c0-3,5`
Extras: `h` (local node), `o` (origin node), `N` (all nodes), `C` (all CPUs)
`<schema file>` is an ASCII file containing a description of the programs which constitute an application.

The first form is fully supported by PBS: all user MPI processes are tracked. The second form is supported, but user MPI processes are not tracked.

CAUTION: Keep in mind that if the `<where>` argument and global option `-np` or `-c` are not specified in the command line, then `pbs_mpilam` will expect an ASCII schema file as argument.

### 10.4.3 MPI Jobs Using AIX, POE

For PBS users of AIX machines running IBM's Parallel Operating Environment, or POE, the `poe` command has been changed slightly. The syntax and arguments are the same except for the following:

| | |
|---|---|
| -procs option: | If this is not set, the number of task processes is set by PBS. If the user sets this value to a number higher than that set by PBS, PBS prints a warning that it is assigning multiple processes per processor. |
| -hostfile option | PBS supplies the hostfile to POE. This should not be supplied by the user, and if the user specifies it, PBS will print a warning that it is replacing the hostfile. |

Notes:

1Debugging using pdbx will work in the expected fashion.
2Your system administrator may have set PBS up so that applications which are directly invoked will be part of a PBS job.
3Since PBS is tracking tasks started by poe, these tasks are counted towards a user's run limits.
4Usernames must be identical across hosts.

For more information on using IBM's Parallel Operating Environment, see
"IBM Parallel Environment for AIX 5L Hitchhiker's Guide"

### 10.4.4 PBS MPI Jobs on HP-UX and Linux

PBS is more tightly integrated with the `mpirun` command on HP-UX so that resources can be tracked and processes managed. When running a PBS MPI job, you can use the same arguments to the `mpirun` command as you would outside of PBS. The `-h host` and `-l user` options will be ignored, and the `-np number` option will be modified to fit the available resources.

### 10.4.5 PBS Jobs with `MPICH-GM's mpirun` Using `rsh/ssh (mpirun.ch_gm)`

PBS provides an interface to `MPICH-GM's mpirun` using `rsh/ssh`. If executed inside a PBS job, this lets PBS track all `MPICH-GM` processes started via `rsh/ssh` so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_gm` had been used.

You use the same command as you would use outside of PBS, either "mpirun.ch_gm" or "mpirun".

### 10.4.5.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.ch_gm` except for the following:

| | |
|---|---|
| `-machinefile <file>` | The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`. |
| `-np` | If not specified, the number of entries found in the `$PBS_NODEFILE` is used. |
| `-pg` | The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS. |

### 10.4.5.2 Examples

Run a single-executable `MPICH-GM` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
PBS_NODEFILE:
pbs-host1
pbs-host2
pbs-host3

qsub -l select=3:ncpus=1
mpirun.ch_gm -np 64 /path/myprog.x 1200
^D
<job-id>
```

Run an `MPICH-GM` job with multiple executables on multiple hosts listed in the process group file "`procgrp`":

```
qsub -l select=3:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp

mpirun.ch_gm -pg procgrp /path/mypro.x
```

```
rm -f procgrp
^D
<job-id>
```

When the job runs, `mpirun.ch_gm` will give this warning message:

warning: “-pg” is allowed but it is up to user to make sure only PBS hosts are specified; MPI processes spawned are not guaranteed to be under the control of PBS.

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under PBS-control.

### 10.4.6 PBS Jobs with `MPICH-MX's mpirun` Using `rsh/ssh (mpirun.ch_mx)`

PBS provides an interface to `MPICH-MX’s mpirun` using rsh/ssh. If executed inside a PBS job, this allows for PBS to track all `MPICH-MX` processes started by rsh/ssh so that PBS can perform accounting and has complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_mx` had been used.

You use the same command as you would use outside of PBS, either “mpirun.ch_mx” or “mpirun”.

#### 10.4.6.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as mpirun.ch_mx except for the following:

| | |
|---|---|
| `-machinefile <file>` | The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`. |
| `-np` | If not specified, the number of entries found in the `$PBS_NODEFILE` is used. |
| `-pg` | The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS. |

### 10.4.6.2 Examples

Run a single-executable MPICH-MX job with 64 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

```
PBS_NODEFILE
pbs-host1
pbs-host2
pbs-host3

qsub -l select=3:ncpus=1
mpirun.ch_mx -np 64 /path/myprog.x 1200
^D
<job-id>
```

Run an MPICH-MX job with multiple executables on multiple hosts listed in the process group file "procgrp":

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" \
> procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2" \
>> procgrp
mpirun.ch_mx -pg procgrp /path/myprog.x
rm -f procgrp
^D
<job-id>
```

mpirun.ch_mx will give the warning message:

warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI processes spawned are not guaranteed to be under PBS-control

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

## 10.4.7 PBS Jobs with MPICH-GM's mpirun Using MPD (mpirun.mpd)

PBS provides an interface to MPICH-GM's mpirun using MPD. If executed inside a

PBS job, this allows for PBS to track all `MPICH-GM` processes started by the MPD daemons so that PBS can perform accounting have and complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun.mpd` with MPD had been used.

You use the same command as you would use outside of PBS, either “mpirun.mpd” or “mpirun”. If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

### 10.4.7.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.mpd` with MPD except for the following:

| | |
|---|---|
| `-m <file>` | The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`. |
| `-np` | If not specified, the number of entries found in the `$PBS_NODEFILE` is used. |
| `-pg` | The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS. |

### 10.4.7.2 MPD Startup and Shutdown

The script starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method based on the value of the environment variable `RSH-COMMAND`. The default is `rsh`. The script also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start GM's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may have been started already by the administrator or by the user. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun.mpd` that the user executed (GM or MX), which would determine the path to the MPD binary.

### 10.4.7.3 Examples

Run a single-executable `MPICH-GM` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
PBS_NODEFILE:
pbs-host1
pbs-host2
pbs-host3

qsub -l select=3:ncpus=1
[MPICH-GM-HOME]/bin/mpirun.mpd -np 64 \
/path/myprog.x 1200
^D
<job-id>
```

If the GM MPD daemons are not running, the PBS interface to `mpirun.mpd` will start them as this user on the allocated PBS hosts. The daemons may have been previously started by the administrator or the user.

Run an `MPICH-GM` job with multiple executables on multiple hosts listed in the process group file "procgrp":

Job script:

```
qsub -l select=3:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" \
   > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" \
   >> procgrp

[MPICH-GM-HOME]/bin/mpirun.mpd -pg procgrp \
   /path/mypro.x 1200
rm -f procgrp
^D
<job-id>
```

When the job runs, `mpirun.mpd` will give the warning message:

warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI processes spawned are not guaranteed to be under PBS-control.

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

### 10.4.8 PBS Jobs with `MPICH-MX's mpirun` Using MPD (`mpirun.mpd`)

PBS provides an interface to `MPICH-MX's mpirun` using MPD. If executed inside a PBS job, this allows for PBS to track all `MPICH-MX` processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun.ch_mx with MPD was used.

You use the same command as you would use outside of PBS, either "mpirun.mpd" or "mpirun". If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

#### 10.4.8.1 Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun.ch_gm` with MPD except for the following:

| | |
|---|---|
| `-m <file>` | The `file` argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`. |
| `-np` | If not specified, the number of entries found in the `$PBS_NODEFILE` is used. |
| `-pg` | The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to user to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS. |

#### 10.4.8.2 MPD Startup and Shutdown

The PBS `mpirun` interface starts MPD daemons on each of the unique hosts listed in `$PBS_NODEFILE`, using either the `rsh` or `ssh` method, based on value of environment variable `RSHCOMMAND`. The default is `rsh`. The interface also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start MX's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may already have been started by the administrator or by the user. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun.mpd` that the user executed

(GM or MX), which would determine the path to the MPD binary.

### 10.4.8.3 Examples

Run a single-executable `MPICH-MX` job with 64 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
PBS_NODEFILE:
pbs-host1
pbs-host2
pbs-host3

qsub -l select=3:ncpus=1
[MPICH-MX-HOME]/bin/mpirun.mpd -np 64 \
/path/myprog.x 1200
^D
<job-id>
```

If the MPD daemons are not running, the PBS interface to `mpirun.mpd` will start GM's MPD daemons as this user on the allocated PBS hosts. The MPD daemons may be already started by the administrator or by the user.

Run an `MPICH-MX` job with multiple executables on multiple hosts listed in the process group file “procgrp”:

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" \
> procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2"\
>> procgrp

[MPICH-MX-HOME]/bin/mpirun.mpd -pg procgrp \
/path/myprog.x 1200
rm -f procgrp
^D
<job-id>
```

`mpirun.mpd` will print a warning message:

warning: “-pg” is allowed but it is up to user to make sure only PBS hosts

are specified; MPI processes spawned are not guaranteed to be under
PBS-control

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

### 10.4.9 PBS Jobs with `MPICH2's mpirun`

PBS provides an interface to `MPICH2's mpirun`. If executed inside a PBS job, this allows for PBS to track all `MPICH2` processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `MPICH2's mpirun` had been used.

You use the same “mpirun” command as you would use outside of PBS.

When submitting PBS jobs that invoke the 8.0 pbsrun wrapper script for MPICH2's mpirun, be sure to explicitly specify the actual number of ranks or MPI tasks in the qsub select specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, specification of the following in 7.1:

```
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

would result in a 7.1 $PBS_NODEFILE listing:

```
hostA
hostB
hostB
```

but in 8.0 it would be:

```
hostA
hostB
```

which would conflict with the "-np 3" specification in mpirun as only 2 MPD daemons will be started.

The correct way in 8.0 is to specify either a) or b) as follows:

a) #PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB

b) #PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2

which would cause $PBS_NODEFILE to list:

hostA
hostB
hostB

and an "mpirun -np 3" would then be consistent.

### 10.4.9.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as `MPICH2's mpirun` except for the following:

| | |
|---|---|
| `-host, -ghost` | For specifying the execution host to run on. Ignored. |
| `-machinefile <file>` | The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE.` |
| `-localonly <x>` | For specifying the `<x>` number of processes to run locally. Not supported. The user is advised instead to use the equivalent arguments: **`"-np <x> -localonly".`** |
| `-np` | If the user does not specify a `-np` option, then no default value is provided by the PBS wrapper scripts. It is up to the local mpirun to decide what the reasonable default value should be, which is usually 1. |

### 10.4.9.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in the `$PBS_NODEFILE`. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

#### 10.4.9.3 Examples

Run a single-executable `MPICH2` job with 6 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
PBS_NODEFILE:
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
# mpirun runs 6 processes mapped to each host listed
# in $PBS_NODEFILE
mpirun -np 6 /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script

<job-id>
```

Run an `MPICH2` job with multiple executables on multiple hosts using `$PBS_NODEFILE` and `mpiexec` arguments in `mpirun`:

```
PBS_NODEFILE:
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2
mpirun -np 2 /tmp/mpitest1 : \
       -np 2 /tmp/mpitest2 : \
       -np 2 /tmp/mpitest3
```

Run job:

```
qsub job.script
```

Run an `MPICH2` job with multiple executables on multiple hosts using `mpirun -configfile` option and `$PBS_NODEFILE`:

```
PBS_NODEFILE:
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2
echo "-np 2 /tmp/mpitest1" > my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file
mpirun -configfile my_config_file
rm -f my_config_file
```

Run job:

```
qsub job.script
```

### 10.4.10 PBS Jobs with Intel MPI's `mpirun`

PBS provides an interface to Intel MPI’s `mpirun`. If executed inside a PBS job, this allows for PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard Intel MPI's `mpirun` was used.

You use the same “mpirun” command as you would use outside of PBS.

When submitting PBS jobs that invoke the 8.0 pbsrun wrapper script for Intel MPI, be sure to explicitly specify the actual number of ranks or MPI tasks in the qsub select specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, specification of the following in 7.1:

```
    #PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
    mpirun -np 3 /tmp/mytask
```

would result in a 7.1 $PBS_NODEFILE listing:

```
hostA
hostB
hostB
```

but in 8.0 it would be:

```
hostA
hostB
```

which would conflict with the "-np 3" specification in mpirun as only 2 MPD daemons will be started.

The correct way in 8.0 is to specify either a) or b) as follows:

a) `#PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB`

b) `#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2`

which would cause $PBS_NODEFILE to list:

```
hostA
hostB
hostB
```

and an "mpirun -np 3" would then be consistent.

### 10.4.10.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as for Intel MPI's `mpirun` except for the following:

| | |
|---|---|
| `-host, -ghost` | For specifying the execution host to run on. Ignored. |

`-machinefile <file>` The file argument contents are ignored and replaced by the contents of the `$PBS_NODEFILE`.

`mpdboot` option `--totalnum=*`
Ignored and replaced by the number of unique entries in `$PBS_NODEFILE`.

`mpdboot` option `--file=*`
Ignored and replaced by the name of `$PBS_NODEFILE.` The argument to this option is replaced by `$PBS_NODEFILE`.

Argument to `mpdboot` option `-f <mpd_hosts_file>`
Replaced by `$PBS_NODEFILE`.

-s If the PBS interface to Intel MPI’s `mpirun` is called inside a PBS job, Intel MPI’s `mpirun -s` argument to `mpdboot` is not supported as this closely matches the `mpirun` option "`-s <spec>`". The user can simply run a separate `mpdboot -s` before calling `mpirun`. A warning message is issued by the PBS interface upon encountering a `-s` option telling users of the supported form.

-np If the user does not specify a `-np` option, then no default value is provided by the PBS interface. It is up to the local `mpirun` to decide what the reasonable default value should be, which is usually 1.

### 10.4.10.2 MPD Startup and Shutdown

Intel MPI's `mpirun` takes care of starting/stopping the MPD daemons. The PBS interface to Intel MPI’s `mpirun` always passes the arguments `-totalnum=<number of mpds to start>` and `-file=<mpd_hosts_file>` to the actual `mpirun`, taking its input from unique entries in `$PBS_NODEFILE`.

### 10.4.10.3 Examples

Run a single-executable Intel MPI job with 6 processes spread out across the PBS-allocated hosts listed in `$PBS_NODEFILE`:

```
PBS_NODEFILE:
```

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
# mpirun takes care of starting the MPD daemons
# on unique hosts listed in $PBS_NODEFILE,
# and also runs 6 processes mapped to each host
# listed in $PBS_NODEFILE; mpirun takes care of
# shutting down MPDs.
mpirun /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

Run an Intel MPI job with multiple executables on multiple hosts using `$PBS_NODEFILE` and `mpiexec` arguments to `mpirun`:

```
$PBS_NODEFILE
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
# mpirun runs MPD daemons on hosts listed
# in $PBS_NODEFILE
# mpirun runs 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on
# hostC.
# mpirun takes care of shutting down the
# MPDs at the end of MPI job run.
mpirun -np 2 /tmp/mpitest1 : \
       -np 2 /tmp/mpitest2 : \
       -np 2 /tmp/mpitest3
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
 <job-id>
```

Run an Intel MPI job with multiple executables on multiple hosts via the `-configfile` option and `$PBS_NODEFILE`:

```
$PBS_NODEFILE:
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
echo "-np 2 /tmp/mpitest1" >> my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file

# mpirun takes care of starting the MPD daemons
# config file says run 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on
# hostC.
# mpirun takes care of shutting down the MPD
# daemons.
mpirun -configfile my_config_file

# cleanup
rm -f my_config_file
```

Run job script:

```
qsub -l select=3:ncpus=2 job.script
<job-id>
```

## 10.5 MPI Jobs on the Altix

### 10.5.1 Jobs on an Altix Running ProPack 4/5

PBS has its own mpiexec for the Altix running ProPack 4 or greater. The PBS mpiexec has the standard mpiexec interface. The PBS mpiexec does require proper configuration of the Altix. See your administrator to find out whether your system is configured for the PBS mpiexec.

You can launch an MPI job on a single Altix, or across multiple Altixes. PBS will manage and track the processes. You can use CSA, if it is configured, to collect accounting information on your jobs. PBS will run the MPI tasks in the cpusets it manages.

You can run MPI jobs in the placement sets chosen by PBS. When a job is finished, PBS will clean up after it.

For MPI jobs across multiple Altixes, PBS will manage the multihost jobs. For example, if you have two Altixes named Alt1 and Alt2, and want to run two applications called mympi1 and mympi2 on them, you can put this in your job script:

```
mpiexec -host Alt1 -n 4 mympi1 : -host Alt2 -n 8 mympi2
```

You can specify the name of the array to use via the PBS_MPI_SGIARRAY environment variable.

To verify how many CPUs are included in a cpuset created by PBS, use:

```
> $ cpuset -d <set name> | egrep cpus
```

This will work either from within a job or not.

The alt_id returned by MOM has the form `cpuset=<name>`. <name> is the name of the cpuset, which is the $PBS_JOBID.

Jobs will share cpusets if the jobs request sharing and the cpusets' sharing attribute is not set to force_excl. Jobs can share the memory on a nodeboard if they have a CPU from that nodeboard. To fit as many small jobs as possible onto vnodes that already have shared jobs on them, request sharing in the job resource requests.

PBS will try to put a job that will fit in a single nodeboard on just one nodeboard. However, if the only CPUs available are on separate nodeboards, and those vnodes are not allocated exclusively to existing jobs, and the job can share a vnode, then the job will be run on the separate nodeboards.

### 10.5.2 Jobs on an Altix Running ProPack 2/3

You can launch MPI jobs on a single Altix running ProPack 2/3. Submit MPI jobs using SGI's `mpirun`.

## 10.6 Jobs on the IBM Blue Gene

### 10.6.1 The Blue Gene System

A Blue Gene system is made up of one service node, one or more front-end nodes, a shared storage location (referred to a the CWFS -- cluster wide file system), dozens or hundreds of I/O nodes, thousands of compute nodes, and various networks that keep everything together. The front-end node, service node, and I/O node run the Linux SUSE Enterprise 9 OS; the compute node runs a lightweight OS called OSK.

A compute node (cnode) is made up of 2 cpus. Each processor supports only 1 thread of execution per processor; no dynamic creation of processes is allowed. Jobs run on the compute nodes in one of 2 modes: 1) co-processor mode, where only 1 CPU is used for computation while the other is used for communication, and the job must fit in 512 MB of memory; 2) virtual mode, where both cpus can be used for computation where the cpus interact using a shared, non-cached area of memory (scratchpad). Each process must fit in 256 MB of memory.

Each compute node is connected to six other nearest neighbors in a torus manner (there's a wraparound edge linked to similar edge processors). Every compute node is associated with at least 1 I/O node, which performs all I/O duties. A set of compute nodes and 1 I/O node make up what is termed a "PSET".

A 3d rectangle of compute nodes of size 8x8x8 = 512 sub-cubes is called a midplane or base partition (BP). 1 rack is made up of 2 base partitions. Each base partition can be quartered to make up 4x4x8 = 128 compute nodes, and in each quarter, 4 more subdivisions can occur resulting in 4x4x2 = 32 compute nodes (cnodes) making up what is termed as a node board or node card. The smallest Blue Gene machine is 1 rack (2 base partitions) totaling 1024 cnodes, with the largest being 64 racks (128 base partitions) for a total of 65536 c-nodes.

Each base partition is connected to 3 switches - one for each dimension in (x, y, z). Some switches can be used as "pass through" switches in mesh configurations. Each network switch has 6 ports where 2 ports (P0, P1) connect to a midplane, while the other ports

(P2…P5) connect to other switches.

A Blue Gene job contains an executable, its arguments, and owner (one who submitted the job). It runs exclusively on a 3d, rectangular, contiguous, isolated set of compute nodes called a partition or bglblock. Valid partition sizes are as follows:

32 c-nodes (1/16 BP)
128 c-nodes (1/4 BP)
512 c-nodes (1 BP)
one or more BPs

### 10.6.2 Setup Requirements on Blue Gene

The system administrator must completely partition the system in advance, to accommodate users' requirements. PBS will take care of finding these previously-defined partitions and scheduling jobs on these partitions.

### 10.6.3 Example Blue Gene Hierarchy

This hierarchy is used for job submission examples later. It looks like this:

```
R_32 = 4096 cnodes (4 racks, full system bglblock)
        R0 = 2048 cnodes (2 racks)
        R00 = 1024 cnodes (1 rack)
             R000 = 512 cnodes
             R001 = 512 cnodes
        R01 = 1024 cnodes (1 rack)
             R010 = 512 cnodes
             R011 = 512 cnodes

        R1 = 2048 cnodes (2 racks)
        R10 = 1024 cnodes (1 rack)
             R100 = 512 cnodes
                   R1000 = 128 cnodes
                   R1001 = 128 cnodes
                   R1002 = 128 cnodes
                   R1003 = 128 cnodes
             R101 = 512 cnodes
        R11 = 1024 cnodes (1 rack)
             R110 = 512 cnodes
             R111 = 512 cnodes
```

### 10.6.4 Support for Blue Gene

All of the functionality of PBS on Linux is the same except preemption and floating PBS licenses, which are not supported. The suspend/resume feature of PBS jobs is not supported. Attempts to suspend a PBS job will return "No support for requested service".

The hold/release feature of PBS either through check_abort, or restart action scripts, foregrounded or transmogrified, is supported.

### 10.6.5 Using `mpirun` on Blue Gene

The user invokes mpirun in the job script to actually run the executable on some assigned partition, as specified in environment variable MPIRUN_PARTITION:

```
#PBS -l select=128:cnodes=1
mpirun <executable> <args>
```

The user specifies the compute node execution mode, which can be either “co-processor”, or “virtual node”, via mpirun inside the job script:

```
#PBS -l select=1024:cnodes=1
mpirun -mode CO -np 1024  <executable> <args>
```

This runs 1024 tasks on 1024 compute nodes using 1 CPU/node.

Or

```
mpirun -mode VN -np 2048 <executable> <args>
```

This runs 2048 tasks on 1024 compute nodes using 2 CPUs/node.

The user specifies

-np (number of tasks) or

-mapfile (assigning unique cnode coordinates for each task)

in the mpirun invocation. In this case, the tasks will run under the compute nodes of the partition that PBS assigned to the job:

```
qsub -l select=512:cnodes=1
mpirun -np 300 -mapfile coord_file <executable> <args>
```

Based on the example hierarchy, the job is assigned partition R111, but only 300 tasks are spawned in a 512 compute nodes pool.

To bind a partition to the job, call:

```
mpirun -partition $MPIRUN_PARTITION
```

where `MPIRUN_PARTITION` is the environment variable instantiated by `pbs_mom` to refer to the assigned partition.

Compute nodes can be under-allocated but not over-allocated. `mpirun` will reject a request if the number of tasks specified is greater than the available cpus on the assigned partition.

IBM's mpirun takes care of instantiating a user's executable on the assigned partition.

The following specified options do not work with predefined partitions:

“-partition”
“-connect” (connection type of base partitions)
“-shape” (for specifying job size in XxYxZ c-nodes format)
“-psets_per_bg” (number of I/O nodes)
“-host” (service node host)
“-noallocate”
“-nofree”

Example:

```
qsub -l select=cnodes=256
mpirun -partition R100 my_exec
WARNING: ignoring option -partition

mpirun -shape 23x10x2 my_exec
WARNING: ignoring option -shape

mpirun -connect TORUS my_exec
WARNING: ignoring option -connect

mpirun -psets_per_bg 8 my_exec
WARNING: ignoring option -psets_per_bg
```

#### 10.6.5.1 The `mpirun` Supplied by PBS

PBS supplies an `mpirun` that calls the Blue Gene `mpirun`, translating some arguments. The user’s script arguments to `mpirun` are translated as follows:

**Table 22: Translation of mpirun Arguments in Script**

| User's mpirun specification | mpirun Translation: #PBS -l select=X:cnodes=1 | mpirun Translation: #PBS -l select=Y:ncpus=1 |
|---|---|---|
| No -mode<br>No -np | -mode CO<br>-np X | -mode CO<br>-np Y/2 |
| No -mode<br>-np Z | -mode CO<br>-np Z | -mode VN<br>-np Z |
| -mode CO<br>no -np | -mode CO<br>-np X | -mode CO<br>-np Y/2 |
| -mode VN<br>no -np | -mode VN<br>-np X*2 | -mode VN<br>-np Y |
| -mode CO<br>-np Z | -mode CO<br>-np Z | -mode CO<br>-np Z |
| -mode VN<br>-np Z | -mode VN<br>-np Z | -mode VN<br>-np Z |

Details of translation:

If the user specified an mpirun -mode value, then that is what mpirun will use as node mode value.

Example:
```
qsub -l select=512:cnodes=1
mpirun -mode VN my_exec  -->
mpirun -mode VN -np 512 my_exec
```

If the user specified an mpirun -np value, then that is what mpirun will use as the number of mpirun tasks to spawn.

Example:
```
qsub -l select=1024:cnodes=1
mpirun -np 768 my_exec  -->
mpirun -mode CO -np 768 my_exec
```

If the user did not specify a -mode value, mpirun puts in a default of -mode CO for co-processor mode.

Example:

```
qsub -l select=1024:cnodes=1
mpirun -np 1024 my_exec  -->
mpirun -mode CO -np 1024 my_exec
```

An exception to this is if the user requested "ncpus" in the qsub line (e.g. qsub -l select=1:ncpus=2) along with the mpirun -np specification on the job script. In this case a default mode of VN for virtual node will be put in by the wrapped mpirun.

Example:

```
qsub -l select=256:ncpus=1
mpirun -np 160 my_exec   translates to -->
mpirun -mode VN 160 my_exec
```

If `mpirun` resolves putting in a CO (co-processor) node mode value, and the user did not specify a -np value, then the value for number of tasks is either

1. the number of compute nodes (X) if the PBS job was submitted as **`qsub -l select=X:cnodes=1`**, or
2. "number of cpus divided by 2" (Y/2) if the PBS job was submitted as **`qsub -l select=Y:ncpus=1`**.

Examples:

```
qsub -l select=129:cnodes=1
mpirun my_exec  -->
mpirun -mode CO -np 129 my_exec

qsub -l select=1024:ncpus=1
mpirun my_exec  -->
mpirun -mode CO -np 512 my_exec

qsub -l select=129:ncpus=1
mpirun my_exec  -->
mpirun -mode CO -np 64 my_exec
```

where (129/2) = 64 rounded down

If `mpirun` resolves putting in a VN (virtual node) node mode value, and the user did not specify a -np value, then the value for number of tasks is either

1. "number of compute nodes times 2" (X*2) if the user submitted the PBS job as **`qsub -l select=X:cnodes=1`**, or
2. number of cpus if the user submitted the PBS job as **`qsub -l select=Y:ncpus=1`**.

Example:

```
qsub -l select=129:cnodes=1
mpirun -mode VN my_exec  -->
mpirun -mode VN -np 258 my_exec

qsub -l select=1024:ncpus=1
mpirun -mode VN my_exec  -->
mpirun -mode VN -np 1024 my_exec
```

If the user does not specify a -np value, number of cnodes requested, or number of cpus requested, `mpirun` adds:

```
-np $MPIRUN_PARTITION_SIZE
```

where `MPIRUN_PARTITION_SIZE` is the environment variable instantiated by pbs_mom to refer to the assigned partition's size in number of compute nodes.

The PBS `mpirun` takes care of attaching a "</dev/null" to the actual `mpirun` command line to empty out standard input. This is needed in order for `mpirun` to run properly inside a batch job since `mpirun` tries to read from STDIN causing failures in a batch environment where there is no tty.

The MOM expects a job attribute called “pset” that is set by the scheduler to the name of the partition chosen to run the job. This attribute is visible to the user. The format is

pset = partition=<host_short_name>-<partition_identifier>

For instance, if the scheduler chose partition R001 on a machine called BG1, then a running job would have the following set:

```
pset = partition=BG1-R001
```

The $PBS_NODEFILE contains the set of vnodes forming a legal partition that has been assigned to the job (e.g. R001, R010).

### 10.6.6 Running Jobs on Blue Gene

Since only user-level checkpointing is available for processes running on the BG/L nodes, the user is responsible for performing periodic checkpoints on their applications. The application must be compiled against the Blue Gene checkpoint library, so that the application can do its own checkpointing.

All pre-defined partitions (containing midplanes) will uniformly have either “torus” or “mesh” as their connection type. Therefore, users do not need to specify a connection type when submitting jobs.

In the job's executing environment, the following environment variables are always set:

```
MPIRUN_PARTITION=<partition_name>
MPIRUN_PARTITION_SIZE=<# of cnodes>
```

where `MPIRUN_PARTITION` is the partition assigned to the PBS job, and `MPIRUN_PARTITION_SIZE` is the number of compute nodes (cnodes) making up the assigned partition.

On a hold request of a running job, the Blue Gene job associated with the PBS job would be canceled.

On a release request, the job is restarted with MPIRUN_PARTITION and MPIRUN_PARTITION_SIZE variables restored in its environment, pointing to an assigned partition.

The Blue Gene administrator must configure each partition to mount the shared file system, otherwise `mpirun` calls will fail with a “login failed:” message.

#### 10.6.6.1 Blue Gene MPI Job Input/Output/Error File Handling

Running MPI jobs in Blue Gene depends on the shared location in the cluster wide filesystem (CWFS) that has been set up for a site. This shared location is what is mounted on the partition as it boots up, and is accessible by the Blue Gene I/O nodes for creation, duplication of input/output/error files. It is recommended that users create their MPI programs in such a way that input is read, and output/error files are created under this shared location.

Users submitting jobs on the front end host no longer need to expect PBS to return output/error files back to this submission host. That is, `pbs_mom` on the service node should not be copying files back to the front-end node.

Users are encouraged to submit jobs with the “-koe” option, which is to keep output/error files on the service node.

Sample MPI program and PBS job script:

```
Mpitest.c:
#include <stdio.h>
#include <mpi.h>
#include <errno.h>

int main(int argc, char *argv[]) {
   int rank, size;
   char localhost[100];
   int     len;
   int     sleepsec;
   FILE    *ofp = NULL;

   MPI_Init(&argc, (char ***)&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   gethostname(localhost, sizeof(localhost));

   if( argv[1] != NULL ) {
      sleepsec = atoi(argv[1]);
      printf("%s: Sleeping for %d seconds\n",  localhost,
             sleepsec);
      sleep(sleepsec);
   }

   if( argv[2] != NULL ) {
        printf("about  to open %s\n", argv[2]);
        ofp = fopen(argv[2], "w");
   }

   if( ofp == NULL ) {
       printf("ofp is stdout since fopen failed errno=%d\n",
        errno);
        ofp = stdout;
   } else {
        printf("ofp is NOT stdout\n");
   }

   fprintf(ofp, "%s: Hello, world! I am %d of %d\n",
        localhost, rank+1, size);
   MPI_Finalize();

   if( ofp != stdout )
        fclose(ofp);

   return 0;
}
```

Job script:

```
#PBS -l select=cnodes=512
#PBS -N midplanejob
#PBS -koe
#PBS -q midplane
#PBS -V

mpirun -cwd /FS/pbs_test/ /FS/pbs_test/mpitest 20 \
mpitest.out
```

where /FS is the shared filesystem.

### 10.6.7 Job Submission Examples

Example 1: Specify 512 nodes; the job will run on one of the 512-node partitions first if available (R000, R001, R010, R011, R100, R101, R110, R111}:

```
qsub -l select=512:cnodes=1
qsub -l select=1024:ncpus=1
```
(512 * 2 cpus)

Example 2: Request 1024 nodes; the job will run on a 1024-node partition first if available: {R00, R01, R10, R11}:

```
qsub -l select=1024:cnodes=1
qsub -l select=2048:ncpus=1
```
(1024 * 2 cpus)

Example 4: Request 2048 nodes; the job will run on one of the 2048-node partitions first if available: {R0, R1}

```
qsub -l select=2048:cnodes=1
qsub -l select=4096:ncpus=1       (2048 * 2 cpus)
```

Example 5: Request the whole system; the job will run on the R_32 partition:

```
qsub -l select=4096:cnodes=1
qsub -l select=8192:ncpus=1      (4096 * 2 cpus)
```

Example 6: Request 32 compute nodes; the job will still end up in one of the 128-node small partitions:

```
qsub -l select=32:cnodes=1
qsub -l select=64:ncpus=1      (32 * 2 cpus)
```

Example 7: Request 129 compute nodes; the job will still end up in one of the 512-node partitions:

```
qsub -l select=129:cnodes=1
qsub -l select=258:ncpus=1     (129 * 2 cpus)
```

Example 8: Request 600 compute nodes; the job will end up in the 1024 node partition:

```
qsub -l select=600:cnodes=1
qsub -l select=1200:ncpus=1         ( 600 * 2 cpus)
```

Example 9: Request 1500 compute nodes; the job will end up in one of the 2048-node partitions:

```
qsub -l select=1500:cnodes=1
qsub -l select=3000:ncpus=1       (1500 * 2 cpus)
```

Example 10: Request 2049 compute nodes; the job will have to run on the full system partition:

```
qsub -l select=2049:cnodes=1
qsub -l select=4098:ncpus=1       (2049 * 2 CPUS)
```

Example 11: The following request is unsatisfiable since it exceeds the number of compute nodes in a full system partition:

```
qsub -l select=5000:cnodes=1
qsub -l select=10000:ncpus=1
```

Example 12: Complex requests (spanning multiple select chunks), though unnecessary, can be satisfied.

```
qsub -l select=128:cnodes=1+512:cnodes=1
```

causes the scheduler to find a partition that can accommodate at least 640 compute nodes.

## 10.7 PVM Jobs with PBS

On a typical system, to execute a Parallel Virtual Machine (PVM) program you would use the `pvmexec` command. For example, here is a sample PBS script for a PVM job:

```
#PBS -l nodes=32
#
pvmexec a.out -inputfile data_in
```

## 10.8 OpenMP Jobs with PBS

To provide support for OpenMP jobs, the environment variable `OMP_NUM_THREADS` is created for the job with the value of the number of CPUs allocated to the job. The variable `NCPUS` is also set to this value.

## 10.9 Checkpointing SGI MPI Jobs

### 10.9.1 Jobs on an Altix

Jobs are suspended or checkpointed on the Altix using the PBS suspend and checkpoint. There is no OS-level checkpoint. Suspended or checkpointed jobs will resume on the original nodeboards.

### 10.9.2 Jobs on IRIX

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) cannot 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.

PBS does not support cpusets with multi-host jobs on the IRIX.

On IRIX, the cpuset name is the first 8 characters of the job ID. If there is already a cpuset by that name, the last character in the name is replaced by a,b,c...z,A,...,Z until a unique name is found.

## 10.10 Jobs on the NEC SX-8

PBS supports the following NEC features:

The NEC checkpoint facility provides the PBS job checkpointing feature.

The NEC job feature creates a NEC jobid for each PBS task. This jobid acts as an inescapable session on a single host. PBS can track MPI processes as long as they are all on one NEC machine.

PBS supports the NEC SX-8, except for the following:

Users cannot run interactive jobs.

No support for running the client commands: xpbs, xpbsmon, pbs_tclsh, or pbs_wish, directly on the SX-8. They can be used from other platforms to connect to an SX-8 system, just not directly run on the SX-8 itself.

Cycle harvesting based on load average and keyboard/mouse activity is not supported.

There is no vmem resource (NEC SX-8 machines do not use virtual memory.)

The pbs_probe command will work the same except for the following:

No files or directories related to Tcl/Tk will exist.

Permissions for PBS_EXEC and PBS_HOME will have the group write bit set.

# Appendix A: PBS Environment Variables

**Table 23: PBS Environment Variables**

| Variable | Meaning |
|---|---|
| `NCPUS` | Number of threads, defaulting to number of CPUs, on the vnode |
| `OMP_NUM_THREADS` | Same as NCPUS. |
| `PBS_ARRAY_ID` | Identifier for job arrays. Consists of sequence number. |
| `PBS_ARRAY_INDEX` | Index number of subjob in job array. |
| `PBS_ENVIRONMENT` | Indicates job type: **`PBS_BATCH`** or **`PBS_INTERACTIVE`** |
| `PBS_JOBCOOKIE` | Unique identifier for inter-MOM job-based communication. |
| `PBS_JOBID` | The job identifier assigned to the job or job array by the batch system. |
| `PBS_JOBNAME` | The job name supplied by the user. |
| `PBS_MOMPORT` | Port number on which this job's MOMs will communicate. |
| `PBS_NODEFILE` | The filename containing a list of vnodes assigned to the job. |
| `PBS_NODENUM` | Logical vnode number of this vnode allocated to the job. |
| `PBS_O_HOME` | Value of **`HOME`** from submission environment. |
| `PBS_O_HOST` | The host name on which the `qsub` command was executed. |
| `PBS_O_LANG` | Value of **`LANG`** from submission environment |
| `PBS_O_LOGNAME` | Value of **`LOGNAME`** from submission environment |
| `PBS_O_MAIL` | Value of **`MAIL`** from submission environment |
| `PBS_O_PATH` | Value of **`PATH`** from submission environment |

**Table 23: PBS Environment Variables**

| Variable | Meaning |
|---|---|
| `PBS_O_QUEUE` | The original queue name to which the job was submitted. |
| `PBS_O_SHELL` | Value of **`SHELL`** from submission environment |
| `PBS_O_SYSTEM` | The operating system name where `qsub` was executed. |
| `PBS_O_TZ` | Value of **`TZ`** from submission environment |
| `PBS_O_WORKDIR` | The absolute path of directory where `qsub` was executed. |
| `PBS_QUEUE` | The name of the queue from which the job is executed. |
| `PBS_TASKNUM` | The task (process) number for the job on this vnode. |
| `TMPDIR` | The job-specific temporary directory for this job. |

# Appendix B: Converting From NQS to PBS

For those converting to PBS from NQS or NQE, PBS includes a utility called **`nqs2pbs`** which converts an existing NQS job script so that it will work with PBS. (In fact, the resulting script will be valid to both NQS and PBS.) The existing script is copied and PBS directives ("`#PBS`") are inserted prior to each NQS directive (either "`#QSUB`" or "`#Q$`") in the original script.

```
nqs2pbs existing-NQS-script new-PBS-script
```

**Important:** Converting NQS date specifications to the PBS form may result in a warning message and an incomplete converted date. PBS does not support date specifications of "today", "tomorrow", or the name of the days of the week such as "Monday". If any of these are encountered in a script, the PBS specification will contain only the time portion of the NQS specification (i.e. `#PBS -a hhmm[.ss]`). It is suggested that you specify the execution time on the `qsub` command line rather than in the script. All times are taken as local time. If any unrecognizable NQS directives are encountered, an error message is displayed. The new PBS script will be deleted if any errors occur.

Section "Setting Up Your UNIX/Linux Environment" on page 23 discusses PBS environment variables.

A queue complex in NQS was a grouping of queues within a batch Server. The purpose of a complex was to provide additional control over resource usage. The advanced scheduling features of PBS eliminate the requirement for queue complexes.

# Index