CLUSTER RESOURCES INC.

TORQUE

Administrator's Guide

version 2.3

Copyright

Trademarks

Table of Contents

TORQUE Administrator's Guide 1

Preface 1

TORQUE Administrator Guide Overview 1

What is a Resource Manager? 2

What are Batch Systems? 2

Basic Job Flow 3

Glossary 4

1.0 Overview 4

1.1.1 TORQUE Architecture 4

1.1.2 Installing TORQUE 4

1.1.3 Compute Nodes 5

1.1.4 Enabling TORQUE as a service (optional) 7

1.2.1 Initialize/Configure TORQUE on the Server (pbs_server) 7

1.2.2 Specify Compute Nodes 8

1.2.3 Configure TORQUE on the Compute Nodes 9

1.2.4 Finalize Configurations 9

Server Node File Configuration 9

1.3.1 Customizing the Install 10

1.3.2 Server Configuration 14

1.4 Manual Setup of Initial Server Configuration 16

1.5 Testing Server Configuration 17

2.0 Submitting and Managing Jobs 17

2.1 Job Submission 17

2.2 Monitoring Jobs 24

2.3 Canceling Jobs 25

2.4 Job Preemption 25

2.5 Completed Jobs 26

2.6 Job Checkpoint and Restart 26

BLCR Implementation Tests 33

2.7 Job Exit Status 43

3.0 Managing Nodes 46

3.1 Adding Nodes 46

3.2 Nodes Properties 46

3.3 Changing Node State 47

3.4 Host Security 47

3.5 Linux Cpuset Support 48

4.0 Setting Server Policies 50

4.1 Queue Configuration 50

4.2 Server High Availability 57

5.0 Interfacing with a Scheduler 61

5.1 Integrating Schedulers for TORQUE 61

6.0 Configuring Data Management 61

6.1 SCP/RCP Setup 61

6.2 NFS and Other Networked Filesystems 64

6.3 File Stage-In/Stage-Out 64

7.0 Interfacing with Message Passing 65

7.1 MPI (Message Passing Interface) Support 65

8.0 Managing Resources 67

8.1 Monitoring Resources 67

9.0 Accounting 69

9.1 Accounting Records 69

10.0 Troubleshooting 70

10.1 Troubleshooting 70

10.3 Debugging 80

Appendices 82

Appendix A: Commands Overview 82

Commands 83

Appendix B: Server Parameters 150

Appendix C: Node Manager (MOM) Configuration 155

Appendix D: Error Codes and Diagnostics 160

Appendix E: Considerations Before Upgrading 162

Appendix F: Large Clusters Considerations 163

Appendix F: Large Clusters Considerations 163

Appendix G: Prologue & Epilogue Scripts 168

Appendix H: Running Multiple TORQUE Servers and Moms On the Same Node 173

Appendix I: Security Overview 175

Appendix J: Job Submission Filter (aka 'qsub Wrapper')* 175

Appendix K: torque.cfg File 176

Appendix L: TORQUE Quick Start Guide 177

Appendix M: User's Guide 180

TORQUE Administrator's Guide

Preface

TORQUE Administrator Guide Overview

Note: Advanced TORQUE Administration is a video tutorial of a session offered at Moab Con that offers further details on advanced TORQUE administration.

This collection of documentation for TORQUE resource manager is intended as a reference for both users and system administrators.

The 1.0 Overview section provides the details for installation and basic, advanced, and (optional) manual configuration options necessary to get the system up and running. System Testing is also covered.

The 2.0 Submitting and Managing Jobs section covers different actions applicable to jobs. The first section, 2.1 Job Submission, details how to submit a job and request resources (nodes, software licenses, and so forth) and provides several examples. Other actions include monitoring, canceling, preemption, and keeping completed jobs.

The 3.0 Managing Nodes section covers administrator tasks relating to nodes, which includes the following: adding nodes, changing node properties, and identifying state. Also an explanation of how to configure restricted user access to nodes is covered in section 3.4 Host Security.

The 4.0 Setting Server Policies section details server side configurations of queue and with high availability matters.

The 5.0 Interfacing with a Scheduler section offers information about using the native scheduler versus an advanced scheduler.

The 6.0 Configuring Data Management section deals with issues of data management. For non-network file systems, the SCP/RCP Setup section details setting up SSH keys and nodes to automate transferring data. The NFS and Other Networked File Systems section covers configuration for these file systems. This chapter also addresses the use of File Stage-In/Stage-Out using the **stagein** and **stageout** directives of the qsub command.

The 7.0 Interfacing with Message Passing section offers details supporting MPI (Message Passing Interface).

The 8.0 Managing Resources section covers configuration, utilization, and states of resources.

The 9.0 Accounting section explains how jobs are tracked by TORQUE for accounting purposes.

The 10.0 Troubleshooting section is a troubleshooting guide that offers help with general problems; it includes an FAQ (Frequently Asked Questions) list and instructions for how to set up and use compute node checks and how to debug TORQUE.

The numerous appendices provide tables of commands, parameters, configuration options, error codes, the Quick Start Guide, and so forth.

- A. Commands Overview
- B. Server Parameters
- C. MOM Configuration
- D. Error Codes and Diagnostics
- E. Considerations Before Upgrading
- F. Large Cluster Considerations
- G. Prologue and Epilogue Scripts
- H. Running Multiple TORQUE Servers and Moms on the Same Node
- I. Security Overview
- J. Submit Filter (aka **qsub** Wrapper)
- K. torque.cfg File
- L. TORQUE Quick Start Guide
- M. User's Manual

What is a Resource Manager?

While TORQUE has a built-in scheduler, **pbs_sched**, it is typically used solely as a *resource manager* with a scheduler making requests to it. Resources managers provide the low-level functionality to start, hold, cancel, and monitor jobs. Without these capabilities, a scheduler alone can not control jobs.

What are Batch Systems?

While TORQUE is flexible enough to handle scheduling a conference room, it is primarily used in batch systems. Batch systems are a collection of computers and other resources (networks, storage systems, license servers, and so forth) that operate under the notion that the whole is greater than the sum of the parts. Some batch systems consist of just a handful of machines running single-processor jobs, minimally managed by the users themselves. Other systems have thousands and thousands of machines executing users' jobs simultaneously while tracking software licenses and access to hardware equipment and storage systems.

Pooling resources in a batch system typically reduces technical administration of resources while offering a uniform view to users. Once configured properly, batch systems abstract away many of the details involved with running and managing jobs, allowing higher resource utilization. For example, users typically only need to specify the minimal constraints of a job and do not need to know the individual machine names of each host on which they are running. With this uniform abstracted view, batch systems can execute thousands and thousands of jobs simultaneously.

Batch systems are comprised of four different components: (1) Master Node, (2) Submit/Interactive Nodes, (3) Compute Nodes, and (4) Resources.

- **Master Node** - A batch system will have a master node where **pbs_server** runs. Depending on the needs of the systems, a master node may be dedicated to this task, or it may fulfill the roles of other components as well.
- **Submit/Interactive Nodes** - Submit or interactive nodes provide an entry point to the system for users to manage their workload. For these nodes, users are able to submit and track their jobs. Additionally, some sites have one or more nodes reserved for interactive use, such as testing and troubleshooting environment problems. These nodes have client commands (such as **qsub** and **qhold**).
- **Compute Nodes** - Compute nodes are the workhorses of the system. Their role is to execute submitted jobs. On each compute node, **pbs_mom** runs to start, kill, and manage submitted jobs. It communicates with **pbs_server** on the master node. Depending on the needs of the systems, a compute node may double as the master node (or more).
- **Resources** - Some systems are organized for the express purpose of managing a collection of resources beyond compute nodes. Resources can include high-speed networks, storage systems, license managers, and so forth. Availability of these resources is limited and needs to be managed intelligently to promote fairness and increased utilization.

Basic Job Flow

The life cycle of a job can be divided into four stages: (1) creation, (2) submission, (3) execution, and (4) finalization.

Creation - Typically, a submit script is written to hold all of the parameters of a job. These parameters could include how long a job should run (**walltime**), what resources are necessary to run, and what to execute. The following is an example submit file:

```
#PBS -N localBlast
#PBS -S /bin/sh
#PBS -l nodes=1:ppn=2,walltime=240:00:00
#PBS -M user@my.organization.com
#PBS -m ea

source ~/.bashrc
cd $HOME/work/dir
sh myBlast.sh -i -v
```

This submit script specifies the name of the job (localBlast), what environment to use (`/bin/sh`), that it needs both processors on a single node (`nodes=1:ppn=2`), that it will run for at most 10 days, and that TORQUE should email user@my.organization.com when the job exits or aborts. Additionally, the user specifies where and what to execute.

- **Submission** - A job is submitted with the **qsub** command. Once submitted, the policies set by the administration and technical staff of the site dictate the priority of the job and therefore, when it will start executing.
- **Execution** - Jobs often spend most of their lifecycle executing. While a job is running, its status can be queried with **qstat**.
- **Finalization** - When a job completes, by default, the `stdout` and `stderr` files are copied to the directory where the job was submitted.

Glossary

Below are a list of terms that appear the documentation:

Epilogue

An optional script executed after a job completes. epilogue.user, epilogue.parallel and epilogue.precancel scripts also exist. See Appendix G: Prologue and Epilogue Scripts for more information.

Prologue

An optional script executed before a job starts. prologue.user and prologue.parallel scripts also exist. See Appendix G: Prologue and Epilogue Scripts for more information.

$TORQUE_HOME

The base directory for configuration directories. Defaults to /var/spool/torque (starting with version 2.1. Previously defaulted to /usr/spool/PBS)

1.0 Overview

1.1.1 TORQUE Architecture

A TORQUE cluster consists of one head node and many compute nodes. The head node runs the **pbs_server** daemon and the compute nodes run the **pbs_mom** daemon. Client commands for submitting and managing jobs can be installed on any host (including hosts not running **pbs_server** or **pbs_mom**).

The head node also runs a scheduler daemon. The scheduler interacts with pbs_server to make local policy decisions for resource usage and allocate nodes to jobs. A simple FIFO scheduler, and code to construct more advanced schedulers, is provided in the TORQUE source distribution. Most TORQUE users choose to use a packaged, advanced scheduler such as Maui or Moab.

Users submit jobs to **pbs_server** using the **qsub** command. When **pbs_server** receives a new job, it informs the scheduler. When the scheduler finds nodes for the job, it sends instructions to run the job with the node list to **pbs_serve**r. Then, **pbs_server** sends the new job to the first node in the node list and instructs it to launch the job. This node is designated the execution host and is called *Mother Superior*. Other nodes in a job are called *sister moms*.

1.1.2 Installing TORQUE

Build the distribution on the machine that will act as the TORQUE server - the machine which monitors and controls all compute nodes by running the **pbs_server** daemon.

Note: The built distribution package works only on compute nodes of a similar architecture. Nodes with different architecture must have the installation package built on them individually.

1. Download the TORQUE distribution file from http://clusterresources.com/downloads/torque.
2. Extract the packaged file and navigate to the unpackaged directory.

   ```
   > tar -xzvf torque-2.3.4.tar.gz
   > cd torque-2.3.4/
   ```

3. Configure the package.

   Note: For information on customizing the build at configure time, see the configure options list.

   ```
   > ./configure
   ```

4. Run **`make`** and **`make install`**.

   Note: TORQUE must be installed by a root user.

   ```
   > make
   > sudo make install
   ```

IMPORTANT!
OSX 10.4 users need to change **`#define __TDARWIN`** in `src/include/pbs_config.h` to **`#define __TDARWIN_8`**.

Note: After installation, verify you have **PATH** environment variables configured for `/usr/local/bin/` and `/usr/local/sbin/`.

TORQUE 2.0p2 and later includes a `torque.spec` file for building your own RPMs. You can also use the checkinstall program to create your own RPM, tgz, or deb package.

1.1.3 Compute Nodes

Use the Cluster Resources tpackage system to create self-extracting tarballs which can be distributed and installed on compute nodes. The tpackages are customizable. See the `INSTALL` file for additional options and features.

To create tpackages

1. Configure and make as normal, and then run **make packages**.

   ```
   > make packages
   Building ./torque-package-clients-linux-i686.sh ...
   Building ./torque-package-mom-linux-i686.sh ...
   Building ./torque-package-server-linux-i686.sh ...
   Building ./torque-package-gui-linux-i686.sh ...
   Building ./torque-package-devel-linux-i686.sh ...
   Done.

   The package files are self-extracting packages that can be copied
   and executed on your production machines.  Use --help for options.
   ```

2. Copy the desired packages to a shared location.

   ```
   > cp torque-package-mom-linux-i686.sh /shared/storage/
   > cp torque-package-clients-linux-i686.sh /shared/storage/
   ```

3. Install the tpackages on the compute nodes.

   Cluster Resources recommends you use a distributed shell to install tpackages on remote systems. The command is **dsh -f *<FILENAME>* *<COMMAND>***. *<FILENAME>* is a file with each line containing a host that you want to run the command. Set up SSH keys so you are not required to supply a password for each host.

   Note: The only required package for the compute nodes is **mom-linux**. Additional packages are recommended so you can use client commands and submit jobs from compute nodes.

   ```
   > dsh -f <FILENAME> torque-package-mom-linux-i686.sh --install
   > dsh -f <FILENAME> torque-package-clients-linux-i686.sh --install
   ```

   Note: You can use a tool like **xCAT** instead of **dsh**.

   1. Copy the tpackage to the nodes.

      ```
      > prcp torque-package-linux-i686.sh
      noderange:/destinationdirectory/
      ```

   2. Install the tpackage.

      ```
      > psh noderange /tmp/torque-package-linux-i686.sh --install
      ```

Alternatively, users with RPM-based Linux distributions can build RPMs from the source tarball in two ways.

- To use the default settings, use the **`rpmbuild`** command.

  ```
  >  rpmbuild -ta torque-2.3.4.tar.gz
  ```

- If configure options are required, untar and build as normal, and then use the **`make rpms`** command instead.

Although optional, it is possible to use the TORQUE server as a compute node and install a **pbs_mom** with the **pbs_server** daemon.

1.1.4 Enabling TORQUE as a service (optional)

The method for enabling TORQUE as a service is dependent on the Linux variant you are using. Startup scripts are provided in the `/contrib/init.d/` directory of the source package.

- **Red Hat (as root)**

-
  ```
  cp /contrib/init.d/pbs_mom /etc/init.d/pbs_mom
  chkconfig --add pbs_mom
  ```

- **SuSe (as root)**

-
  ```
  cp /contrib/init.d/suse.pbs_mom /etc/init.d/suse.pbs_mom
  insserv -d pbs_mom
  ```

- **Debian (as root)**

-
  ```
  cp /contrib/init.d/debian.pbs_mom /etc/init.d/debian.pbs_mom
  update-rc.d pbs_mom defaults
  ```

These options can be added to the self-extracting packages. For more details, see the `INSTALL` file.

See Also

- TORQUE Installation TroubleShooting

1.2.1 Initialize/Configure TORQUE on the Server (pbs_server)

Configure the **pbs_server** daemon by executing the command **`torque.setup`** ***<USER>***, where <*USER*> is a username that will act as the TORQUE administrator.

Note: Cluster Resources recommends that the TORQUE administrator be root.

1.2.2 Specify Compute Nodes

$TORQUECFG is where configuration files are stored. For TORQUE 2.1 and later, *$TORQUECFG* is `/var/spool/torque/`. For earlier versions, *$TORQUECFG* is `/usr/spool/PBS/`.

The **pbs_server** needs to know which systems on the network are its compute nodes. Each node must be specified on a line in the server's `nodes` file. This file is located at *`$TORQUECFG`*`/server_priv/nodes`. In most cases, it is sufficient to specify just the names of the nodes on individual lines, however, various properties can be applied to each node.

Syntax of nodes file

```
node-name[:ts] [np=] [properties]
```

The **`[:ts]`** option marks the node as timeshared. Timeshared nodes are listed by the server in the node status report, but the server does not allocate jobs to them.

The **`[np=]`** option specifies the number of processors the node has. Node processor count can be automatically detected by the TORQUE server if **`qmgr -c "set server auto_node_np = True"`** is used.

The **`[properties]`** option allows you to specify arbitrary strings to identify the node. Property strings are alphanumeric characters only and must begin with an alphabetic character.

Comment lines are allowed in the nodes file if the first non-white space character is the pound sign (#).

The example below shows a possible node file listing.

```
$TORQUECFG/server_priv/nodes

# Nodes 001 and 003-005 are cluster nodes
#
node001 np=2 cluster01 rackNumber22
#
# node002 will be replaced soon
node002:ts waitingToBeReplaced
# node002 will be replaced soon
#
node003 np=4 cluster01 rackNumber24
node004  cluster01 rackNumber25
node005 np=2 cluster01 rackNumber26 RAM16GB
node006
node007 np=2
node008:ts np=4
...
```

1.2.3 Configure TORQUE on the Compute Nodes

If using TORQUE self extracting packages with default compute node configuration, no additional steps are required and you can skip this section.

If installing manually, or advanced compute node configuration is needed, edit the `$TORQUECFG/mom_priv/config` file on each node. The recommended settings are below.

```
$TORQUECFG/mom_priv/config

$pbsserver      headnode          # note: hostname running pbs_server
$logevent       255               # bitmap of which events to log
```

This file is identical for all compute nodes and can be created on the head node and distributed in parallel to all systems.

1.2.4 Finalize Configurations

After **serverdb** and the **server_priv/nodes** file are configured, and MOM has a minimal configuration, restart the **pbs_server**.

```
> qterm -t quick
> pbs_server
```

After waiting several seconds, the **pbsnodes -a** command should list all nodes in state **free**.

See Also

- MOM Configuration Overview
- Advanced Configuration

Server Node File Configuration

Basic Node Specification

In order for the **pbs_server** to communicate with each of the moms, it needs to know which machines to contact. Each node which is to be a part of the batch system must be specified on a line in the server **nodes** file. This file is located at `$TORQUEHOME/server_priv/nodes`. In most cases, it is sufficient to specify just the node name on a line as in the following example:

server_priv/nodes

```
node001
node002
node003
node004
```

Specifying Node Count

If the compute node has multiple processors, specify the number of processors with `np=<number of processors>`. For example, if `node001` has `2` processors and `node002` has `4`:

server_priv/nodes

```
node001 np=2
node002 np=4
...
```

Specifying Node Features (aka Node Properties)

Node features can be specified by placing one or more whitespace delimited strings on the line for the associated host as in the example below:

server_priv/nodes

```
node001 np=2 fast ia64
node002 np=4 bigmem fast ia64 smp
...
```

These features can be used by users to request specific nodes when submitting jobs.

See Also

- Server Commands
- Moab Node Feature Overview

For additional information on this file, please see the **PBS Administrators Guide**.

1.3.1 Customizing the Install

The TORQUE **configure** command has several options available. Listed below are some suggested options to use when running **`./configure`**.

- By default, TORQUE does not install the admin manuals. To enable this, use `--enable-docs`

- By default, TORQUE uses **rcp** to copy data files. Using **scp** is highly recommended, use `--with-scp`
- By default, TORQUE does not enable **syslog** usage. To enable this, use `--enable-syslog`

Full Configure Options List

Optional features:

Option	**Description**
--disable-FEATURE	do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG]	include FEATURE [ARG=yes]
--enable-maintainer-mode	enable make rules and dependencies not useful and sometimes confusing) to the casual installer
--enable-autorun	turn on the AUTORUN_JOBS flag
--enable-maxnotdefault	turn on the RESOURCEMAXNOTDEFAULT flag
--enable-quickcommit	turn on the QUICKCOMMIT flag
--disable-qsub-keep-override	do not allow the qsub -k flag to override -o -e
--enable-nochildsignal	turn on the NO_SIGCHLD flag
--disable-server	do not include server and scheduler
--disable-mom	do not include the mom daemon
--disable-clients	do not include the clients
--disable-drmaa	do not build the DRMAA 1.0 library (default is off)
--disable-dependency-tracking	speeds up one-time build
--enable-dependency-tracking	do not reject slow dependency extractors
--enable-shared[=PKGS]	build shared libraries [default=yes]
--enable-static[=PKGS]	build static libraries [default=yes]
--enable-fast-install[=PKGS]	optimize for fast installation [default=yes]
--disable-libtool-lock	avoid locking (might break parallel builds)
--enable-debug	turn on the compilation of DEBUG code
--disable-gcc-warnings	Disable gcc strictness and warnings. If using gcc, default is to error on any warning
--disable-xopen-networking	With HPUX and GCC, don't force usage of XOPEN and libxnet
--enable-nodemask	enable nodemask-based scheduling on the Origin 2000
--enable-pemask	enable pemask-based scheduling on the Cray T3e

--enable-srfs	enable support for SRFS on Cray
--enable-sp2	build PBS for an IBM SP2
--enable-cpuset	enable Linux 2.6 kernel cpusets (in development)
--enable-array	setting this under IRIX enables the SGI Origin 2000 parallel support. Normally autodetected from the /etc/config/array file.
--enable-blcr	enable BLCR support
--enable-acct-x	enable adding x attributes to accounting log
--enable-cpa	enable Cray's CPA support
--disable-rpp	use RPP/UDP for resource queries from PBS server to mom. This is enabled by default. If disabled, TCP is used. This does not effect general node/job status messages, job launch/exit messages, inter-mom messages, etc.
--enable-filesync	open files with sync on each write operation. Don't bother enabling this, all it does is slow down TORQUE. This is disabled by default.
--enable-plock-daemons[=ARG]	enable daemons to lock themselves into memory: logical-or of 1 for pbs_server, 2 for pbs_scheduler, 4 for pbs_mom (no argument means 7 for all three)
--enable-syslog	enable (default) the use of syslog for error reporting
--disable-shell-pipe	give the job script file as standard input to the shell instead of passing its name via a pipe
--disable-spool	if disabled, TORQUE will create output and error files directly in $HOME/.pbs_spool if it exists or in $HOME otherwise. By default, TORQUE will spool files in $TORQUEHOME/spool and copy them to the users home directory when the job completes.
--enable-shell-use-argv	enable this to put the job script name on the command line that invokes the shell. Not on by default. Ignores --enable-shell-pipe setting.
--disable-posixmemlock	disable the moms use of mlockall. Some versions of OSs seem to have buggy POSIX MEMLOCK.
--disable-privports	disable the use of privileged ports for authentication. Some versions of OSX have a buggy bind() and cannot bind to privileged ports.
--disable-mom-checkspool	Don't check free space on spool directory and set an error
--enable-force-nodefile	forces creation of nodefile regardless of job submission parameters. Not on by default.
--enable-unixsockets	enable the use of Unix Domain sockets for authentication
--disable-gui	do not include the GUI-clients
--enable-tcl-qstat	setting this builds qstat with Tcl interpreter features. This is enabled if Tcl is enabled.

Optional Packages:

Option	**Description**
--with-PACKAGE[=ARG]	use PACKAGE [ARG=yes]
--without-PACKAGE	do not use PACKAGE (same as --with-PACKAGE=no)
--with-gnu-ld	assume the C compiler uses GNU ld [default=no]
--with-pic	try to use only PIC/non-PIC objects [default=use both]
--with-tags[=TAGS]	include additional configurations [automatic]

--with-debug	compile with debugging symbols
--with-cpa-include=DIR	include path for cpalib.h
--with-cpa-lib=DIR	lib path for libcpalib
--with-sched=TYPE	sets the scheduler type. If TYPE is "c" the scheduler will be written in C"tcl" the server will use a Tcl based scheduler "basl" will use the rule based scheduler "no" then their will be no scheduling done (the "c" scheduler is the default)
--with-sched-code=PATH	sets the name of the scheduler to use. This only applies to BASL schedulers and those written in the C language. For C schedulers this should be a directory name and for BASL schedulers a filename ending in ".basl". It will be interpreted relative to srctree/src/schedulers.SCHD_TYPE/samples. As an example, an appropriate BASL scheduler realtive path would be "nas.basl". The default scheduler code for "C" schedulers is "fifo".
--with-maildomain=MAILDOMAIN	override the default domain for outgoing mail messages, i.e. "user@maildomain". The default maildomain is the hostname where the job was submitted from.
--with-tmpdir=DIR	set the tmp directory that pbs_mom will use defaults to "/tmp". This is a Cray-specific feature.
--with-server-home=DIR	set the server home/spool directory for PBS use defaults to /var/spool/torque
--with-server-name-file=FILE	set the file that will contain the name of the default server for clients to use. If this is not an absolute pathname, it will be evaluated relative to the server home directory that either defaults to /usr/spool/PBS or is set using the --with-server-home option to configure. If this option is not specified, the default name for this file will be set to "server_name".
--with-default-server=HOSTNAME	set the name of the computer that clients will access when no machine name is specified as part of the queue name. It defaults to the hostname of the machine on which PBS is being compiled.
--with-environ=PATH	set the path containing the environment variables for the daemons. For SP2 and AIX systems, suggested setting is to /etc/environment. Defaults to the file "pbs_environment" in the server-home. Relative paths are interpreted within the context of the server-home.
--with-qstatrc-file=FILE	set the name of the file that qstat will use if there is no ".qstatrc" file in the directory where it is being invoked. Relative path names will be evaluated relative to the server home directory (see above). If this option is not specified, the default name for this file will be set to "qstatrc" (no dot) in the server home directory.
--with-momlogdir	use this directory for MOM logs.
--with-momlogsuffix	use this suffix for MOM logs.
--with-scp	use scp instead of mom_rcp (deprecated, use --with-rcp=scp).
--with-rcp	one of "scp", "rcp", "mom_rcp", or the fullpath of a remote file copy program. scp is the default if found, otherwise mom_rcp is used. Some rcp programs don't always exit with valid error codes in case of failure. mom_rcp is a copy of BSD rcp included with this source that has correct error codes, but it is also old, unmaintained, and doesn't have largefile support.
--with-sendmail[=FILE]	sendmail executable to use
--with-pam=DIR	Directory that holds the system PAM modules. Defaults to /lib(64)/security on Linux.
--with-xauth=PATH	Specify path to xauth program

--without-readline	do not include readline support (default: included if found)
--with-modulefiles[=DIR]	use modulefiles in specified directory [/etc/modulefiles]
--with-tcl	directory containing tcl configuration (tclConfig.sh)
--with-tclinclude	directory containing the public Tcl header files
--with-tclx	directory containing tclx configuration (tclxConfig.sh)
--with-tk	directory containing tk configuration (tkConfig.sh)
--with-tkinclude	directory containing the public Tk header files.
--with-tkx	directory containing tkx configuration (tkxConfig.sh)
--with-tclatrsep=CHAR	set the Tcl attribute separator character this will default to "." if unspecified

Recommended Configure Options - Most recommended configure options have been selected as default. The few exceptions include `--with-scp` and possibly `--enable-syslog`.

1.3.2 Server Configuration

1.3.2.1 Server Configuration Overview

There are several steps to ensure that the server and the nodes are completely aware of each other and able to communicate directly. Some of this configuration takes place within TORQUE directly using the **qmgr** command. Other configuration settings are managed using the pbs_server `nodes` file, DNS files such as /etc/hosts and the /etc/hosts.equiv file.

1.3.2.2 Name Service Config

Each node, as well as the server, must be able to resolve the name of every node with which it will interact. This can be accomplished using /etc/hosts, **DNS**, **NIS**, or other mechanisms. In the case of /etc/hosts, the file can be shared across systems in most cases.

A simple method of checking proper name service configuration is to verify that the server and the nodes can "ping" each other.

1.3.2.3 Configuring Job Submission Hosts

Using RCmd Authentication

When jobs can be submitted from several different hosts, these hosts should be trusted via the R* commands (such as rsh and rcp). This can be enabled by adding the hosts to the /etc/hosts.equiv file of the machine executing the **pbs_server** daemon or using other R* command authorization methods. The exact specification can vary from OS to OS (see the man page for ***ruserok*** to find out how your OS validates remote users). In most cases, configuring this file is as simple as adding a line to your /etc/hosts.equiv file, as in the following:

hosts.equiv

```
#[+ | -] [hostname] [username]
mynode.myorganization.com
.....
```

Please note that when a hostname is specified, it must be the fully qualified domain name (FQDN) of the host. Job submission can be further secured using the server or queue acl_hosts and acl_host_enabled parameters.

Using the `submit_hosts` Server Parameter

Trusted submit host access may be directly specified without using RCmd authentication by setting the server submit_hosts parameter via qmgr as in the following example:

qmgr

```
> qmgr -c 'set server submit_hosts = login1'
> qmgr -c 'set server submit_hosts += login2'
> qmgr -c 'set server submit_hosts += login3'
```

Note: Use of **submit_hosts** is potentially subject to DNS spoofing and should not be used outside of controlled and trusted environments.

Allowing Job Submission from Compute Hosts

If preferred, all compute nodes can be enabled as job submit hosts without setting `.rhosts` or `hosts.equiv` by setting the allow_node_submit parameter to **true**.

1.3.2.4 Configuring TORQUE on a Multi-Homed Server

If the **pbs_server** daemon is to be run on a multi-homed host (a host possessing multiple network interfaces), the interface to be used can be explicitly set using the SERVERHOST parameter.

1.3.2.5 Architecture Specific Notes

1.3.2.5.1 Mac OS/X Specific Notes

With some versions of Mac OS/X, it is required to add the line `$restricted *.<DOMAIN>` to the **pbs_mom** configuration file. This is required to work around some socket bind bugs in the OS.

1.3.2.6 Specifying Non-Root Administrators

By default, only root is allowed to start, configure and manage the **pbs_server** daemon. Additional trusted users can be authorized using the parameters **managers** and **operators**. To configure these parameters use the qmgr command, as in the following example:

moab.cfg

```
> qmgr

Qmgr: set server managers += josh@*.fsc.com
Qmgr: set server operators += josh@*.fsc.com
```

All manager and operator specifications must include a user name and either a fully qualified domain name or a host expression.

Note: To enable all users to be trusted as both operators and administrators, place the **+** (plus) character on its own line in the `server_priv/acl_svr/operators` and `server_priv/acl_svr/managers` files.

See Also

- Appendix B: Server Parameters

1.4 Manual Setup of Initial Server Configuration

Configuration of the pbs_server daemon is accomplished using the **qmgr** command. On a new system, the configuration database must be initialized using the command **pbs_server -t create**. Once this is done, the minimal configuration requires setting up the desired queue structure and enabling the scheduling interface.

The following example shows a simple one-queue configuration:

```
pbs_server -t create
qmgr -c "set server scheduling=true"
qmgr -c "create queue batch queue_type=execution"
qmgr -c "set queue batch started=true"
qmgr -c "set queue batch enabled=true"
qmgr -c "set queue batch resources_default.nodes=1"
qmgr -c "set queue batch resources_default.walltime=3600"
qmgr -c "set server default_queue=batch"
```

These commands need to be executed by root.

In this example, the configuration database is initialized and the scheduling interface is activated (using '`scheduling=true`'. This interface allows the scheduler to receive job and node events which allow it to be more responsive. The next step creates a queue and specifies the queue type. Within PBS, the queue must be declared an '`execution` queue in order for it to run jobs. Additional configuration (i.e., setting the queue to `started` and `enabled`) allows the queue to *accept* job submissions, and *launch* queued jobs.

The next two lines are optional, setting default `node` and `walltime` attributes for a submitted job. These defaults will be picked up by a job if values are not explicitly set be the submitting user. The final line, `default_queue=batch`, is also a convenience line and indicates that a job should be placed in the `batch` queue unless explicitly assigned to another queue.

Additional information on configuration can be found in the admin manual and in the qmgr man page.

1.5 Testing Server Configuration

The **pbs_server** daemon was started on the TORQUE server when the `torque.setup` file was executed or when it was manually configured. It must now be restarted so it can reload the updated configuration changes.

```
# verify all queues are properly configured
> qstat -q

# view additional server configuration
> qmgr -c 'p s'

# verify all nodes are correctly reporting
> pbsnodes -a

# submit a basic job - DO NOT RUN AS ROOT
> su - testuser
> echo "sleep 30" | qsub

# verify jobs display
> qstat
```

At this point, the job should be in the **Q** state and will not run because a scheduler is not running yet. TORQUE can use its native scheduler by running **pbs_sched** or an advanced scheduler (such as Moab Workload Manager). Section 5.1 Integrating Schedulers for TORQUE details setting up an advanced scheduler.

2.0 Submitting and Managing Jobs

2.1 Job Submission

- 2.1.1 Multiple Jobs Submission
- 2.1.2 Requesting Resources
- 2.1.3 Requesting Generic Resources
- 2.1.4 Requesting Floating Resources
- 2.1.5 Requesting Other Resources
- 2.1.6 Exported Batch Environment Variables
- 2.1.7 Enabling Trusted Submit Hosts

Job submission is accomplished using the qsub command, which takes a number of command line arguments and integrates such into the specified *PBS command file*. The PBS command file may be specified as a filename on the **qsub** command line or may be entered via STDIN.

- The PBS command file does not need to be executable.
- The PBS command file may be *piped* into **qsub** (i.e., `cat pbs.cmd | qsub`)
- In the case of parallel jobs, the PBS command file is staged to, and executed on, the first allocated compute node only. (Use pbsdsh to run actions on multiple nodes.)
- The command script is executed from the user's home directory in all cases. (The script may determine the submission directory by using the $PBS_O_WORKDIR environment variable)
- The command script will be executed using the default set of user environment variables unless the **-V** or **-v** flags are specified to include aspects of the job submission environment.

By default, job submission is allowed only on the TORQUE server host (host on which **pbs_server** is running). Enablement of job submission from other hosts is documented in Configuring Job Submit Hosts.

2.1.1 Multiple Jobs Submission

Sometimes users will want to submit large numbers of jobs based on the same job script. Rather than using a script to repeatedly call **qsub**, a feature known as job arrays now exists to allow the creation of multiple jobs with one **qsub** command. Additionally, this feature includes a new job naming convention that allows users to reference the entire set of jobs as a unit, or to reference one particular job from the set.

Example

```
qsub -t 0-4 job_script
1098.hostname

qstat
1098-0.hostname ...
1098-1.hostname ...
1098-2.hostname ...
1098-3.hostname ...
1098-4.hostname ...
```

Job arrays are submitted through the **-t** option to **qsub**, or by using `#PBS -t` in your batch script. This option takes a comma-separated list consisting of either a single job ID number, or a pair of numbers separated by a dash. Each of these jobs created will use the same script and will be running in a nearly identical environment.

Versions of TORQUE earlier than 2.3 had different semantics for the **-t** arguement. In these versions, **-t** took a single integer number—a count of the number of jobs to be created.

Syntax Example

```
qsub -t 0-99 would be equvalent to qsub -t 100 in torque 2.2

you can also pass comma delimited lists of ids and ranges:

qsub -t 0,10,20,30,40

or

qsub -t 0-50,60,70,80
```

Each 1098-x job has an environment variable called PBS_ARRAYID, which is set to the value of the array index of the job, so 1098-0.hostname would have PBS_ARRAYID set to 0. This will allow you to create job arrays where each job in the array will perform slightly different actions based on the value of this variable, such as performing the same tasks on different input files. One other difference in the environment between jobs in the same array is the value of the PBS_JOBNAME variable.

Currently, each job in the array shows up when **qstat** is run. Essentially they are fully independent TORQUE jobs. All normal TORQUE commands will work on the individual jobs. Eventually, as the job arrays are further developed, a single entry in **qstat** would be displayed which would summarize the job. An additional flag for **qstat** would be provided that would show the details of an array. Currently the only TORQUE command that operates on an array as a whole is the **qdel** command. In the previous example, `qdel 1098` would delete every job in the array, while `qdel 1098-0` would delete just that one job. Support for **qhold** and **qrls** on an entire array will be available shortly, and array awareness will be added to all TORQUE commands one at a time.

Please be aware that job arrays are under development and may have bugs, and certainly are not feature complete. If you have any suggestions or bug reports, please bring them to our attention.

We are currently aware of one limitation that causes the creation of large job arrays on TORQUE installations with high job IDs. This is due to a historical file name length limitation. With thousands of similarly named jobs (and for the time being, each of these jobs has its own job file within pbs_server—they do all share a copy of the script file while on the server), the file name hashing algorithm that attempts to create unique short file names encounters limitations. This limitation for job arrays or sharing the job file will eventually be removed.

2.1.2 Requesting Resources

Various resources can be requested at the time of job submission. A job can request a particular node, a particular node attribute, or even a number of nodes with particular attributes. Either native TORQUE resources, or external scheduler resource extensions may be specified. The native TORQUE resources are listed in the following table:

Resource	Format	Description
arch	string	Specifies the administrator defined

		system architecture required. This defaults to whatever the **PBS_MACH** string is set to in "local.mk".
cput	seconds, or [[HH:]MM:]SS	Maximum amount of CPU time used by all processes in the job.
file	size*	The amount of total disk requested for the job. (Ignored on Unicos.)
host	string	Name of the host on which the job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent.
mem	size*	Maximum amount of physical memory used by the job. (Ignored on Darwin, Digital Unix, Free BSD, HPUX 11, IRIX, NetBSD, and SunOS. Also ignored on Linux if number of nodes is not 1. Not implemented on AIX and HPUX 10.)
nice	integer	Number between -20 (highest priority) and 19 (lowest priority). Adjust the process execution priority.
nodes	{<node_count> | <hostname>} [:ppn=<ppn>][:<property>[:<property>]...] [+ ...]	Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more node_specs joined with the **+** (plus) character: node_spec[+node_spec...]. Each node_spec is a number of nodes required of the type declared in the node_spec and a name of one or more properties desired for the nodes. The number, the name, and each property in the node_spec are separated by a **:** (colon). If no number is specified, one (1) is assumed. The name of a node is its hostname. The properties of nodes are: • **ppn=**# - specify the number of processors per node requested. Defaults to 1. • property - a string assigned by the system administrator specifying a node's features. Check with your administrator as to the node names and properties available to you. See Example 1 (-l nodes) for examples. **NOTE**: By default, the **node** resource is mapped to a virtual node (that is, directly to a processor, not a full physical compute node). This behavior

		can be changed within Maui or Moab by setting the JOBNODEMATCHPOLICY parameter. (See Appendix F of the Moab Workload Manager Administrator's Guide for more information.)
opsys	string	Specifies the administrator defined operating system as defined in the mom configuration file.
other	string	Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent.
pcput	seconds, or [[HH:]MM:]SS	Maximum amount of CPU time used by any single process in the job.
pmem	size*	Maximum amount of physical memory used by any single process of the job. (Ignored on Fujitsu. Not implemented on Digital Unix and HPUX.)
pvmem	size*	Maximum amount of virtual memory used by any single process in the job. (Ignored on Unicos.)
software	string	Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. (See Scheduler License Management in the Moab Workload Manager Administrator's Guide for more information.)
vmem	size*	Maximum amount of virtual memory used by all concurrent processes in the job. (Ignored on Unicos.)
walltime	seconds, or [[HH:]MM:]SS	Maximum amount of real time during which the job can be in the running state.

***size** format:
The size format specifies the maximum amount in terms of bytes or words. It is expressed in the form *integer[suffix]*. The suffix is a multiplier defined in the following table ('b' means bytes (the default) and 'w' means words). The size of a word is calculated on the execution server as its word size.

Suffix		Multiplier
b	w	1
kb	kw	1024
mb	mw	1,048,576
gb	gw	1,073,741,824
tb	tw	1,099,511,627,776

Example 1 (-l nodes)

Usage	Description
qsub -l `> qsub -l nodes=12`	request 12 nodes of any type
qsub -l `> qsub -l nodes=2:server+14`	request 2 "server" nodes and 14 other nodes (a total of 16) - this specifies two node_specs, "2:server" and "14"
qsub -l `> qsub -l nodes=server:hippi+10:noserver+3:bigmem:hippi`	request (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi
qsub -l `> qsub -l nodes=b2005+b1803+b1813`	request 3 specific nodes by hostname
qsub -l `> qsub -l nodes=4:ppn=2`	request 2 processors on each of four nodes
qsub -l `> qsub -l nodes=1:ppn=4`	request 4 processors on one node
qsub -l `> qsub -l nodes=2:blue:ppn=2+red:ppn=3+b1014`	request 2 processors on each of two blue nodes, three processors on one red node, and the compute node "b1014"

Example 2

```
> qsub -l mem=200mb /home/user/script.sh
```

This job requests a node with 200 MB of available memory.

Example 3

```
> qsub -l nodes=node01,mem=200mb /home/user/script.sh
```

This job will wait until node01 is free with 200 MB of available memory.

2.1.3 Requesting Generic Resources

When **generic** resources have been assigned to nodes using the server's nodes file, these resources can be requested at the time of job submission using the ***other*** field. (See the Consumable Generic Resources page in the Moab Workload Manager Administrator's Guide for details on configuration within Moab).

Example 1

```
> qsub -l other=matlab /home/user/script.sh
```
This job will run on any node that has the generic resource ***matlab***.

NOTE: This can also be requested at the time of job submission using the ***-W x=GRES:matlab*** flag.

2.1.4 Requesting Floating Resources

When **floating** resources have been set up inside Moab, they can be requested in the same way as **generic** resources. Moab will automatically understand that these resources are **floating** and will schedule the job accordingly. (See the Floating Generic Resources page in the Moab Workload Manager Administrator's Guide for details on configuration within Moab.)

Example 2

```
> qsub -l other=matlab /home/user/script.sh
```
This job will run on any node when there are enough floating resources available.

NOTE: This can also be requested at the time of job submission using the ***-W x=GRES:matlab*** flag.

2.1.5 Requesting Other Resources

Many other resources can be requested at the time of job submission using the Moab Workload Manger. See the Resource Manager Extensions page in the Moab Workload Manager Administrator's Guide for a list of these supported requests and correct syntax.

2.1.6 Exported Batch Environment Variables

When a batch job is started, a number of variables are introduced into the job's environment that can be used by the batch script in making decisions, creating output files, and so forth. These variables are listed in the following table:

Variable	Description
PBS_JOBNAME	user specified jobname
PBS_ARRAYID	zero-based value of job array index for this job (in version 2.2.0 and later)
PBS_O_WORKDIR	job's submission directory
PBS_ENVIRONMENT	N/A

PBS_TASKNUM	number of tasks requested
PBS_O_HOME	home directory of submitting user
PBS_MOMPORT	active port for mom daemon
PBS_O_LOGNAME	name of submitting user
PBS_O_LANG	language variable for job
PBS_JOBCOOKIE	job cookie
PBS_NODENUM	node offset number
PBS_O_SHELL	script shell
PBS_O_JOBID	unique pbs job id
PBS_O_HOST	host on which job script is currently running
PBS_QUEUE	job queue
PBS_NODEFILE	file containing line delimited list on nodes allocated to the job
PBS_O_PATH	path variable used to locate executables within job script

2.1.7 Enabling Trusted Submit Hosts

By default, only the node running the **pbs_server** daemon is allowed to submit jobs. Additional nodes can be trusted as submit hosts by taking any of the following steps:

- Set the allow_node_submit server parameter.
  - Allows any host trusted as a compute host to also be trusted as a submit host.
- Set the submit_hosts server parameter (comma-delimited).
  - Allows specified hosts to be trusted as a submit host.
- Use .rhosts to enable *ruserok()* based authentication.

See Job Submission Host Advanced Config for more information.

NOTE: If **allow_node_submit** is set, the parameter **allow_proxy_user** must be set to allow user proxying when submitting/running jobs.

See Also

- Maui Documentation
- qsub wrapper - Allow local checking and modification of submitted jobs

2.2 Monitoring Jobs

TORQUE allows users and administrators to monitor submitted jobs with the qstat command. If the command is run by just a user, it will output just that user's jobs. For example:

```
> qstat

Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4807             scatter          user01           12:56:34 R batch
...
```

2.3 Canceling Jobs

TORQUE allows users and administrators to cancel submitted jobs with the qdel command. The job will be sent TERM and KILL signals killing the running processes. When the top-level job script exits, the job will exit. Simply supply it to the job ID to be canceled.

If a job is canceled by an operator or manager, an email notification will be sent to the user. Operators and managers may add a comment to this email with the **-m** option.

```
$ qstat
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4807             scatter          user01           12:56:34 R batch
...
$ qdel -m "hey! Stop abusing the NFS servers" 4807
$
```

2.4 Job Preemption

TORQUE supports job preemption by allowing authorized users to suspend and resume jobs. This is supported using one of two methods. If the node supports OS-level preemption, TORQUE will recognize that during the configure process and enable it. Otherwise, the MOM may be configured to launch a custom *checkpoint script* in order to support preempting a job. Using a custom checkpoint script requires that the job understand how to resume itself from a checkpoint after the preemption occurs.

Configuring a Checkpoint Script on a MOM

To configure the MOM to support a checkpoint script, the `$checkpoint_script` parameter must be set in the MOM's configuration file found in `$TORQUEHOME/mom_priv/config`. The checkpoint script should have execute permissions set. A typical configuration file might look as follows:

mom_priv/config

```
$pbsserver          node06
$logevent           255
$restricted         *.mycluster.org
$checkpoint_script  /opt/moab/tools/mom-checkpoint.sh
```

The second thing that must be done to enable the checkpoint script is to change the value of `MOM_CHECKPOINT` to `1` in `.../src/include/pbs_config.h`. In some instances, `MOM_CHECKPOINT` may already be defined as `1`. The new line should be as follows:

.../src/include/pbs_config.h

```
#define MOM_CHECKPOINT 1
```

2.5 Completed Jobs

TORQUE provides the ability to report on the status of completed jobs for a configurable duration after the job has completed. This can be enabled by setting the **keep_completed** attribute on the job execution queue. This should be set to the number of seconds that jobs should be held in the queue. Completed jobs will be reported in the `C` state and the exit status is seen in the `exit_status` job attribute.

By maintaining status information about completed (or canceled, failed, etc.) jobs, administrators can better track failures and improve system performance. This also allows TORQUE to better communicate with Moab Workload Manager and track the status of jobs. This also gives Moab the ability to track specific failures and to schedule the workload around possible hazards. (See NODEFAILURERESERVETIME in Appendix F of the Moab Workload Manager Administrator's Guide for more information.)

2.6 Job Checkpoint and Restart

While TORQUE has had a job checkpoint and restart capability for many years, this was tied to machine specific features. Now there is an architecture independent package available that provides for process checkpoint and restart. The package is BLCR and TORQUE now provides support for BLCR.

Introduction to BLCR

BLCR is a kernel level package. It must be downloaded and installed from BLCR.

After building and making the package, it must be installed into the kernel with commands as follows. These can be installed into the file **/etc/modules** but all of the testing was done with explict invocations of **modprobe**.

Installing BLCR into the kernel

```
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr_imports.ko
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr_vmadump.ko
# /sbin/insmod /usr/local/lib/blcr/2.6.12-1.234/blcr.ko
```

The BLCR system provides four command line utilities: (1) **cr_checkpoint**, (2) **cr_info**, (3) **cr_restart**, and (4) **cr_run**.

For more information about BLCR, see the BLCR Administrator's Guide.

Note that the support for BLCR is at the beta stage, and this support should be enabled explicitly when configuring and building TORQUE. BLCR support is available in the 2.4 (trunk) version of TORQUE.

The following shows the configure options used in developing and testing the implementation of support for BLCR.

Configuring and Building TORQUE for BLCR

```
>  ./configure --enable-unixsockets=no --enable-blcr
>  make
>  sudo make install
```

Configuration files and scripts

The pbs_mom configuration file located in /var/spool/torque/mom_priv must be modified to identify the script names associated with invoking the BLCR commands. The following variables should be used in the configuration file when using BLCR checkpointing.

- $checkpoint_interval - How often periodic job checkpoints will be taken (minutes).
- $checkpoint_script - The name of the script file to execute to perform a job checkpoint.
- $restart_script - The name of the script file to execute to perform a job restart.
- $checkpoint_run_exe - The name of an executable program to be run when starting a checkpointable job (for BLCR, cr_run).

The following example shows the contents of the configuration file used for testing the BLCR feature in TORQUE.

Note that the script files below must be executable by the user. Be sure to use **chmod** to set the permissions to 754.

Script file permissions

```
# chmod 754 blcr*
# ls -l
total 20
-rwxr-xr-- 1 root root 2112 2008-03-11 13:14 blcr_checkpoint_script
-rwxr-xr-- 1 root root 1987 2008-03-11 13:14 blcr_restart_script
-rw-r--r-- 1 root root  215 2008-03-11 13:13 config
drwxr-x--x 2 root root 4096 2008-03-11 13:21 jobs
-rw-r--r-- 1 root root    7 2008-03-11 13:15 mom.lock
```

mom_priv/config

```
$checkpoint_script  /var/spool/torque/mom_priv/blcr_checkpoint_script
$restart_script  /var/spool/torque/mom_priv/blcr_restart_script
$checkpoint_run_exe /usr/local/bin/cr_run
$pbsserver makua.cridomain
$loglevel 7
```

mom_priv/blcr_checkpoint_script

```
#! /usr/bin/perl
```

```
###############################################################################
###
#
# Usage: checkpoint_script
#
# This script is invoked by pbs_mom to checkpoint a job.
#
###############################################################################
###
use strict;
use Sys::Syslog;

# Log levels:
# 0 = none -- no logging
# 1 = fail -- log only failures
# 2 = info -- log invocations
# 3 = debug -- log all subcommands
my $logLevel = 3;

logPrint(2, "Invoked: $0 " . join(' ', @ARGV) . "\n");

my ($sessionId, $jobId, $userId, $signalNum, $checkpointDir,
$checkpointName);
my $usage =
  "Usage: $0        \n";

# Note that depth is not used in this script but could control a limit to the
number of checkpoint
# image files that are preserved on the disk.
#
# Note also that a request was made to identify whether this script was
invoked by the job's
# owner or by a system administrator.  While this information is known to
pbs_server, it
# is not propagated to pbs_mom and thus it is not possible to pass this to
the script.
# Therefore, a workaround is to invoke qmgr and attempt to set a trivial
variable.
# This will fail if the invoker is not a manager.

if (@ARGV == 7)
{
    ($sessionId, $jobId, $userId, $checkpointDir, $checkpointName, $signalNum
$depth) =
      @ARGV;
}
else { logDie(1, $usage); }

# Change to the checkpoint directory where we want the checkpoint to be
created
chdir $checkpointDir
  or logDie(1, "Unable to cd to checkpoint dir ($checkpointDir): $!\n")
  if $logLevel;

my $cmd = "cr_checkpoint";
$cmd .= " --signal $signalNum" if $signalNum;
$cmd .= " --tree $sessionId";
```

```
$cmd .= " --file $checkpointName";
my $output = `$cmd 2>&1`;
my $rc     = $? >> 8;
logDie(1, "Subcommand ($cmd) failed with rc=$rc:\n$output")
  if $rc && $logLevel >= 1;
logPrint(3, "Subcommand ($cmd) yielded rc=$rc:\n$output")
   if $logLevel >= 3;
exit 0;

##############################################################################
###
# logPrint($message)
# Write a message (to syslog) and die
##############################################################################
###
sub logPrint
{
    my ($level, $message) = @_;
    my @severity = ('none', 'warning', 'info', 'debug');

    return if $level > $logLevel;

    openlog('checkpoint_script', '', 'user');
    syslog($severity[$level], $message);
    closelog();
}

##############################################################################
###
# logDie($message)
# Write a message (to syslog) and die
##############################################################################
###
sub logDie
{
    my ($level, $message) = @_;

    logPrint($level, $message);
    die($message);
}
```

mom_priv/blcr_restart_script

```
#! /usr/bin/perl
##############################################################################
###
#
# Usage: restart_script
#
# This script is invoked by pbs_mom to restart a job.
#
##############################################################################
###
use strict;
use Sys::Syslog;
```

```
# Log levels:
# 0 = none -- no logging
# 1 = fail -- log only failures
# 2 = info -- log invocations
# 3 = debug -- log all subcommands
my $logLevel = 3;

logPrint(2, "Invoked: $0 " . join(' ', @ARGV) . "\n");

my ($sessionId, $jobId, $userId, $checkpointDir, $restartName);
my $usage =
  "Usage: $0      \n";
if (@ARGV == 5)
{
    ($sessionId, $jobId, $userId, $checkpointDir, $restartName) =
      @ARGV;
}
else { logDie(1, $usage); }

# Change to the checkpoint directory where we want the checkpoint to be
created
chdir $checkpointDir
  or logDie(1, "Unable to cd to checkpoint dir ($checkpointDir): $!\n")
  if $logLevel;


my $cmd = "cr_restart";
$cmd .= " $restartName";
my $output = `$cmd 2>&1`;
my $rc     = $? >> 8;
logDie(1, "Subcommand ($cmd) failed with rc=$rc:\n$output")
  if $rc && $logLevel >= 1;
logPrint(3, "Subcommand ($cmd) yielded rc=$rc:\n$output")
   if $logLevel >= 3;
exit 0;

###############################################################################
###
# logPrint($message)
# Write a message (to syslog) and die
###############################################################################
###
sub logPrint
{
    my ($level, $message) = @_;
    my @severity = ('none', 'warning', 'info', 'debug');

    return if $level > $logLevel;

    openlog('restart_script', '', 'user');
    syslog($severity[$level], $message);
    closelog();
}

###############################################################################
###
# logDie($message)
# Write a message (to syslog) and die
################################################################################
###
sub logDie
{
    my ($level, $message) = @_;

    logPrint($level, $message);
    die($message);
}
```

Starting a checkpointable job

Not every job is checkpointable. A job for which checkpointing is desirable must be started with the **-c** command line option. This option takes a comma-separated list of arguments that are used to control checkpointing behavior. The list of valid options available in the 2.4 version of Torque is show below.

- **none** - No checkpointing (not highly useful, but included for completeness).
- **enabled** - Specify that checkpointing is allowed, but must be explicitly invoked by either the **qhold** or **qchkpt** commands.
- **shutdown** - Specify that checkpointing is to be done on a job at pbs_mom shutdown.
- **periodic** - Specify that periodic checkpointing is enabled. The default interval is 10 minutes and can be changed by the $checkpoint_interval option in the mom configuration file, or by specifying an interval when the job is submitted.
- **interval=minutes** - Specify the checkpoint interval in minutes.
- **depth=number** - Specify a number (depth) of checkpoint images to be kept in the checkpoint directory.
- **dir=path** - Specify a checkpoint directory (default is /var/spool/torque/checkpoint).

Sample test program

```
#include "stdio.h"
int main( int argc, char *argv[] )
{
int i;
        for (i=0; i<100; i++)
        {
                printf("i = %d\n", i);
                fflush(stdout);
                sleep(1);
        }
}
```

Instructions for building test program

```
>  gcc -o test test.c
```

Sample test script

```
#!/bin/bash
```

```
./test
```

Starting the test job

```
>  qstat
>  qsub -c enabled,periodic,shutdown,interval=1 test.sh
77.jakaa.cridomain
>  qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
77.jakaa                  test.sh          jsmith                 0 Q batch
>
```

If you have no scheduler running, you might need to start the job with qrun.

As this program runs, it writes its output to a file in /var/spool/torque/spool. This file can be observered with the command **tail -f**.

Checkpointing a job

Jobs are checkpointed by issuing a qhold command. This causes an image file representing the state of the process to be written to disk. The directory by default is /var/spool/torque/checkpoint.

This default can be altered at the queue level with the **qmgr** command. For example, the command **qmgr -c set queue batch checkpoint_dir=/tmp** would change the checkpoint directory to **/tmp**.

The default directory can also be specified at job submission time with the **-c dir=/tmp** command line option.

The name of the checkpoint directory and the name of the checkpoint image file become attributes of the job and can be observed with the command **qstat -f**. Notice in the output the names **checkpoint_dir** and **checkpoint_name**. The variable **checkpoint_name** is set when the image file is created and will not exist if no checkpoint has been taken.

A job can also be checkpointed without stopping or holding the job with the command **qchkpt**.

Periodic job checkpoints

A job can have checkpoints taken at a regular interval without causing the job to be terminated. This option is enabled by starting the job with the **qsub -c periodic,interval=n job-script** syntax. Then **n** argument specifies a number of minutes between checkpoints. See qsub for more details.

Restarting a job in the Held state

The qrls command is used to restart the hibernated job. If you were using the **tail -f** command to watch the output file, you will see the test program start counting again.

It is possible to use the qalter command to change the name of the checkpoint file associated with a job. This could be useful if there were several job checkpoints and it restarting the job from an older image was specified.

Restarting a job in the Completed state

In this case, the job must be moved to the Queued state with the qrerun command. Then the job must go to the Run state either by action of the scheduler or if there is no scheduler, through using the qrun command.

Acceptance tests

A number of tests were made to verify the functioning of the BLCR implementation. See tests-2.4 for a description of the testing.

BLCR Implementation Tests

rmine where the source of this problem resides.

Test environment

All these tests assume the following test program and shell script, test.sh.

```
#include
int main( int argc, char *argv[] )
{
int i;

    for (i=0; i<100; i++)
    {
        printf("i = %d\n", i);
        fflush(stdout);
        sleep(1);
    }
}
```

```
#!/bin/bash

/home/test/test
```

Test 1 - Basic operation

Introduction

This test determines if the proper environment has been established.

Test Steps

Submit a test job and the issue a hold on the job.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
```

Possible Failures

Normally the result of **qhold** is nothing. If an error message is produced saying that **qhold** is not a supported feature then one of the following configuration errors might be present.

- The Torque images may have not be configured with **--enable-blcr**
- BLCR support may not be installed into the kernel with insmod.
- The config script in mom_priv may not exist with **$checkpoint_script** defined.
- The config script in mom_priv may not exist with **$restart_script** defined.
- The config script in mom_priv may not exist with **$checkpoint_run_exe** defined.
- The scripts referenced in the config file may not exist.
- The scripts referenced in the config file may not have the correct permissions.

Successful Results

If no configuration was done to specify a specific directory location for the checkpoint file, the default location is off of the Torque directory which in my case is **/var/spool/torque/checkpoint**.

Otherwise, go to the specified directory for the checkpoint image files. This was done by either specifying an option on job submission i.e. **-c dir=/home/test** or by setting an attribute on the execution quere. This is done with the command **qmgr -c 'set queue batch checkpoint_dir=/home/test'**.

Doing a directory listing shows the following.

```
# find /var/spool/torque/checkpoint
/var/spool/torque/checkpoint
/var/spool/torque/checkpoint/999.xxx.yyy.CK
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630
# find /var/spool/torque/checkpoint |xargs ls -l
-r-------- 1 root root 543779 2008-03-11 14:17
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630

/var/spool/torque/checkpoint:
total 4
```

```
drwxr-xr-x 2 root root 4096 2008-03-11 14:17 999.xxx.yyy.CK

/var/spool/torque/checkpoint/999.xxx.yyy.CK:
total 536
-r-------- 1 root root 543779 2008-03-11 14:17 ckpt.999.xxx.yyy.1205266630
```

Doing a **qstat -f** command should show the job in a held state, **job_state = H**. Note that the attribute **checkpoint_name** is set to the name of the file seen above.

If a checkpoint directory has been specified, there will also be an attribute **checkpoint_dir** in the output of **qstat -f**.

```
$ qstat -f
Job Id: 999.xxx.yyy
    Job_Name = test.sh
    Job_Owner = test@xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:06
    job_state = H
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Tue Mar 11 14:17:04 2008
    Error_Path = xxx.yyy:/home/test/test.sh.e999
    exec_host = test/0
    Hold_Types = u
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Tue Mar 11 14:17:10 2008
    Output_Path = xxx.yyy:/home/test/test.sh.o999
    Priority = 0
    qtime = Tue Mar 11 14:17:04 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 9402
    substate = 20
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
        bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 999.xxx.yyy
    queue_rank = 3
    queue_type = E
    comment = Job started on Tue Mar 11 at 14:17
    exit_status = 271
    submit_args = test.sh
    start_time = Tue Mar 11 14:17:04 2008
    start_count = 1
    checkpoint_dir = /var/spool/torque/checkpoint/999.xxx.yyy.CK
    checkpoint_name = ckpt.999.xxx.yyy.1205266630
```

Test 2 - Persistance of checkpoint images

Introduction

This test determines if the checkpoint files remain in the default directory after the job is removed from the Torque queue.

Note that this behavior was requested by a customer but in fact may not be the right thing to do as it leaves the checkpoint files on the execution node. These will gradually build up over time on the node being limited only by disk space. The right thing would seem to be that the checkpoint files are copied to the users home directory after the job is purged from the execution node.

Test Steps

Assuming the steps of Test 1, delete the job and then wait until the job leaves the queue after the completed job hold time. Then look at the contents of the default checkpoint directory to see if the files are still there.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qdel 999
> sleep 100
> qstat
>
> find /var/spool/torque/checkpoint
... files ...
```

Possible Failures

The files are not there, did Test 1 actually pass?

Successful Results

The files are there.

Test 3 - Restart after checkpoint

Introduction

This test determines if the job can be restarted after a checkpoint hold.

Test Steps

Assuming the steps of Test 1, issue a **qrls** command. Have another window open into the **/var/spool/torque/spool** directory and tail the job.

Successful Results

After the **qrls**, the job's output should resume.

Test 4 - Multiple checkpoint / restart

Introduction

This test determines if the checkpoint/restart cycle can be repeated multiple times.

Test Steps

Start a job and then while **tail**'ing the job output, do multiple **qhold**/**qrls** operations.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qrls 999
> qhold 999
> qrls 999
> qhold 999
> qrls 999
```

Successful Results

After each **qrls**, the job's output should resume. Also tried "while true; do qrls 999; qhold 999; done" and this seemed to work as well.

Test 5 - Periodic checkpoint

Introduction

This test determines if automatic periodic checkpoint will work.

Test Steps

Start the job with the option **-c enabled,periodic,interval=1** and look in the checkpoint directory for checkpoint images to be generated about every minute.

```
> qsub -c enabled,periodic,interval=1 test.sh
999.xxx.yyy
```

Successful Results

The checkpoint directory should contain multiple checkpoint images and the time on the files should be roughly a minute apart.

Test 6 - Restart from previous image

Introduction

This test determines if the

Test 1 - Basic operation

Introduction

This test determines if the proper environment has been established.

Test Steps

Submit a test job and the issue a hold on the job.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
```

Possible Failures

Normally the result of **qhold** is nothing. If an error message is produced saying that **qhold** is not a supported feature then one of the following configuration errors might be present.

- The Torque images may have not be configured with **--enable-blcr**
- BLCR support may not be installed into the kernel with insmod.
- The config script in mom_priv may not exist with **$checkpoint_script** defined.
- The config script in mom_priv may not exist with **$restart_script** defined.

- The config script in mom_priv may not exist with **$checkpoint_run_exe** defined.
- The scripts referenced in the config file may not exist.
- The scripts referenced in the config file may not have the correct permissions.

Successful Results

If no configuration was done to specify a specific directory location for the checkpoint file, the default location is off of the Torque directory which in my case is **/var/spool/torque/checkpoint**.

Otherwise, go to the specified directory for the checkpoint image files. This was done by either specifying an option on job submission i.e. **-c dir=/home/test** or by setting an attribute on the execution quere. This is done with the command **qmgr -c 'set queue batch checkpoint_dir=/home/test'**.

Doing a directory listing shows the following.

```
# find /var/spool/torque/checkpoint
/var/spool/torque/checkpoint
/var/spool/torque/checkpoint/999.xxx.yyy.CK
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630
# find /var/spool/torque/checkpoint |xargs ls -l
-r-------- 1 root root 543779 2008-03-11 14:17
/var/spool/torque/checkpoint/999.xxx.yyy.CK/ckpt.999.xxx.yyy.1205266630

/var/spool/torque/checkpoint:
total 4
drwxr-xr-x 2 root root 4096 2008-03-11 14:17 999.xxx.yyy.CK

/var/spool/torque/checkpoint/999.xxx.yyy.CK:
total 536
-r-------- 1 root root 543779 2008-03-11 14:17 ckpt.999.xxx.yyy.1205266630
```

Doing a **qstat -f** command should show the job in a held state, **job_state = H**. Note that the attribute **checkpoint_name** is set to the name of the file seen above.

If a checkpoint directory has been specified, there will also be an attribute **checkpoint_dir** in the output of **qstat -f**.

```
$ qstat -f
Job Id: 999.xxx.yyy
    Job_Name = test.sh
    Job_Owner = test@xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:06
    job_state = H
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Tue Mar 11 14:17:04 2008
    Error_Path = xxx.yyy:/home/test/test.sh.e999
    exec_host = test/0
    Hold_Types = u
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Tue Mar 11 14:17:10 2008
    Output_Path = xxx.yyy:/home/test/test.sh.o999
    Priority = 0
    qtime = Tue Mar 11 14:17:04 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 9402
    substate = 20
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
        bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 999.xxx.yyy
    queue_rank = 3
    queue_type = E
    comment = Job started on Tue Mar 11 at 14:17
    exit_status = 271
    submit_args = test.sh
    start_time = Tue Mar 11 14:17:04 2008
    start_count = 1
    checkpoint_dir = /var/spool/torque/checkpoint/999.xxx.yyy.CK
    checkpoint_name = ckpt.999.xxx.yyy.1205266630
```

Test 2 - Persistance of checkpoint images

Introduction

This test determines if the checkpoint files remain in the default directory after the job is removed from the Torque queue.

Note that this behavior was requested by a customer but in fact may not be the right thing to do as it leaves the checkpoint files on the execution node. These will gradually build up over time on the node being limited only by disk space. The right thing would seem to be that the checkpoint files are copied to the users home directory after the job is purged from the execution node.

Test Steps

Assuming the steps of Test 1, delete the job and then wait until the job leaves the queue after the completed job hold time. Then look at the contents of the default checkpoint directory to see if the files are still there.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qdel 999
> sleep 100
> qstat
>
> find /var/spool/torque/checkpoint
... files ...
```

Possible Failures

The files are not there, did Test 1 actually pass?

Successful Results

The files are there.

Test 3 - Restart after checkpoint

Introduction

This test determines if the job can be restarted after a checkpoint hold.

Test Steps

Assuming the steps of Test 1, issue a **qrls** command. Have another window open into the **/var/spool/torque/spool** directory and tail the job.

Successful Results

After the **qrls**, the job's output should resume.

Test 4 - Multiple checkpoint / restart

Introduction

This test determines if the checkpoint/restart cycle can be repeated multiple times.

Test Steps

Start a job and then while **tail**'ing the job output, do multiple **qhold**/**qrls** operations.

```
> qsub -c enabled test.sh
999.xxx.yyy
> qhold 999
> qrls 999
> qhold 999
> qrls 999
> qhold 999
> qrls 999
```

Successful Results

After each **qrls**, the job's output should resume. Also tried "while true; do qrls 999; qhold 999; done" and this seemed to work as well.

Test 5 - Periodic checkpoint

Introduction

This test determines if automatic periodic checkpoint will work.

Test Steps

Start the job with the option **-c enabled,periodic,interval=1** and look in the checkpoint directory for checkpoint images to be generated about every minute.

```
> qsub -c enabled,periodic,interval=1 test.sh
999.xxx.yyy
```

Successful Results

The checkpoint directory should contain multiple checkpoint images and the time on the files should be roughly a minute apart.

Test 6 - Restart from previous image

Introduction

This test determines if the job can be restarted from a previous checkpoint image.

Test Steps

Start the job with the option **-c enabled,periodic,interval=1** and look in the checkpoint directory for checkpoint images to be generated about every minute. Do a **qhold** on the job to stop it. Change the attribute checkpoint_name with the **qalter** command. Then do a **qrls** to restart the job.

```
> qsub -c enabled,periodic,interval=1 test.sh
999.xxx.yyy
> qhold 999
> qalter -W checkpoint_name=ckpt.999.xxx.yyy.1234567
> qrls 999
```

Successful Results

The job output file should be truncated back and the count should resume at an earlier number.

2.7 Job Exit Status

Once a job under TORQUE has completed, the `exit_status` attribute will contain the result code returned by the job script. This attribute can be seen by submitting a **qstat -f** command to show the entire set of information associated with a job. The `exit_status` field is found near the bottom of the set of output lines.

qstat -f (job failure example)

```
Job Id: 179.host
    Job_Name = STDIN
    Job_Owner = user@host
    job_state = C
    queue = batchq
    server = host
    Checkpoint = u
    ctime = Fri Aug 29 14:55:55 2008
    Error_Path = host:/opt/moab/STDIN.e179
    exec_host = node1/0
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Fri Aug 29 14:55:55 2008
    Output_Path = host:/opt/moab/STDIN.o179
    Priority = 0
    qtime = Fri Aug 29 14:55:55 2008
    Rerunable = True
    Resource_List.ncpus = 2
    Resource_List.nodect = 1
    Resource_List.nodes = node1
    Variable_List = PBS_O_HOME=/home/user,PBS_O_LOGNAME=user,

PBS_O_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:,PBS_O_SHE
LL=/bin/bash,PBS_O_HOST=host,
  PBS_O_WORKDIR=/opt/moab,PBS_O_QUEUE=batchq
    sched_hint = Post job file processing error; job 179.host on host
node1/0Ba
  d UID for job execution REJHOST=pala.cridomain MSG=cannot find user 'user'
in password file
    etime = Fri Aug 29 14:55:55 2008
    exit_status = -1
```

This code can be useful in diagnosing problems with jobs that may have unexpectedly terminated.

If TORQUE was unable to start the job, this field will contain a negative number produced by the pbs_mom.

Otherwise, if the job script was successfully started, the value in this field will be the return value of the script.

TORQUE Supplied Exit Codes

Name	Value	Description
JOB_EXEC_OK	0	job exec successful
JOB_EXEC_FAIL1	-1	job exec failed, before files, no retry
JOB_EXEC_FAIL2	-2	job exec failed, after files, no retry
JOB_EXEC_RETRY	-3	job execution failed, do retry
JOB_EXEC_INITABT	-4	job aborted on MOM initialization
JOB_EXEC_INITRST	-5	job aborted on MOM init, chkpt, no migrate
JOB_EXEC_INITRMG	-6	job aborted on MOM init, chkpt, ok migrate
JOB_EXEC_BADRESRT	-7	job restart failed
JOB_EXEC_CMDFAIL	-8	exec() of user command failed

Example of exit code from C program

```
$ cat error.c

#include
#include


int
main(int argc, char *argv)
```

```
{
  exit(256+11);
}


$ gcc -o error error.c


$ echo ./error | qsub
180.xxx.yyy


$ qstat -f
Job Id: 180.xxx.yyy
    Job_Name = STDIN
    Job_Owner = test.xxx.yyy
    resources_used.cput = 00:00:00
    resources_used.mem = 0kb
    resources_used.vmem = 0kb
    resources_used.walltime = 00:00:00
    job_state = C
    queue = batch
    server = xxx.yyy
    Checkpoint = u
    ctime = Wed Apr 30 11:29:37 2008
    Error_Path = xxx.yyy:/home/test/STDIN.e180
    exec_host = node01/0
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Wed Apr 30 11:29:37 2008
    Output_Path = xxx.yyy:/home/test/STDIN.o180
    Priority = 0
    qtime = Wed Apr 30 11:29:37 2008
    Rerunable = True
    Resource_List.neednodes = 1
    Resource_List.nodect = 1
    Resource_List.nodes = 1
    Resource_List.walltime = 01:00:00
    session_id = 14107
    substate = 59
    Variable_List = PBS_O_HOME=/home/test,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=test,
        PBS_O_PATH=/usr/local/perltests/bin:/home/test/bin:/usr/local/s
        bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games,
        PBS_O_SHELL=/bin/bash,PBS_SERVER=xxx.yyy,
        PBS_O_HOST=xxx.yyy,PBS_O_WORKDIR=/home/test,
        PBS_O_QUEUE=batch
    euser = test
    egroup = test
    hashname = 180.xxx.yyy
    queue_rank = 8
    queue_type = E
    comment = Job started on Wed Apr 30 at 11:29
    etime = Wed Apr 30 11:29:37 2008
    exit_status = 11
    start_time = Wed Apr 30 11:29:37 2008
    start_count = 1
```

Notice that the C routine **exit** passes only the low order byte of its argument. In this case, 256+11 is really 267 but the resulting exit code is only 11 as seen in the output.

3.0 Managing Nodes

3.1 Adding Nodes

TORQUE can add and remove nodes either dynamically with **qmgr** or by manually editing the **$TORQUEHOME/server_priv/nodes** file. (See Basic Configuration).

3.1.1 Run-Time Node Changes

TORQUE can dynamically add nodes with the **qmgr** command. For example, the following command will add node `node003`:

```
> qmgr -c "create node node003"
```
This will append the file **$TORQUE_HOME/server_priv/nodes** with:

```
node003
```
Nodes can also be removed with a similar command:

```
> qmgr -c "delete node node003"
```

Typically, an administrator will want to change the state of a node instead of remove it. (See Changing Node State.)

As of TORQUE 2.0, this capability was unreliable. It is highly recommended that node changes be followed by a restart of pbs_server, or just edit the nodes file manually and restart it.

3.2 Nodes Properties

TORQUE can associate properties with nodes to aid in identifying groups of nodes. It's typical for a site to conglomerate a heterogeneous sets of resources. To identify the different sets, properties can be given to each node in a set. For example, a group of nodes that has a higher speed network connection could have the property "ib". TORQUE can set, update, or remove properties either dynamically with **qmgr** or by manually editing the **nodes** file.

3.2.1 Run-Time Node Changes

TORQUE can dynamically change the properties of a node with the **qmgr** command. For example, note the following to give `node001` the properties of `bigmem` and `dualcore`:

```
> qmgr -c "set node node001 properties = bigmem"
> qmgr -c "set node node001 properties += dualcore"
```

To relinquish a stated property, use the "-=" operator.

3.2.2 Manual Node Changes

The properties of each node are enumerated in **$TORQUEHOME/server_priv/nodes**. The feature(s) must be in a space delimited list after the node name. For example, to give `node001` the properties of `bigmem` and `dualcore` and `node002` the properties of `bigmem` and `matlab`, edit the **nodes** file to contain the following:

server_priv/nodes

```
node001 bigmem dualcore
node002 np=4 bigmem matlab
```

For changes to the **nodes** file to be activated, **pbs_server** must be restarted.

For a full description of this file, please see the **PBS Administrator Guide**.

See Also:

- 2.1 Job Submission for specifying nodes properties for submitted jobs.

3.3 Changing Node State

A common task is to prevent jobs from running on a particular node by marking it **offline** with **pbsnodes -o nodename**. Once a node has been marked offline, the scheduler will no longer consider it available for new jobs. Simply use **pbsnodes -c nodename** when the node is returned to service.

Also useful is **pbsnodes -l** which lists all nodes with an *interesting* state, such as down, unknown, or offline. This provides a quick glance at nodes that migth be having a problem.

See the **pbsnodes** manpage for details.

3.4 Host Security

For systems requiring dedicated access to compute nodes (for example, users with sensitive data), TORQUE prologue and epilogue scripts provide a vehicle to leverage the authenication provided by linux-PAM modules. (See Appendix G Prologue and Epilogue Scripts for more information.)

To allow only users with running jobs (and root) to access compute nodes, do the following:

- Untar contrib/pam_authuser.tar.gz (found in the src tar ball).
- Compile pam_authuser.c with **make** and **make install** on every compute node.
- Edit /etc/system-auth as described in README.pam_authuser, again on every compute node.
- Either make a tarball of the epilogue* and prologue* scripts (to preserve the symbolic link) and untar it in the mom_priv directory, or just copy epilogue* and prologue* to mom_priv/.

The prologue* scripts are Perl scripts that add the user of the job to /etc/authuser. The epilogue* scripts then remove the first occurrence of the user from /etc/authuser. File locking is employed in all scripts to eliminate the chance of race conditions. Also, in the epilogue* scripts, there is code that is commented out that when activated kills all processes owned by the user (using pkill), when that user does not have another valid job on the same node.

prologue* and epilogue* scripts were added to the pam_authuser tarball in version 2.1 of TORQUE.

3.5 Linux Cpuset Support

3.5.1 Cpuset Overview

Linux kernel 2.6 Cpusets are logical, hierarchical groupings of CPUs and units of memory. Once created, individual processes can be placed within a cpuset. The processes will only be allowed to run/access the specified CPUs and memory. Cpusets are managed in a virtual file system mounted at /dev/cpuset. New cpusets are created by simply making new directories. Cpusets gain CPUs and memory units by simply writing the unit number to files within the cpuset.

TORQUE Cpuset support is new in 2.3.x and should be regarded as experimental and under development.

3.5.2 Cpuset Support

When started, pbs_mom will create an initial top-level cpuset at /dev/cpuset/torque. This cpuset contains all CPUs and memory of the host machine. If this "torqueset" already exists, it will be left unchanged to allow the administrator to override the default behavior. All subsequent cpusets are created within the torqueset.

When a job is started, the jobset is created at /dev/cpuset/torque/$jobid and populated with the CPUs listed in the exec_host job attribute. Also created are individual tasksets for each CPU within the jobset. This happens before prologue, which allows it to be easily modified, and it happens on all nodes.

The top-level batch script process is executed in the jobset. Tasks launched through the TM interface (pbsdsh and PW's mpiexec) will be executed within the appropriate taskset.

On job exit, all tasksets and the jobset are deleted.

3.5.3 Cpuset Configuration

At the moment, there are no run-time configurations. The support is disabled by default at build-time. Run configure with `--enable-cpuset` if you would like to test the code.

If enabled and run on a machine without cpuset support, pbs_mom will simply ignore it and will not complain.

A run-time pbs_mom boolean needs to be created to enable/disable it.

On the Linux host, the virtual file system must be mounted:

```
mount -t cpuset none /dev/cpuset
```

3.5.4 Cpuset advantages / disadvantages

Presently, any job can request a single CPU and proceed to use everything available in the machine. This is occasionally done to circumvent policy, but most often is simply an error on the part of the user. Cpuset support will easily constrain the processes to not interfere with other jobs.

Jobs on larger NUMA systems may see a performance boost if jobs can be intelligently assigned to specific CPUs. Jobs may perform better if striped across physical processors, or contained within the fewest number of memory controllers.

TM tasks are constrained to a single core, thus a multi-threaded process could seriously suffer.

3.5.5 Cpuset TODO

- The code requires cleanup with correct error handling.
- No attempt is made to be "smart" about the CPU assignment. We need a mechanism to expose the physical topology to the scheduler, and let the scheduler assign CPUs. Currently, pbs_server assigns "subnodes" to jobs, which are analogous to CPUs.

Proposal: pbs_mom "stringifies" the topology in some unambiguous format to a new node attribute. Moab will be able to read this information, and probably have its own way of supplying/overriding this information in its own configuration. Moab can set a new job attribute with a string that specifies the CPUs to assign. Then pbs_mom will ignore exec_host and use this job attribute instead.

- Memory is not handled at all. All memory units are added to all jobsets and tasksets. So far, it is unclear how memory should be handled.

4.0 Setting Server Policies

4.1 Queue Configuration

- 4.1.1 Queue Attributes
- 4.1.2 Example Queue Configuration
- 4.1.3 Setting a Default Queue
- 4.1.4 Mapping a Queue to a Subset of Resources
- 4.1.5 Creating a Routing Queue

Under TORQUE, queue configuration is accomplished using the qmgr command. With this tool, the first step is to create the queue. This is accomplished using the **create** subcommand of **qmgr** as in the following example:

```
> qmgr -c "create queue batch queue_type=execution"
```

Once created, the queue must be configured to be operational. At a minimum, this includes setting the options **started** and **enabled**. Further configuration is possible using any combination of the attributes listed in what follows.

For boolean attributes, T, t, 1, Y, and y are all synonymous with true, and F, f, 0, N, and n all mean false.

For queue_type, E and R are synonymous with Execution and Routing.

4.1.1 Queue Attributes

parameter	format	description	example

acl_groups	<GROUP>[@<HOST>][+<USER>[@<HOST>]]...	specifies the list of groups which may submit jobs to the queue. If acl_group_enable is set to **true**, only users with a primary group listed in **acl_groups** may utilize the queue. **NOTE**: If the **PBSACLUSEGROUPLIST** variable is set in the **pbs_server** environment, acl_groups will be	qmgr
			`> qmgr -c "set queue batch acl_groups=staff"`
			`> qmgr -c "set queue batch acl_groups+=ops@h2"`
			`> qmgr -c "set queue batch acl_groups+=staff@h3"`
			NOTE: used in conjunction with **acl_group_enable**)

		check against all groups of which the job user is a member.	
acl_group_enable	<BOOLEAN>	if **TRUE**, constrains TORQUE to only allow jobs submitted from groups specified by the acl_groups parameter. **DEFAULT**: FALSE	`qmgr -c "set queue batch acl_group_enable=true"`
acl_group_sloppy	<BOOLEAN>	if **TRUE**, acl_groups will be checked against all groups of which the job user is a member. **DEFAULT**: FALSE	
acl_hosts	<HOST>[+<HOST>]...	specifies the list of hosts that may submit jobs to the queue	`qmgr -c "set queue batch acl_hosts=h1+h2+h3"` (NOTE: used in conjunction with **acl_host_enable**)
acl_host_enable	<BOOLEAN>	if TRUE, constrains TORQUE to only allow jobs submitted from hosts specified by the **acl_hosts** parameter. **DEFAULT**: FALSE	`qmgr -c "set queue batch acl_host_enable=true"`
acl_logic_or	<BOOLEAN>	if **TRUE**, user and group acls are logically OR'd together, meaning that either acl may be met to allow access. If false or unset, then both acls are AND'd, meaning that both acls must be satisfied. **DEFAULT**: FALSE	

acl_users	<USER>[@<HOST>][+<USER>[@<HOST>]]...	specifies the list of users who may submit jobs to the queue. If acl_user_enable is set to **TRUE**, only users listed in	`qmgr`
			
			`> qmgr -c "set queue batch acl_users=john"`
			`> qmgr -c "set queue batch`

		acl_users may use the queue	`acl_users+=steve@h2"`
			`> qmgr -c "set queue batch acl_users+=stevek@h3"`
			NOTE: used in conjunction with **acl_user_enable**

acl_user_enable	<BOOLEAN>	if **TRUE**, constrains TORQUE to only allow jobs submitted from users specified by the acl_users parameter. **DEFAULT**: FALSE	`qmgr -c "set queue batch acl_user_enable=true"`
enabled	<BOOLEAN>	specifies whether the queue accepts new job submissions. **DEFAULT**: FALSE	`qmgr -c "set queue batch enabled=true"`
keep_completed	<INTEGER>	specifies the number of seconds jobs should be held in the Completed state after exiting. **DEFAULT**: 0	
kill_delay	<INTEGER>	specifies the number of seconds between sending a SIGTERM and a SIGKILL to a job being cancelled. **DEFAULT**: 2 seconds	`qmgr -c "set queue batch kill_delay=30"`
max_queuable	<INTEGER>	specifies the maximum number of jobs allowed in the queue at any given time (includes idle, running, and blocked jobs). **DEFAULT**: unlimited	`qmgr -c "set queue batch max_queuable=20"`
max_running	<INTEGER>	specifies the maximum number of jobs in the queue allowed to run at any given time. **DEFAULT**: unlimited	`qmgr -c "set queue batch max_running=20"`

max_user_queuable	<INTEGER>	specifies the maximum number of jobs, per user, allowed in the queue at any given time (includes idle, running, and blocked jobs). **DEFAULT**: unlimited. Version 2.1.3 and greater.	`qmgr -c "set queue batch max_user_queuable=20"`
max_user_run	<INTEGER>	specifies the maximum number of jobs, per user, in the queue allowed to run at any given time. **DEFAULT**: unlimited	`qmgr -c "set queue batch max_user_run=10"`
priority	<INTEGER>	specifies the priority value associated with the queue. **DEFAULT**: 0	`qmgr -c "set queue batch priority=20"`
queue_type	one of **e**, **execution**, **r**, or route	specifies the queue type. **NOTE**: This value must be explicitly set for all queues.	`qmgr -c "set queue batch queue_type=execution"`
resources_available	<STRING>	specifies to cumulative resources available to all jobs running in the queue. **DEFAULT**: N/A	`qmgr -c "set queue batch resources_available.nodect=20"` (**NOTE:** pbs_server must be restarted for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and the server.resources_available.)
resources_default	<STRING>	specifies default resource requirements for jobs submitted to the queue. **DEFAULT**: N/A	`qmgr -c "set queue batch resources_default.walltime=3600"`
resources_max	<STRING>	specifies the maximum resource limits for jobs submitted to the queue. **DEFAULT**: N/A	`qmgr -c "set queue batch resources_max.nodect=16"`
resources_mi	<STRING>	specifies the	`qmgr -c "set queue`

n		minimum resource limits for jobs submitted to the queue. **DEFAULT**: N/A	batch resources_min.nodect=2 "
route_destinations	<queue>[@<host>][+<queue>[@<host>]]...	specifies the potential destination queues for jobs submitted to the associated routing queue. **NOTE**: This attribute is only valid for routing queues. **DEFAULT**: N/A	qmgr > qmgr -c "set queue route route_destinations=fast" > qmgr -c "set queue route route_destinations+=slow"
started	<BOOLEAN>	specifies whether jobs in the queue are allowed to execute. **DEFAULT**: FALSE	qmgr -c "set queue batch started=true"

Resources may include one or more of the following: **arch**, **mem**, **nodes**, **ncpus**, **nodect**, **pvmem**, and **walltime**

4.1.2 Example Queue Configuration

The following series of **qmgr** commands will create and configure a queue named `batch`:

```
qmgr -c "create queue batch queue_type=execution"
qmgr -c "set queue batch started=true"
qmgr -c "set queue batch enabled=true"
qmgr -c "set queue batch resources_default.nodes=1"
qmgr -c "set queue batch resources_default.walltime=3600"
```

This queue will accept new jobs and, if not explicitly specified in the job, will assign a nodecount of 1 and a walltime of 1 hour to each job.

4.1.3 Setting a Default Queue

By default, a job must explicitly specify which queue it is to run in. To change this behavior, the server parameter default_queue may be specified as in the following example:

```
qmgr -c "set server default_queue=batch"
```

4.1.4 Mapping a Queue to a Subset of Resources

TORQUE does not currently provide a simple mechanism for mapping queues to nodes. However, schedulers such as Moab and Maui can provide this functionality.

The simplest method is using default_resources.neednodes on an execution queue, setting it to a particular node attribute. Maui/Moab will use this information to ensure that jobs in that queue will be assigned nodes with that attribute. For example, suppose we have some nodes bought with money from the chemistry department, and some nodes paid by the biology department.

```
$TORQUE_HOME/server_priv/nodes:
node01 np=2 chem
node02 np=2 chem
node03 np=2 bio
node04 np=2 bio

qmgr:
set queue chem resources_default.neednodes=chem
set queue bio  resources_default.neednodes=bio
```

This example does not preclude other queues from accessing those nodes. One solution is to use some other generic attribute with all other nodes and queues.

More advanced configurations can be made with standing reservations and QOSes.

4.1.5 Creating a Routing Queue

A routing queue will *steer* a job to a destination queue based on job attributes and queue constraints. It is set up by creating a queue of queue_type Route with a route_destinations attribute set, as in the following example.

qmgr

```
# routing queue
create queue route
set queue route queue_type = Route
set queue route route_destinations = reg_64
set queue route route_destinations += reg_32
set queue route route_destinations += reg
set queue route enabled = True
set queue route started = True

# queue for jobs using 1-15 nodes
create queue reg
set queue reg queue_type = Execution
set queue reg resources_min.ncpus = 1
set queue reg resources_min.nodect = 1
set queue reg resources_default.ncpus = 1
set queue reg resources_default.nodes = 1
set queue reg enabled = True
set queue reg started = True
```

```
# queue for jobs using 16-31 nodes
create queue reg_32
set queue reg_32 queue_type = Execution
set queue reg_32 resources_min.ncpus = 31
set queue reg_32 resources_min.nodes = 16
set queue reg_32 resources_default.walltime = 12:00:00
set queue reg_32 enabled = True
set queue reg_32 started = True

# queue for jobs using 32+ nodes
create queue reg_64
set queue reg_64 queue_type = Execution
set queue reg_64 resources_min.ncpus = 63
set queue reg_64 resources_min.nodes = 32
set queue reg_64 resources_default.walltime = 06:00:00
set queue reg_64 enabled = True
set queue reg_64 started = True

# have all jobs go through the routing queue
set server default_queue = batch
set server resources_default.ncpus = 1
set server resources_default.walltime = 24:00:00
  ...
```

In this example, the compute nodes are dual processors and default walltimes are set according to the number of processors/nodes of a job. Jobs with 32 nodes (63 processors) or more will be given a default walltime of 6 hours. Also, jobs with 16-31 nodes (31-62 processors) will be given a default walltime of 12 hours. All other jobs will have the server default walltime of 24 hours.

The ordering of the route_destinations is important. In a routing queue, a job is assigned to the first possible destination queue based on the resources_max, resources_min, acl_users, and acl_groups attributes. In the preceding example, the attributes of a single processor job would first be checked against the reg_64 queue, then the reg_32 queue, and finally the reg queue.

Adding the following settings to the earlier configuration elucidates the queue resource requirements:

qmgr

```
set queue reg resources_max.ncpus = 30
set queue reg resources_max.nodect = 15

set queue reg_16 resources_max.ncpus = 62
set queue reg_16 resources_max.ncpus = 31
```

The time of enforcement of server and queue defaults is important in this example. TORQUE applies server and queue defaults differently in job centric and queue centric modes. For job centric mode, TORQUE waits to apply the server and queue defaults until the job is assigned to its final execution queue. For queue centric mode, it enforces server defaults before it is placed in the routing queue. In either mode, queue defaults override the server defaults. TORQUE defaults to job centric mode. To set queue centric mode, set queue_centric_limits, as in what follows:

qmgr

```
set server queue_centric_limits = true
```

An artifact of job centric mode is that if a job does not have an attribute set, the server and routing queue defaults are not applied when queue resource limits are checked. Consequently, a job that requests 32 nodes (not ncpus=32) will not be checked against a min_resource.ncpus limit. Also, for the preceding example, a job without any attributes set will be placed in the reg_64 queue, since the server ncpus default will be applied after the job is assigned to an execution queue.

Routine queue defaults are NOT applied to job attributes in versions 2.1.0 and before.

If the error message '`qsub: Job rejected by all possible destinations`' is reported when submitting a job, it may be necessary to add queue location information, (i.e., in the routing queue's route_destinations attribute, change '`batch`' to '`batch@localhost`').

See Also

- Server Parameters
- qalter - command which can move jobs from one queue to another

4.2 Server High Availability

The option of running TORQUE in a redundant or high availability mode has been implemented. This means that there can be multiple instances of the server running and waiting to take over processing in the event that the currently running server fails.

The high availability feature is available in the 2.3 and 2.4(trunk) versions of TORQUE.

Multiple server host machines

Two server host machines can run **pbs_server** at the same time. The two servers have their **torque/server_priv** directory mounted on a shared NFS file system. The **pbs_server** needs to be started with the **--ha** command line option that allows two servers to run at the same time. Only the first server to start will complete the full startup. The second server to start will block very early in the startup when it tries to lock the file **torque/server_priv/server.lock**. When the second server cannot obtain the lock, it will spin in a loop and wait for the lock to clear. The sleep time between checks of the lock file is one second.

Notice that not only can the servers run on independent server hardware, there can also be multiple instances of the **pbs_server** running on the same machine. This was not possible before as the second one to start would always write an error and quit when it could not obtain the lock.

How commands select the correct server host

The various commands that send messages to **pbs_server** usually have an option of specifying the server name on the command line, or if none is specified will use the default server name. The default server name comes either from the environment variable **PBS_DEFAULT** or from the file **torque/server_name**.

The definition of the contents of the file **torque/server_name** has been extended to allow this specification to be a comma-separated list of server names.

When a command is executed and no explicit server is mentioned, an attempt is made to connect to the first server name in the list. If this fails, then the second server name is tried. If both servers are unreachable, an error is returned and the command fails.

Note that there is a period of time after the failure of the current server during which the new server is starting up where it is unable to process commands. The new server must read the existing configuration and job infromation from the disk and so the length of this time is based on the size of the disk based state information. Commands issued during this period of time might fail due to timeouts expiring.

Job names

One aspect of this enhancement is in the construction of job names. Job names normally contain the name of the host machine where **pbs_server** is running. When job names are constructed, only the first name from the server specification list is used in building the job name.

Persistence of the pbs_server process

The system adminstrator must ensure that **pbs_server** continues to run on the server nodes. This could be as simple as a **cron** job that counts the number of **pbs_server**'s in the process table and starts some more if needed.

High availability of the NFS server

One consideration of this implemention is that it depends on NFS file system also being redundant. NFS can be set up as a redundant service. See the following.

- Setting Up A Highly Available NFS Server
- Making NFS Work On Your Network
- Sourceforge Linux NFS FAQ
- NFS v4 main site

There are also other ways to set up a shared file system. See the following.

- Red Hat Global File System
- Data sharing with a GFS storage cluster

Example configuration

The following section describes the test setup used to verify the operation of this new feature. Three machines constitute the desktop machine: jakaa, and two lab machines, node12 and node13, where the **pbs_server**'s were resident. Commands were submitted from the desktop machine, jakaa. This machine also ran **pbs_mom** and **pbs_sched**. It was also the NFS server for the shared **server_priv** directory.

The NFS setup on jakaa is shown in what follows. This setup allows the entire TORQUE directory structure to be shared although the server machines only shared the **server_priv** part of the share.

NFS exports

```
root@jakaa:/etc# cat exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync) hostname2(ro,sync)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt)
# /srv/nfs4/homes  gss/krb5i(rw,sync)
#
/tmp            *.cridomain(rw,sync)
/var/spool/torque               *.cridomain(rw,sync,no_root_squash)
```

Next is shown the fstab file that describes how the mounts were done on node12 and node13.

FSTAB file contents

```
root@node12:/var/spool/torque/mom_priv# cat /etc/fstab
# /etc/fstab: static file system information.
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
/dev/hda1       /               ext3    defaults,errors=remount-ro 0       1
/dev/hda5       none            swap    sw              0       0
jakaa:/var/spool/torque/server_priv      /var/spool/torque/server_priv
nfs     bg,intr,soft,rw        0       0
```

Nodes 12 and 13 each had their own /var/spool/torque directories. The NFS mount just replaced the server_priv subdirectory with the shared one on jakaa. As far as the local configuration one nodes 12 and 13, the only thing done was to set up the server_name file.

Contents of the torque directory

```
root@node12:/var/spool/torque# ll
total 56
drwxr-xr-x 12 root root 4096 2007-11-28 15:40 ./
drwxr-xr-x  4 root root 4096 2007-12-05 10:14 ../
drwxr-xr-x  2 root root 4096 2007-12-04 15:24 aux/
drwx------  2 root root 4096 2007-11-28 15:40 checkpoint/
```

```
drwxr-xr-x  2 root root 4096 2007-12-06 00:01 mom_logs/
drwxr-x--x  3 root root 4096 2007-11-30 11:22 mom_priv/
-rw-r--r--  1 root root   36 2007-11-28 15:40 pbs_environment
drwxr-xr-x  2 root root 4096 2007-12-04 10:26 sched_logs/
drwxr-x---  3 root root 4096 2007-12-07 10:08 sched_priv/
drwxr-xr-x  2 root root 4096 2007-12-07 00:07 server_logs/
-rw-r--r--  1 root root   34 2007-12-06 11:33 server_name
drwxr-x--- 12 root root 4096 2007-12-07 11:38 server_priv/
drwxrwxrwt  2 root root 4096 2007-12-05 15:40 spool/
drwxrwxrwt  2 root root 4096 2007-12-04 12:47 undelivered/

root@node12:/var/spool/torque# cat server_name
node12.cridomain,node13.cridomain
```

File torque/server_name

```
root@jakaa:/var/spool/torque# cat server_name
node12,node13
```

The following shows the setup of the batch queue.

PBS server configuration

```
root@node12:/var/spool/torque# qmgr -c 'p s'
#
# Create queues and set their attributes.
#
#
# Create and define queue batch
#
create queue batch
set queue batch queue_type = Execution
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch resources_available.nodect = 999999
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server acl_host_enable = True
set server acl_hosts = jakaa.cridomain
set server acl_hosts += node13.cridomain
set server acl_hosts += node12.cridomain
set server acl_hosts += jakaa
set server managers = root@node12.cridomain
set server managers += ssnelgrove@jakaa.cridomain
set server managers += ssnelgrove@node12.cridomain
set server operators = root@node12.cridomain
set server operators += ssnelgrove@node12.cridomain
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server scheduler_iteration = 600
set server node_check_rate = 150
set server tcp_timeout = 6
set server mom_job_sync = True
set server pbs_version = 2.2.2
set server keep_completed = 300
set server submit_hosts = jakaa
```

Note that in this setup, node12 and node13 had to be explicitly added to the acl_hosts lists. This requirement was removed by having the code automatically add the names in the server_name file to the acl_hosts list.

The following shows the processes running on all of the machines.

Running instances of PBS processes

```
root@jakaa:/var/spool/torque# ps -ef|grep pbs
root     11548     1  0 11:34 ?        00:00:00 pbs_mom
root     19305     1  0 11:42 ?        00:00:00 pbs_sched

root@node12:/var/spool/torque# ps -ef|grep pbs
root      4992     1  0 11:38 ?        00:00:00 pbs_server --ha

root@node12:/var/spool/torque# ps -ef|grep pbs
root     15190     1  0 13:11 pts/0    00:00:00 pbs_server --ha
```

5.0 Interfacing with a Scheduler

5.1 Integrating Schedulers for TORQUE

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default TORQUE scheduler, **pbs_sched**, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager are highly recommended. If using Maui/Moab, refer to the Moab-PBS Integration Guide. If using **pbs_sched**, start this daemon now.

If you are installing ClusterSuite, TORQUE and Moab were configured at installation for interoperability and no further action is required.

6.0 Configuring Data Management

6.1 SCP / RCP Setup

To use *scp* based data management, TORQUE must be authorized to migrate data to any of the compute nodes. If this is not already enabled within the cluster, this can be achieved with the process described below. This process enables uni-directional access for a particular user from a *source* host to a *destination* host.

These directions were written using OpenSSH version 3.6 and may not transfer correctly to older versions.

6.1.1 - Generate SSH Key on Source Host

On the source host as the transfer user, execute the following:

```
>  ssh-keygen -t rsa
```

This will prompt for a passphrase (optional) and create two files: *id_rsa* and *id_rsa.pub* inside ~/.ssh/.

6.1.2 - Copy Public SSH Key to Each Destination Host

Transfer public key to each destination host as the transfer user:

Easy Key Copy

```
ssh-copy-id [-i [identity_file]] [user@]machine
```

Manual Steps to Copy Keys

```
> scp ~/.ssh/id_rsa.pub destHost:~ (enter password)
```

Create an *authorized_keys* file on each destination host.

```
> ssh destHost (enter password)
> cat id_rsa.pub >> .ssh/authorized_keys
```

(If the *.ssh* directory does not exist, create it with 700 privileges (`mkdir .ssh;chmod 700 .ssh`)

```
> chmod 600 .ssh/authorized_keys

> rm id_rsa.pub
```

6.1.3 - Configure the SSH Daemon on Each Destination Host

Some configuration of the ssh daemon may be required on the *destination* host. (Because this is not always the case, skip to step 4 and test the changes made to this point. If the tests fail, proceed with this step and then try testing again.) Typically, this is done by editing the `/etc/ssh/sshd_config` file (root access needed). To verify correct configuration, see that the following attributes are set (not commented):

```
RSAAuthentication    yes
PubkeyAuthentication yes
```

If configuration changes were required, the ssh daemon will need to be restarted (root access needed):

```
> /etc/init.d/sshd restart
```

6.1.4 - Validating Correct SSH Configuration

If all is properly configured, the following command issued on the *source* host should succeed and not prompt for a password:

```
> scp destHost:/etc/motd /tmp
```

Note that if this is your first time accessing *destination* from *source*, it may ask you if you want to add the fingerprint to a file of known hosts. If you specify yes, this message should no longer appear and should not interfere with scp copying via TORQUE. Also, it is important that the full hostname appear in the known_hosts file. To do this, use the full hostname for *destHost*, as in `machine.domain.org` instead of just `machine`.

6.1.5 - Enabling Bi-Directional SCP Access

The preceding steps allow *source* access to *destination* without prompting for a password. The reverse, however, is not true. Repeat the steps, but this time using the *destination* as the *source*, etc. to enable bi-directional SCP access (i.e. *source* can send to *destination* and *destination* can send to *source* without password prompts.)

6.1.6 - Compile TORQUE to Support SCP

TORQUE must be re-configured (and then rebuilt) to use SCP by passing in the `--with-scp` flag to the configure script:

```
> ./configure --prefix=xxx --with-scp
> make
```

If special scp flags are required in your local setup, these can be specified using the rcpcmd parameter.

Troubleshooting

If, after following all of these steps, TORQUE is still having problems transferring data with scp, set the `PBSDEBUG` environment variable and restart the `pbs_mom` for details about copying. Also check the MOM log files for more details.

6.2 NFS and Other Networked Filesystems

6.2.1 TORQUE Data Management

When a batch job starts, its stdin file (if specified) is copied from the submission directory on the remote submission host. This file is placed in the `$PBSMOMHOME` directory on the mother superior node (i.e., `/usr/spool/PBS/spool`). As the job runs, stdout and stderr files are generated and placed in this directory using the naming convention **$JOBID.OU** and **$JOBID.ER**.

When the job completes, the MOM copies the files into the directory from which the job was submitted. By default, this file copying will be accomplished using a remote copy facility such as **rcp** or **scp**.

If a shared file system such as NFS, DFS, or AFS is available, a site can specify that the MOM should take advantage of this by specifying the `$usecp` directive inside the MOM configuration file (located in the `$PBSMOMHOME/mom_priv` directory) using the following format **$usecp <HOST>:<SRCDIR> <DSTDIR>**

`HOST` can be specified with a leading wildcard ('*****') character. The following example demonstrates this directive:

mom_priv/config

```
# /home is NFS mounted on all hosts

$usecp *:/home  /home

# submission hosts in domain fte.com should map '/data' directory on submit
host to
# '/usr/local/data' on compute host

$usecp *.fte.com:/data  /usr/local/data
```

If for any reason the MOM daemon is unable to copy the output or error files to the submission directory, these files are instead copied to the `undelivered` directory also located in `$PBSMOMHOME`.

6.3 File Stage-In/Stage-Out

File staging requirements are specified using the **stagein** and **stageout** directives of the qsub command. Stagein requests occur before the job starts execution, while stageout requests happen after a job completes.

On completion of the job, all staged-in and staged-out files are removed from the execution system. The **file_list** is in the form **local_file@hostname:remote_file[,...]** regardless of the direction of the copy. The name **local_file** is the name of the file on the system where the job executed. 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. The use of wildcards in the file name is not recommended.

The file names map to a remote copy program (rcp/scp/cp, depending on configuration) called on the execution system in the following manner:

- For stagein: rcp/scp hostname:remote_file local_file
- For stageout: rcp/scp local_file hostname:remote_file

Examples

```
# stage input.txt from node13.fsc to /home/john/input on master compute node
> qsub -l nodes=1,walltime=100 -W
stagein=input.txt@node13.fsc:/home/john/input.txt
```

```
# stage /home/bill/output.txt on master compute node to /tmp/output.txt on
node15.fsc
> qsub -l nodes=1,walltime=100 -W
stageout=/tmp/output.txt@node15.fsc:/home/bill/output.txt
```

```
$ fortune >xxx;echo cat xxx|qsub -W stagein=xxx@`hostname`:xxx
199.myhost.mydomain
$ cat STDIN*199
Anyone who has had a bull by the tail knows five or six more things
than someone who hasn't.
                -- Mark Twain
```

7.0 Interfacing with Message Passing

7.1 MPI (Message Passing Interface) Support

7.1.1 MPI (Message Passing Interface) Overview

A message passing library is used by parallel jobs to augment communication between the tasks distributed across the cluster. TORQUE can run with any message passing library and provides limited integration with some MPI libraries.

7.1.2 MPICH

One of the most popular MPI libraries is MPICH available from Argonne National Lab. If using this release, you may want to consider also using the mpiexec tool for launching MPI applications. Support for **mpiexec** has been integrated into TORQUE.

MPIExec Overview

mpiexec is a replacement program for the script **mpirun**, which is part of the **mpich** package. It is used to initialize a parallel job from within a PBS batch or interactive environment. **mpiexec** uses the task manager library of PBS to spawn copies of the executable on the nodes in a PBS allocation.

Reasons to use **mpiexec** rather than a script (mpirun) or an external daemon (mpd):

- Starting tasks with the TM interface is much faster than invoking a separate rsh * once for each process.
- Resources used by the spawned processes are accounted correctly with mpiexec, and reported in the PBS logs, because all the processes of a parallel job remain under the control of PBS, unlike when using mpirun-like scripts.
- Tasks that exceed their assigned limits of CPU time, wallclock time, memory usage, or disk space are killed cleanly by PBS. It is quite hard for processes to escape control of the resource manager when using mpiexec.
- You can use mpiexec to enforce a security policy. If all jobs are forced to spawn using mpiexec and the PBS execution environment, it is not necessary to enable rsh or ssh access to the compute nodes in the cluster.

See the mpiexec homepage for more information.

MPIExec Troubleshooting

Although problems with **mpiexec** are rare, if issues do occur, the following steps may be useful:

- determine current version using `mpiexec --version` and review the change log available on the MPI homepage to determine if the reported issue has already been corrected
- send email to the **mpiexec** mailing list at **mpiexec@osc.edu**
- browse the **mpiexec** user list archives for similar problems and resolutions
- read the FAQ contained in the **README** file and the mpiexec man pages contained within the **mpiexec** distribution
- increase the logging of **mpiexec** operation with **mpiexec --verbose** (reports messages to stderr)
- increase logging of the master and slave resource manager execution daemons associated with the job (with **TORQUE**, use `$loglevel` to 5 or higher in `$TORQUEROOT/mom_priv/config` and look for '**tm**' messages after associated `join job` messages).
- use **tracejob** (included with **TORQUE**) or **qtracejob** (included with OSC's **pbstools** package) to isolate failures within the cluster.
- if the message '`exec: Error: get_hosts: pbs_connect: Access from host not allowed, or unknown host`' appears, this indicates that **mpiexec** cannot communicate with the **pbs_server** daemon. In most cases, this indicates that the '`$TORQUEROOT/server_name`' file points to the wrong server or the node cannot resolve the server's name. The **qstat** command can be run on the node to test this.

General MPI Troubleshooting

When using MPICH, some sites have issues with orphaned MPI child processes remaining on the system after the master MPI process has been terminated. To address this, TORQUE epilogue scripts can be created that properly clean up the orphaned processes. Some sample scripts have been contributed and are accessible via the links in the following table.

Contributing Site	Description
Soton	Extracts of Perl Script

7.1.3 MPICH-VMI

MPICH-VMI is a highly-optimized open-source message passing layer available from NCSA. Additional information can be found in the VMI tutorial.

7.1.4 Open MPI

Open MPI is a new MPI implementation that combines technologies from multiple projects to create the best possible library. It supports the TM interface for intergration with TORQUE. Mmore inforamtion is available in the FAQ.

8.0 Managing Resources

8.1 Monitoring Resources

8.1.1 Resource Overview

A primary task of any resource manager is to monitor the state, health, configuration, and utilization of managed resources. TORQUE is specifically designed to monitor compute hosts for use in a batch environment. TORQUE is not designed to monitor non-compute host resources such as software licenses, networks, file systems, and so forth, although these resources can be integrated into the cluster using some scheduling systems.

With regard to monitoring compute nodes, TORQUE reports about a number of attributes broken into three major categories: (1) configuration, (2) utilization, and (3) state.

8.1.1.1 Configuration

Configuration includes both detected hardware configuration and specified batch attributes.

Attribute	Description	Details
Architecture (arch)	operating system of the node	the value reported is a derivative of the operating system installed
Node Features	arbitrary string attributes	no node features are specified by default. If required, they are set using the nodes file located in the

(properties)	associated with the node	$TORQUEHOME/server_priv directory. They may specify any string and are most commonly used to allow users to request certain subsets of nodes when submitting jobs.
Local Disk (size)	configured local disk	by default, local disk space is not monitored. If the mom configuration size parameter is set, TORQUE will report, in kilobytes, configured disk space within the specified directory.
Memory (physmem)	local memory/RAM	local memory/RAM is monitored and reported in kilobytes.
Processors (ncpus/np)	real/virtual processors	the number of processors detected by TORQUE is reported via the **ncpus** attribute. However, for scheduling purposes, other factors are taken into account. In its default configuration, TORQUE operates in **dedicated** mode with each node possessing a single virtual processor. In dedicated mode, each job task will consume one virtual processor and TORQUE will accept workload on each node until all virtual processors on that node are in use. While the number of virtual processors per node defaults to 1, this may be configured using the nodes file located in the $TORQUEHOME/server_priv directory. An alternative to dedicated mode is *timeshared* mode. If TORQUE's *timeshared* mode is enabled, TORQUE will accept additional workload on each node until the node's **maxload** limit is reached.
Swap (totmem)	virtual memory/Swap	virtual memory/Swap is monitored and reported in kilobytes.

8.1.1.2 Utilization

Utilization includes information regarding the amount of node resources currently available (in use) as well as information about who or what is consuming it.

Attribute	**Description**	**Details**
Disk (size)	local disk availability	by default, local disk space is not monitored. If the mom configuration size parameter is set, TORQUE will report configured and currently available disk space within the specified directory in kilobytes.
Memory (availmem)	real memory/RAM	available real memory/RAM is monitored and reported in kilobytes.
Network (netload)	local network adapter usage	reports total number of bytes transferred in or out by the network adapter
Processor Utilization (loadave)	node's cpu load average	reports the node's 1 minute bsd load average

8.1.1.3 State

State information includes administrative status, general node *health* information, and general usage status

Attribute	**Description**	**Details**
Idle Time	time since local	time in seconds since local keyboard/mouse activity has

(idletime)	keyboard/mouse activity has been detected	been detected
State (state)	monitored/admin node state	a node can be in one or more of the following states • **busy** - node is full and will not accept additional work • **down** - node is failing to report, is detecting local failures with node configuration or resources, or is marked down by and administrator • **free** - node is ready to accept additional work • **offline** - node has been instructed by an admin to no longer accept work • **unknown** - node has not been detected • **job-exclusive** - all available virtual processors are assigned to jobs

9.0 Accounting

9.1 Accounting Records

TORQUE maintains accounting records for batch jobs in the following directory:

$TORQUEROOT/server_priv/accounting/<TIMESTAMP>

`$TORQUEROOT` defaults to `/usr/spool/PBS` and <TIMESTAMP> is in the form YYYYMMDD. These records include events, timestamps, and information on resources requested and used.

Records for four different event types are produced and are described in the following table.

Record Marker	Record Type	Description
D	delete	job has been deleted
E	exit	job has exited (either successfully or unsuccessfully)
Q	queue	job has been submitted/queued
S	start	an attempt to start the job has been made (if the job fails to properly start, it may have multiple job start records)

Accounting Variables

The following table offers accounting variable descriptions. Descriptions for accounting variables not indicated in the table, particularly those prefixed with `Resources_List`, are available at 2.1 Job Submission.

Variable	Description
ctime	time job was created
etime	time job was queued
qtime	time job became eligible to run
start	time job started to run

A sample record in this file may look like the following:

```
06/06/2005 14:04:25;D;408.ign1.zeta2.org;requestor=guest@ign1.zeta2.org
06/06/2005 14:04:35;Q;409.ign1.zeta2.org;queue=batch
06/06/2005 14:04:44;Q;410.ign1.zeta2.org;queue=batch
06/06/2005 14:06:06;S;407.ign1.zeta2.org;user=guest group=guest jobname=STDIN
queue=batch ctime=1118087915 qtime=1118087915 etime=1118087915
start=1118088366
exec_host=ign1.zeta2.org/0 Resource_List.neednodes=ign1.zeta2.org
Resource_List.nodect=1 Resource_List.nodes=1 Resource_List.walltime=00:16:40
06/06/2005 14:07:17;D;407.ign1.zeta2.org;requestor=guest@ign1.zeta2.org
06/06/2005 14:07:17;E;407.ign1.zeta2.org;user=guest group=guest jobname=STDIN
queue=batch ctime=1118087915 qtime=1118087915 etime=1118087915
start=1118088366
exec_host=ign1.zeta2.org/0 Resource_List.nodect=1 Resource_List.nodes=1
Resource_List.walltime=00:16:40 session=6365 end=1118088437 Exit_status=271
resources_used.cput=00:00:00 resources_used.mem=3068kb
resources_used.vmem=16080kb
resources_used.walltime=00:01:11
```

10.0 Troubleshooting

10.1 Troubleshooting

There are a few general strategies that can be followed to determine unexpected behavior. These are a few of the tools available to help determine where problems occur.

- 10.1.1 Host Resolution
- 10.1.2 Firewall Configuration
- 10.1.3 TORQUE Log File
- 10.1.4 Using tracejob to Locate Job Failures
- 10.1.5 Using GDB to Locate Failures
- 10.1.6 Other Diagnostic Options
- 10.1.7 Frequently Asked Questions

10.1.1 Host Resolution

The TORQUE server host must be able to perform both forward and reverse name lookup on itself and on all compute nodes. Likewise, all compute nodes must be able to perform both forward and reverse name lookup on itself, the TORQUE server host, and all other compute nodes. In many cases, name resolution is handled by configuring the node's `/etc/hosts` file although **DNS** and **NIS** services may also be used. Commands such as **nslookup** or **dig** can be used to verify proper host resolution.

NOTE: Invalid host resolution may exhibit itself with compute nodes reporting as down within the output of pbsnodes -a and with failure of the momctl -d 3 command.

10.1.2 Firewall Configuration

Be sure that if you have firewalls running on the server or node machines that you allow connections on the appropriate ports for each machine. TORQUE pbs_mom daemons use UDP port 1023 and the pbs_server/pbs_mom daemons use ports 15001-15004 by default.

Firewall based issues are often associated with server to mom communication failures and messages such as 'premature end of message' in the log files.

Also, the `tcpdump` program can be used to verify the correct network packets are being sent.

10.1.3 TORQUE Log Files

The **pbs_server** keeps a daily log of all activity in the "<TORQUE_HOME_DIR>/server_logs/" directory. The **pbs_mom** also keeps a daily log of all activity in the "<TORQUE_HOME_DIR>/mom_logs/" directory. These logs contain information on communication between server and mom as well as information on jobs as they enter the queue and as they are dispatched, ran, and terminated. These logs can be very helpful in determining general job failures. For mom logs, the verbosity of the logging can be adjusted by setting the loglevel parameter in the `mom_priv/config` file. For server logs, the verbosity of the logging can be adjusted by setting the server log_level attribute in qmgr.

For both **pbs_mom** and **pbs_server** daemons, the log verbosity level can also be adjusted by setting the environment variable **PBSLOGLEVEL** to a value between 0 and 7. Further, to dynamically change the log level of a running daemon, use the *SIGUSR1* and *SIGUSR2* signals to increase and decrease the active loglevel by one. Signals are sent to a process using the **kill** command. For example, **kill -USR1 `pgrep pbs_mom`** would raise the log level up by one. The current loglevel for **pbs_mom** can be displayed with the command **momctl -d3**.

10.1.4 Using tracejob to Locate Job Failures

Overview

The **tracejob** utility extracts job status and job events from accounting records, mom log files, server log files, and scheduler log files. Using it can help identify where, how, a why a job failed. This tool takes a job id as a parameter as well as arguments to specify which logs to search, how far into the past to search, and other conditions.

Syntax

```
tracejob [-a|s|l|m|q|v|z] [-c count] [-w size] [-p path] [ -n <DAYS>] [-f
filter_type] <JOBID>

  -p : path to PBS_SERVER_HOME
  -w : number of columns of your terminal
  -n : number of days in the past to look for job(s) [default 1]
  -f : filter out types of log entries, multiple -f's can be specified
       error, system, admin, job, job_usage, security, sched, debug,
       debug2, or absolute numeric hex equivalent
  -z : toggle filtering excessive messages
  -c : what message count is considered excessive
  -a : don't use accounting log files
  -s : don't use server log files
  -l : don't use scheduler log files
  -m : don't use mom log files
  -q : quiet mode - hide all error messages
  -v : verbose mode - show more error messages
```

Example

```
> tracejob -n 10 1131

Job: 1131.icluster.org

03/02/2005 17:58:28  S    enqueuing into batch, state 1 hop 1
03/02/2005 17:58:28  S    Job Queued at request of dev@icluster.org, owner =
                          dev@icluster.org, job name = STDIN, queue = batch
03/02/2005 17:58:28  A    queue=batch
03/02/2005 17:58:41  S    Job Run at request of dev@icluster.org
03/02/2005 17:58:41  M    evaluating limits for job
03/02/2005 17:58:41  M    phase 2 of job launch successfully completed
03/02/2005 17:58:41  M    saving task (TMomFinalizeJob3)
03/02/2005 17:58:41  M    job successfully started
03/02/2005 17:58:41  M    job 1131.koa.icluster.org reported successful start
on 1 node(s)
03/02/2005 17:58:41  A    user=dev group=dev jobname=STDIN queue=batch
ctime=1109811508
                          qtime=1109811508 etime=1109811508 start=1109811521
                          exec_host=icluster.org/0 Resource_List.neednodes=1
Resource_List.nodect=1
                          Resource_List.nodes=1
Resource_List.walltime=00:01:40
03/02/2005 18:02:11  M    walltime 210 exceeded limit 100
03/02/2005 18:02:11  M    kill_job
03/02/2005 18:02:11  M    kill_job found a task to kill
03/02/2005 18:02:11  M    sending signal 15 to task
03/02/2005 18:02:11  M    kill_task: killing pid 14060 task 1 with sig 15
03/02/2005 18:02:11  M    kill_task: killing pid 14061 task 1 with sig 15
```

```
03/02/2005 18:02:11  M    kill_task: killing pid 14063 task 1 with sig 15
03/02/2005 18:02:11  M    kill_job done
03/02/2005 18:04:11  M    kill_job
03/02/2005 18:04:11  M    kill_job found a task to kill
03/02/2005 18:04:11  M    sending signal 15 to task
03/02/2005 18:06:27  M    kill_job
03/02/2005 18:06:27  M    kill_job done
03/02/2005 18:06:27  M    performing job clean-up
03/02/2005 18:06:27  A    user=dev group=dev jobname=STDIN queue=batch
ctime=1109811508
                          qtime=1109811508 etime=1109811508 start=1109811521
                          exec_host=icluster.org/0 Resource_List.neednodes=1
Resource_List.nodect=1
                          Resource_List.nodes=1
Resource_List.walltime=00:01:40 session=14060
                          end=1109811987 Exit_status=265
resources_used.cput=00:00:00
                          resources_used.mem=3544kb
resources_used.vmem=10632kb
                          resources_used.walltime=00:07:46

...
```

NOTE: The **tracejob** command operates by searching the pbs_server accounting records and the pbs_server, mom, and scheduler logs. To function properly, it must be run on a node and as a user which can access these files. By default, these files are all accessible by the user **root** and only available on the cluster *management* node. In particular, the files required by **tracejob** located in the following directories:

- $TORQUEHOME/server_priv/accounting
- $TORQUEHOME/server_logs
- $TORQUEHOME/mom_logs
- $TORQUEHOME/sched_logs

tracejob may only be used on systems where these files are made available. Non-root users may be able to use this command if the permissions on these directories or files is changed appropriately.

10.1.5 Using GDB to Locate Failures

If either the **pbs_mom** or **pbs_server** fail unexpectedly (and the log files contain no information on the failure) gdb can be used to determine whether or not the program is crashing. To start **pbs_mom** or **pbs_server** under GDB export the environment variable **PBSDEBUG=yes** and start the program (i.e., `gdb pbs_mom` and then issue the **run** subcommand at the gdb prompt). GDB may run for some time until a failure occurs and which point, a message will be printed to the screen and a gdb prompt again made available. If this occurs, use the gdb **where** subcommand to determine the exact location in the code. The information provided may be adequate to allow local diagnosis and correction. If not, this output may be sent to the mailing list or to help for further assistance. (for more information on submitting bugs or requests for help please see the Mailing List Instructions)

NOTE: See the PBSCOREDUMP parameter for enabling creation of core files.

10.1.6 Other Diagnostic Options

- when **PBSDEBUG** is set, some client commands will print additional diagnostic information.

```
$ export PBSDEBUG=yes
$ cmd
```

Some hard problems in Torque deal with the amount of time spent in routines. For example, one currently open problem appears to be caused by the design of the code in linux/mom_mach.c where the statistics are gathered for the node status. It appears that the **/proc** filesystem that contains information about the kernel and the processes is being accessed so often on some machines that the responces to some other message traffic is affected. The machine where this is happening has 128 processors.

To debug these kinds of problems, it can be useful to see where in the code time is being spent. This is called profiling and there is a linux utility **gprof** that will output a listing of routines and the amount of time spent in these routines. This does require that the code be compiled with special options to instrument the code and to produce a file, gmon.out, that will be written at the end of program execution.

The following listing shows how to build Torque with profiling enabled. Notice that the output file for pbs_mom will end up in the mom_priv directory because its startup code changes the default directory to this location.

```
# ./configure "CFLAGS=-pg -lgcov -fPIC"
# make -j5
# make install
# pbs_mom
... do some stuff for a while ...
# momctl -s
# cd /var/spool/torque/mom_priv
# gprof -b `which pbs_mom` gmon.out |less
#
```

Another way to see areas where a program is spending most of its time is with the valgrind program. The advantage of using valgrind is that the programs do not have to be specially compiled.

```
# valgrind --tool=callgrind pbs_mom
```

10.1.7 Frequently Asked Questions (FAQ)

- Cannot connect to server: error=15034
- Manually deleting jobs
- Which user must run TORQUE?
- Scheduler cannot start jobs: rc=15003
- PBS_Server: pbsd_init, Unable to read server database
- qsub will not allow submission of jobs requesting many processors
- qsub reports 'Bad UID for job execution'
- Why does my job keep bouncing from running to queued?
- How do I use PVM with TORQUE?
- My build fails attempting to find the TCL library"
- My job will not start, failing with the message 'cannot send job to mom, state=PRERUN'
- I want to allow root to run jobs
- How do I determine what version of Torque I am using?

Cannot connect to server: error=15034

This error occurs in TORQUE clients (or their APIs) because TORQUE cannot find the `server_name` file and/or the **PBS_DEFAULT** environment variable is not set. The `server_name` file or **PBS_DEFAULT** variable indicate the pbs_server's hostname that the client tools should communicate with. The `server_name` file is usually located in TORQUE's local state directory. Make sure the file exists, has proper permissions, and that the version of TORQUE you are running was built with the proper directory settings. Alternatively you can set the **PBS_DEFAULT** environment variable. Restart TORQUE daemons if you make changes to these settings.

Deleting 'Stuck' Jobs

To manually delete a *stale* job which has no process, and for which the mother superior is still alive, sending a sig 0 with qsig will often cause MOM to realize the job is stale and issue the proper JobObit notice. Failing that, use momctl -c to forcefully cause MOM to purge the job. The following process should never be necessary:

- shut down the MOM on the mother superior node
- delete all files and directories related to the job from "<TORQUEHOMEDIR>/mom_priv/jobs"
- restart the MOM on the mother superior node.

If the mother superior mom has been lost and cannot be recovered (i.e, hardware or disk failure), a job running on that node can be purged from the output of qstat using the qdel -p command or can be removed manually using the following steps:

To remove job X:

1. shutdown pbs_server (**qterm**)
2. remove job spool files (**rm <TORQUEHOMEDIR>/server_priv/jobs/X.SC <TORQUEHOMEDIR>/server_priv/jobs/X.JB**)
3. restart pbs_server (**pbs_server**)

Which user must run TORQUE?

TORQUE (**pbs_server** & **pbs_mom**) must be started by a user with root privileges.

Scheduler cannot run jobs - rc: 15003

For a scheduler, such as Moab or Maui, to control jobs with TORQUE, the scheduler needs to be run be a user in the server operators / managers list (see qmgr (set server operators / managers)). The default for the server operators / managers list is root@localhost. For TORQUE to be used in a grid setting with Silver, the scheduler needs to be run as root.

PBS_Server: pbsd_init, Unable to read server database

If this message is displayed upon starting **pbs_server** it means that the local database cannot be read. This can be for several reasons. The most likely is a version mismatch. Most versions of TORQUE can read each others' databases. However, there are a few incompatibilities between OpenPBS and TORQUE. Because of enhancements to TORQUE, it cannot read the job database of an OpenPBS server (job structure sizes have been altered to increase functionality). Also, a compiled in 32 bit mode cannot read a database generated by a 64 bit **pbs_server** and vice versa.

To reconstruct a database (excluding the job database), first print out the old data with this command:

```
%> qmgr -c "p s"
#
# Create queues and set their attributes.
#
#
# Create and define queue batch
#
create queue batch
set queue batch queue_type = Execution
set queue batch acl_host_enable = False
set queue batch resources_max.nodect = 6
set queue batch resources_default.nodes = 1
set queue batch resources_default.walltime = 01:00:00
set queue batch resources_available.nodect = 18
set queue batch enabled = True
set queue batch started = True
#
# Set server attributes.
#
set server scheduling = True
set server managers = griduser@oahu.icluster.org
set server managers += scott@*.icluster.org
set server managers += wightman@*.icluster.org
set server operators = griduser@oahu.icluster.org
set server operators += scott@*.icluster.org
set server operators += wightman@*.icluster.org
set server default_queue = batch
set server log_events = 511
set server mail_from = adm
set server resources_available.nodect = 80
set server scheduler_iteration = 600
set server node_ping_rate = 300
set server node_check_rate = 600
set server tcp_timeout = 6
```

Copy this information somewhere. Restart **pbs_server** with the following command:

```
> pbs_server -t create
```

When it to prompts to overwrite the previous database enter 'y' then enter the data exported by the *qmgr* command with a command similar to the following:

```
> cat data | qmgr
```

Restart **pbs_server** without the flags:

```
> qterm
> pbs_server
```

This will reinitialize the database to the current version. Note that reinitializing the server database will reset the next jobid to 1.

---

qsub will not allow the submission of jobs requesting many processors

TORQUE's definition of a node is context sensitive and can appear inconsistent. The qsub '**-l nodes=<X>**' expression can at times indicate a request for **X** processors and other time be interpreted as a request for **X** nodes. While **qsub** allows multiple interpretations of the keyword *nodes*, aspects of the TORQUE server's logic are not so flexible. Consequently, if a job is using '-l nodes' to specify processor count and the requested number of processors exceeds the available number of physical nodes, the server daemon will reject the job.

To get around this issue, the server can be told it has an *inflated* number of nodes using the **resources_available** attribute. To take affect, this attribute should be set on both the server and the associated queue as in the example below. See resources_available for more information.

```
> qmgr
Qmgr: set server resources_available.nodect=2048
Qmgr: set queue batch resources_available.nodect=2048
```

NOTE: The **pbs_server** daemon will need to be restarted before these changes will take affect.

---

qsub reports 'Bad UID for job execution'

```
[guest@login2]$ qsub test.job
qsub: Bad UID for job execution
```

Job submission hosts must be explicitly specified within TORQUE or enabled via RCmd security mechanisms in order to be trusted. In the example above, the host '`login2`' is not configured to be trusted. This process is documented in Configuring Job Submission Hosts describing how this configuration is done.

Why does my job keep bouncing from running to queued?

There are several reasons why a job will fail to start. Do you see any errors in the MOM logs? Be sure to increase the loglevel on MOM if you don't see anything. Also be sure TORQUE is configured with **--enable-syslog** and look in `/var/log/messages` (or wherever your syslog writes).

Also verify the following on all machines:

- DNS resolution works correctly with matching forward and reverse
- time is synchronized across the head and compute nodes
- user accounts exist on all compute nodes
- user home directories can be mounted on all compute nodes
- prologue scripts (if specified) exit with 0

If using a scheduler such as Moab or Maui, use a scheduler tool such as CHECKJOB to identify job start issues.

How do I use PVM with TORQUE?

- Start the master pvmd on a compute node and then add the slaves
- mpiexec can be used to launch slaves using rsh or ssh (use export PVM_RSH=/usr/bin/ssh to use ssh)

NOTE: Access can be managed by rsh/ssh without passwords between the batch nodes, but denying it from anywhere else, including the interactive nodes. This can be done with xinetd and sshd configuration (root is allowed to ssh everywhere, of course). This way, the pvm daemons can be started and killed from the job script.

The problem is that this setup allows the users to bypass the batch system by writing a job script that uses rsh/ssh to launch processes the batch nodes. If there are relatively few users and they can more or less be trusted, this setup can work.

My build fails attempting to use the TCL library

TORQUE builds can fail on TCL dependencies even if a version of TCL is available on the system. TCL is only utilized to support the **xpbsmon** client. If your site does not use this tool (most sites do not use **xpbsmon**), you can work around this failure by rerunning **configure** with the **--disable-gui** argument.

My job will not start, failing with the message 'cannot send job to mom, state=PRERUN'

If a node crashes or other major system failures occur, it is possible that a job may be *stuck* in a corrupt state on a compute node. TORQUE 2.2.0 and higher automatically handle this when the **mom_job_sync** parameter is set via qmgr (the default). For earlier versions of TORQUE, set this parameter and restart the **pbs_mom** daemon.

I want to submit and run jobs as root

While this can be a *very* bad idea from a security point of view, in some restricted environments this can be quite useful and can be enabled by setting the acl_roots parameter via qmgr command as in the following example:
qmgr

```
> qmgr -c 's s acl_roots+=root@*'
```

How do I determine what version of Torque I am using?

There are times when you want to find out what version of Torque you are using. An easy way to do this is to run the following command:
qmgr

```
> qmgr -c "p s"|grep pbs_ver
```

See Also

- PBSCOREDUMP parameter

10.3 Debugging

10.3.1 Debugging Facilities

TORQUE supports a number of diagnostic and debug options including the following:

- **PBSDEBUG** environment variable - If set to 'yes', this variable will prevent **pbs_server**, **pbs_mom**, and/or **pbs_sched** from backgrounding themselves allowing direct launch under a debugger. Also, some client commands will provide additional diagnostic information when this value is set.
- **PBSLOGLEVEL** environment variable - Can be set to any value between 0 and 7 and specifies the logging verbosity level (default = 0)
- **PBSCOREDUMP** environment variable - If set, it will cause the offending resource manager daemon to create a core file if a **SIGSEGV**, **SIGILL**, **SIGFPE**, **SIGSYS**, or **SIGTRAP** signal is received. The core dump will be placed in the daemon's home directory (`$PBSHOME/mom_priv` for **pbs_mom**).
- **NDEBUG** #define - if set at build time, will cause additional low-level logging information to be output to stdout for **pbs_server** and **pbs_mom** daemons.
- **tracejob** reporting tool - can be used to collect and report logging and accounting information for specific jobs

10.3.2 TORQUE Error Codes

Error Code Number	Error Code Name	Description
PBSE_NONE	15000	no error
PBSE_UNKJOBID	15001	Unknown Job Identifier
PBSE_NOATTR	15002	Undefined Attribute
PBSE_ATTRRO	15003	attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	No permission
PBSE_BADHOST	15008	access from host not allowed
PBSE_JOBEXIST	15009	job already exists
PBSE_SYSTEM	15010	system error occurred
PBSE_INTERNAL	15011	internal server error occurred
PBSE_REGROUTE	15012	parent job of dependent in rte queue

PBSE_UNKSIG	15013	unknown signal name
PBSE_BADATVAL	15014	bad attribute value
PBSE_MODATRRUN	15015	Cannot modify attribute in run state
PBSE_BADSTATE	15016	request invalid for job state
PBSE_UNKQUE	15018	Unknown queue name
PBSE_BADCRED	15019	Invalid Credential in request
PBSE_EXPIRED	15020	Expired Credential in request
PBSE_QUNOENB	15021	Queue not enabled
PBSE_QACESS	15022	No access permission for queue
PBSE_BADUSER	15023	Bad user - no password entry
PBSE_HOPCOUNT	15024	Max hop count exceeded
PBSE_QUEEXIST	15025	Queue already exists
PBSE_ATTRTYPE	15026	incompatible queue attribute type
PBSE_QUEBUSY	15027	Queue Busy (not empty)
PBSE_QUENBIG	15028	Queue name too long
PBSE_NOSUP	15029	Feature/function not supported
PBSE_QUENOEN	15030	Cannot enable queue,needs add def
PBSE_PROTOCOL	15031	Protocol (ASN.1) error
PBSE_BADATLST	15032	Bad attribute list structure
PBSE_NOCONNECTS	15033	No free connections
PBSE_NOSERVER	15034	No server to connect to
PBSE_UNKRESC	15035	Unknown resource
PBSE_EXCQRESC	15036	Job exceeds Queue resource limits
PBSE_QUENODFLT	15037	No Default Queue Defined
PBSE_NORERUN	15038	Job Not Rerunnable
PBSE_ROUTEREJ	15039	Route rejected by all destinations
PBSE_ROUTEEXPD	15040	Time in Route Queue Expired
PBSE_MOMREJECT	15041	Request to MOM failed
PBSE_BADSCRIPT	15042	(qsub) cannot access script file
PBSE_STAGEIN	15043	Stage In of files failed
PBSE_RESCUNAV	15044	Resources temporarily unavailable
PBSE_BADGRP	15045	Bad Group specified
PBSE_MAXQUED	15046	Max number of jobs in queue
PBSE_CKPBSY	15047	Checkpoint Busy, may be retries
PBSE_EXLIMIT	15048	Limit exceeds allowable
PBSE_BADACCT	15049	Bad Account attribute value
PBSE_ALRDYEXIT	15050	Job already in exit state
PBSE_NOCOPYFILE	15051	Job files not copied
PBSE_CLEANEDOUT	15052	unknown job id after clean init
PBSE_NOSYNCMSTR	15053	No Master in Sync Set
PBSE_BADDEPEND	15054	Invalid dependency

PBSE_DUPLIST	15055	Duplicate entry in List
PBSE_DISPROTO	15056	Bad DIS based Request Protocol
PBSE_EXECTHERE	15057	cannot execute there
PBSE_SISREJECT	15058	sister rejected
PBSE_SISCOMM	15059	sister could not communicate
PBSE_SVRDOWN	15060	requirement rejected -server shutting down
PBSE_CKPSHORT	15061	not all tasks could checkpoint
PBSE_UNKNODE	15062	Named node is not in the list
PBSE_UNKNODEATR	15063	node-attribute not recognized
PBSE_NONODES	15064	Server has no node list
PBSE_NODENBIG	15065	Node name is too big
PBSE_NODEEXIST	15066	Node name already exists
PBSE_BADNDATVAL	15067	Bad node-attribute value
PBSE_MUTUALEX	15068	State values are mutually exclusive
PBSE_GMODERR	15069	Error(s) during global modification of nodes
PBSE_NORELYMOM	15070	could not contact Mom
PBSE_NOTSNODE	15071	no time-shared nodes

See Also

- Troubleshooting Guide

Appendices

Appendix A: Commands Overview

A.1 Client Commands

Command	**Description**
momctl	manage/diagnose MOM (node execution) daemon
pbsdsh	launch tasks within a parallel job
pbsnodes	view/modify batch status of compute nodes
qalter	modify queued batch jobs
qchkpt	checkpoint batch jobs
qdel	delete/cancel batch jobs
qhold	hold batch jobs
qmgr	manage policies and other batch configuration
qrerun	rerun a batch job
qrls	release batch job holds
qrun	start a batch job
qsig	send a signal to a batch job
qstat	view queues and jobs

qsub	submit jobs
qterm	shutdown pbs server daemon
tracejob	trace job actions and states recorded in TORQUE logs

A.2 Server Commands

Command	Description
pbs_iff	interprocess authentication service
pbs_mom	start MOM (node execution) daemon
pbs_server	start server daemon
pbs_track	tell pbs_mom to track a new process

See Also

- MOM Configuration
- Server Parameters

Commands

momctl (PBS Mom Control)

Synopsis

```
momctl -c { <JOBID> | all }
momctl -C
momctl -d { <INTEGER> | <JOBID> }
momctl -f <FILE>
momctl -h <HOST>[,<HOST>]...
momctl -p <PORT_NUMBER>
momctl -q <ATTRIBUTE>
momctl -r { <FILE> | LOCAL:<FILE> }
momctl -s
```

Overview

The **momctl** command allows remote shutdown, reconfiguration, diagnostics, and querying of the pbs_mom daemon.

Format

Name	Format	Default	Description	Example
Clear	{ <JOBID> | **all** }	---	Clear stale job information	"momctl -h node1 **-c 15406**"
Cycle	---	---	Cycle pbs_mom(s)	"momctl -h node1 **-C**"

				Cycle pbs_mom on node1
Diagnose	{ <INTEGER> | <JOBID> }	0	Diagnose mom(s) See the Diagnose Detail table below for more information.	`"momctl -h node1 **-d 2**"` (Print level 2 and lower diagnose information for the mom on node1)
Host File	<FILE>	---	A file contain only comma or whitespace (i.e., " ", "\t", or "\n") delimited hostnames	`"momctl **-f hosts.txt** -d"` (Print diagnose information for the moms running on the hosts specified in hosts.txt)
Host List	<HOST>[,<HOST>]...	localhost	A comma separated list of hosts	`"momctl **-h node1,node2,node3** -d"` (Print diagnose information for the moms running on node1, node2 and node3)
Port	<PORT_NUMBER>	TORQUE's default port number	The port number for the specified mom(s)	`"momctl **-p 5455** -h node1 -d"` (Request diagnose information over port 5455 on node1)
Query	<ATTRIBUTE>	---	Query <ATTRIBUTE> on specified mom (where <ATTRIBUTE> is a property listed by pbsnodes -a)	`"momctl -q physmem"` (Print the amount of physmem on localhost)
Reconfigure	{ <FILE> | **LOCAL:**<FILE> }	---	Reconfigure mom(s) with remote or local config file, <FILE>	`"momctl **-r /home/user1/new.config** -h node1"` (Reconfigure mom on node1 with /home/user1/new.config on node1)
Shutdown		---	Shutdown pbs_mom	`"momctl -s"` (Terminates pbs_mom process on localhost)

Query Attributes

- **arch** - node hardware architecture
- **availmem** - available RAM
- **loadave** - 1 minute load average
- **ncpus** - number of CPUs available on the system
- **netload** - total number of bytes transferred over all network interfaces
- **nsessions** - number of sessions active
- **nusers** - number of users active

- **physmem** - configured RAM
- **sessions** - list of active sessions
- **totmem** - configured RAM plus configured swap

Diagnose Detail

Level	Description
0	Display the following information: • Local hostname • Expected server hostname • Execution version • MOM home directory • MOM config file version (if specified) • Duration MOM has been executing • Duration since last request from **pbs_server** daemon • Duration since last request to **pbs_server** daemon • RM failure messages (if any) • Log verbosity level • Local job list
1	All information for level 0 plus the following: • Interval between updates sent to server • Number of initialization messages sent to **pbs_server** daemon • Number of initialization messages received from **pbs_server** daemon • Prolog/epilog alarm time • List of trusted clients
2	All information from level 1 plus the following: • PID • Event alarm status

Example 1: MOM Diagnostics

```
> momctl -d 1

Host: nsrc/nsrc.fllcl.com   Server: 10.10.10.113   Version: torque_1.1.0p4
HomeDirectory:          /usr/spool/PBS/mom_priv
ConfigVersion:          147
MOM active:             7390 seconds
Last Msg From Server:   7389 seconds (CLUSTER_ADDRS)
Server Update Interval: 20 seconds
Server Update Interval: 20 seconds
Init Msgs Received:     0 hellos/1 cluster-addrs
Init Msgs Sent:         1 hellos
LOGLEVEL:               0 (use SIGUSR1/SIGUSR2 to adjust)
Prolog Alarm Time:      300 seconds
Trusted Client List:    12.14.213.113,127.0.0.1
JobList:                NONE

diagnostics complete
```

Example 2: System Shutdown

```
> momctl -s -f /opt/clusterhostfile

shutdown request successful on node001
shutdown request successful on node002
shutdown request successful on node003
shutdown request successful on node004
shutdown request successful on node005
shutdown request successful on node006
```

pbsdsh

distribute tasks to nodes under pbs

SYNOPSIS

```
pbsdsh [-c copies] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-n node] [-o] [-s] [-u] [-v] program [args]
pbsdsh [-h nodename] [-o] [-v] program [args]
```

DESCRIPTION

Executes (spawns) a normal Unix program on one or more nodes under control of the Portable Batch System, PBS. Pbsdsh uses the Task Manager API, see tm_spawn(3), to distribute the program on the allocated nodes.

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

Users will find the PBS_TASKNUM, PBS_NODENUM, and the PBS_VNODENUM environmental variables useful. They contain the TM task id, the node identifier, and the cpu (virtual node) identifier.

OPTIONS

-c *copies*
The program is spawned on the first Copies nodes allocated. This option is mutual exclusive with -n.

-h *hostname*
The program is spawned on the node specified.

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

-o
Capture stdout of the spawned program. Normally stdout goes to the job’s output.

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

-u
The program is run once on each node (unique). This ignores the number of allocated processors on a given node.

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

OPERANDS

The first operand, program, is the program to execute.

Additional operands, args, are passed as arguments to the program.

STANDARD ERROR

The pbsdsh command will write a diagnostic message to standard error for each error occurrence.

EXIT STATUS

Upon successful processing of all the operands presented to the command, the exit status will be a value of zero.

If the pbsdsh command fails to process any operand, or fails to contact the MOM daemon on the localhost the command exits with a value greater than zero.

SEE ALSO

qsub(1B), tm_spawn(3B)

pbsnodes

pbs node manipulation

Synopsis

```
pbsnodes [-{a|x}] [-q] [-s server] [node|:property]
pbsnodes -l [-q] [-s server] [state] [nodename|:property ...]
pbsnodes [-{c|d|o|r}] [-q] [-s server] [-n] [-N "note"] [node|:property]
```

Description

The pbsnodes command is used to mark nodes down, free or offline. It can also be used to list nodes and their state. Node information is obtained by sending a request to the PBS job server. Sets of nodes can be operated on at once by specifying a node property prefixed by a colon.

Nodes do not exist in a single state, but actually have a set of states. For example, a node can be simultaneously "busy" and "offline". The "free" state is the absense of all other states and so is never combined with other states.

In order to execute pbsnodes with other than the -a or -l options, the user must have PBS Manager or Operator privilege.

Options

-a
: All attributes of a node or all nodes are listed. This is the default if no flag is given.

-x
: Same as -a, but the output has an XML-like format.

-c
: Clear OFFLINE from listed nodes.

-d
: Print MOM diagnosis on the listed nodes. Not yet implemented. Use momctl instead.

-o
: Add the OFFLINE state. This is different from being marked DOWN. OFFLINE prevents new jobs from running on the specified nodes. This gives the administrator a tool to hold a node out of service without changing anything else. The OFFLINE state will never be set or cleared automatically by pbs_server; it is purely for the manager or operator.

-p
: Purge the node record from pbs_server. Not yet implemented.

-r
: Reset the listed nodes by clearing OFFLINE and adding DOWN state. pbs_server will ping the node and, if they communicate correctly, free the node.

-l
: List node names and their state. If no state is specified, only nodes in the DOWN, OFFLINE, or UNKNOWN states are listed. Specifying a state string acts as an output filter. Valid state strings are "active", "all", "busy", "down", "free", "offline", "unknown", and "up".

-N
: Specify a "note" attribute. This allows an administrator to add an arbitrary annotation to the listed nodes. To clear a note, use -N "" or -N n.

-n
: Show the "note" attribute for nodes that are DOWN, OFFLINE, or UNKNOWN. This option requires -l.

-q
: Supress all error messages.

-s

Specify the PBS servers hostname or IP address.

See Also

pbs_server(8B) and the PBS External Reference Specification

qalter

alter batch job

SYNOPSIS

```
qalter [-a date_time][-A account_string][-c interval][-e path_name]
       [-h hold_list][-j join_list][-k keep_list][-l resource_list]
       [-m mail_options][-M mail_list][-N name][-o path_name]
       [-p priority][-q ][-r y|n][-S path_name_list][-u user_list]
       [-v variable_list][-W additional_attributes]
       job_identifier ...
```

DESCRIPTION

The qalter command modifies the attributes of the job or jobs specified by job_identifier on the command line. Only those attributes listed as options on the command will be modified. If any of the specified attributes cannot be modified for a job for any reason, none of that jobs attributes will be modified.

The qalter command accomplishes the modifications by sending a Modify Job batch request to the batch server which owns each job.

OPTIONS

-a *date_time*

Replaces the time at which the job becomes eligible for execution. The date_time argument syntax is:

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

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.
This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

-A *account_string*

Replaces the the account string associated with the job. This attribute cannot be altered once the job has begun execution.

-c *interval*

Replaces the the interval at which the job will be check pointed. If the job executes upon a host which does not support checkpoint, 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 cpu time for the queue from which the job is executing.

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. If the number is less than the default checkpoint time, the default time will be used.

This attribute can be altered once the job has begun execution, but the new value does not take affect until the job is rerun.

-e *path_name*

Replaces the the path to be used for the standard error stream of the batch job. 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 in the syntax recognized by POSIX 1003.1. The argument will be interpreted as follows:

path_name

Where path_name is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component.

hostname:path_name

Where path_name is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname.

path_name

Where path_name specifies an absolute path name, then qalter will supply the name of the host on which it is executing for the hostname.

hostname:path_name

Where path_name specifies an absolute path name, the path will be used as specified.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

-h *hold_list*

Updates the the types of holds on the job. The hold_list argument is a string of one or more of the following characters:

u

Add the USER type hold.

s

Add the SYSTEM type hold if the user has the appropriate level of privilege. [Typically reserved to the batch administrator.]

o

Add the OTHER (or OPERATOR) type hold if the user has the appropriate level of privilege. [Typically reserved to the batch administrator and batch operator.]

n

Set to none; that is clear the hold types which could be applied with the users level of privilege.

Repetition of characters is permitted, but "n" may not appear in the same option argument with the other three characters. This attribute can be altered once the job has begun execution, but the hold will not take affect until the job is rerun.

-j *join*

Declares which standard streams of the job will be merged together. The join argument value may be the characters "oe" and "eo", or the single character "n".

A argument value of oe directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard output. A argument value of eo directs that the standard output and standard error streams of the job will be merged, intermixed, and returned as the standard error.

A value of n directs that the two streams will be two separate files. This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

-k *keep*

Defines which if either of standard output or standard error of the job will be retained on the execution host. If set for a stream, this option overrides the path name for that stream.

The argument is either the single letter "e", "o", or "n", or one or more of the letters "e" and "o" combined in either order.

n

No streams are to be retained.

e

The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:

```
job_name.esequence
```

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

o

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

```
job_name.osequence
```

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

eo

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

oe

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

This attribute cannot be altered once the job has begun execution.

-l *resource_list*

Modifies the list of resources that are required by the job. The Resource_List argument is in the following syntax:

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

If a requested modification to a resource would exceed the resource limits for jobs in the current queue, the server will reject the request.

If the job is running, only certain, resources can be altered. Which resources can be altered in the run state is system dependent. A user may only lower the limit for those resources.

-m *mail_options*

Replaces the set of conditions under which the execution server will send a mail message about the job. The mail_options argument is a string which consists of the single character "n", or one or more of the characters "a", "b", and "e".

If the character "n" is specified, no mail will be sent.

For the letters "a", "b", and "e":

a
: mail is sent when the job is aborted by the batch system.

b
: mail is sent when the job begins execution.

e
: mail is sent when the job terminates.

-M *user_list*
: Replaces 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],...]
```

-N *name*
: Renames the job. The name specified may be up to and including 15 characters in length. It must consist of printable, non white space characters with the first character alphabetic.

-o *path*
: Replaces the path to be used for the standard output stream of the batch job. 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 in the syntax recognized by POSIX. The argument will be interpreted as follows:

path_name
: Where path_name is not an absolute path name, then the qalter command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component.

hostname:path_name
: Where path_name is not an absolute path name, then the qalter command will not expand the path name. The execution server will expand it relative to the home directory of the user on the system specified by hostname.

```
path_name
```

Where path_name specifies an absolute path name, then qalter will supply the name of the host on which it is executing for the hostname.

```
hostname:path_name
```

Where path_name specifies an absolute path name, the path will be used as specified.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

-p *priority*

Replaces the priority of the job. The priority argument must be a integer between -1024 and +1023 inclusive.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

-r *c*

Declares whether the job is rerunable. See the qrerun command. The option argument c is a single character. PBS recognizes the following characters: y and n.

If the argument is "y", the job is marked rerunable. If the argument is "n", the job is marked as not rerunable.

-S *path*

Declares the shell that interprets the job script.

The option argument path_list is in the form:

```
path[@host][,path[@host],...]
```

Only one path may be specified for any host named. 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 if present the path specified without a host will be selected.

If the -S option is not specified, the option argument is the null string, or no entry from the path_list is selected, the execution will use the login shell of the user on the execution host.

This attribute can be altered once the job has begun execution, but it will not take affect until the job is rerun.

-u *user_list*

Replaces the user name under which the job is to run on the execution system.

The user_list argument is of the form:

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

Only one user name may be given for per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will be used for execution on any host not named in the argument list.

This attribute cannot be altered once the job has begun execution.

-W *additional_attributes*

The -W option allows for the modification of additional job attributes.

Note if white space occurs anywhere within the option argument string or the equal sign, "=", occurs within an attribute_value string, then the string must be enclosed with either single or double quote marks.

PBS currently supports the following attributes within the -W option.

depend=dependency_list

Redefines the dependencies between this and other jobs. The dependency_list is in the form:

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

The argument is either a numeric count or a PBS job id accord ing to type. If argument is a count, it must be greater than 0. If it is a job id and is not fully specified in the form: seq_number.server.name, it will be expanded according to the default server rules. If argument is null (the preceding colon need not be specified), the dependency of the corresponding type is cleared (unset).

synccount:count

This job is the first in a set of jobs to be executed at the same time. Count is the number of additional jobs in the set.

syncwith:jobid

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

after:jobid [:jobid...]

This job may be scheduled for execution at any point after jobs jobid have started execution.

afterok:jobid [:jobid...]

This job may be scheduled for execution only after jobs jobid have terminated with no errors. See the csh warning under "Extended Description".

afternotok:jobid [:jobid...]

This job may be scheduled for execution only after jobs jobid have terminated with errors. See the csh warning under "Extended Description".

afterany:jobid [:jobid...]

This job may be scheduled for execution after jobs jobid have terminated, with or without errors.

on:count

This job may be scheduled for execution after count dependencies on other jobs have been satisfied. This form is used in conjunction with one of the forms, see below.

before:jobid [:jobid...]

When this job has begun execution, then jobs jobid... may begin.

beforeok:jobid [:jobid...]

If this job terminates execution without errors, then jobs jobid... may begin. See the csh warning under "Extended Description".

beforenotok:jobid [:jobid...]

If this job terminates execution with errors, then jobs jobid... may begin. See the csh warning under "Extended Description".

beforeany:jobid [:jobid...]

When this job terminates execution, jobs jobid... may begin.

If any of the before forms are used, the job referenced by jobid must have been submitted with a dependency type of on.

If any of the before forms are used, the jobs referenced by jobid must have the same owner as the job being altered. Otherwise, the dependency will not take effect.

Error processing of the existence, state, or condition of the job specified to qalter is a deferred service, i.e. the check is performed after the job is queued. If an error is detected, the job will be deleted by the server. Mail will be sent to the job submitter stating the error.

group_list=g_list

Alters 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.

stagein=file_list
stageout=file_list

Alters which files are staged (copied) in before job start or staged out after the job completes execution. The file_list is in the form:

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

The name local_file is the name on the system where the job executes. It may be an absolute path or a path 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 users home directory on the destination host.

OPERANDS

The qalter command accepts one or more job_identifier operands of the form:

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

STANDARD ERROR

Any error condition, either in processing the options or the operands, or any error received in reply to the batch requests will result in an error message being written to standard error.

EXIT STATUS

Upon successful processing of all the operands presented to the qalter command, the exit status will be a value of zero.

If the qalter command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

Batch Environment Services , *qdel* , *qhold* , *qmove* , *qrls* , *qsub* , *touch*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1, 2003 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright (C) 2001-2003 by the Institute of

Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

qchkpt

checkpoint pbs batch jobs

Synopsis

```
qchkpt <JOBID>[ <JOBID>] ...
```

Description

The qchkpt command requests that the PBS Mom generate a checkpoint file for a running job.

This is an extension to POSIX.2d.

The qchkpt command sends a Chkpt Job batch request to the server as described in the general section.

Options

None

Operands

The qchkpt command accepts one or more job_identifier operands of the form:

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

Examples

```
> qchkpt 3233     request a checkpoint for job 3233
```

Standard Error

The qchkpt command will write a diagnostic message to standard error for each error occurrence.

Exit Status

Upon successful processing of all the operands presented to the qchkpt command, the exit status will be a value of zero.

If the qchkpt command fails to process any operand, the command exits with a value greater than zero.

See Also

```
qhold(1B), qrls(1B), qalter(1B), qsub(1B), pbs_alterjob(3B), pbs_holdjob(3B),
  pbs_rlsjob(3B), pbs_job_attributes(7B), pbs_resources_unicos8(7B)
```

qdel (delete job)

Synopsis

```
qdel [{-m <MESSAGE>|-p|-W <DELAY>}] <JOBID>[ <JOBID>]... | 'all' | 'ALL'
```

Description

The qdel command deletes jobs in the order in which their job identifiers are presented to the command. A job is deleted by sending a Delete Job batch request to the batch server that owns the job. A job that has been deleted is no longer subject to management by batch services.

A batch job may be deleted by its owner, the batch operator, or the batch administrator.

A batch job being deleted by a server will be sent a SIGTERM signal following by a SIGKILL signal. The time delay between the two signals is an attribute of the execution queue from which the job was run (set table by the administrator). This delay may be overridden by the -W option.

See the PBS ERS section 3.1.3.3, "Delete Job Request", for more information.

Options

-w *delay*
: Specify the delay between the sending of the SIGTERM and SIGKILL signals. The argument delay specifies a unsigned integer number of seconds.

-p *purge*
: Forcibly purge the job from the server. This should only be used if a running job will not exit because its allocated nodes are unreachable. The admin should make every attempt at resolving the problem on the nodes. If a jobs mother superior recovers after purging the job, any epilogue scripts may still run. This option is only available to a batch operator or the batch administrator.

-m *message*
: Specify a comment to be included in the email. The argument message specifies the comment to send. This option is only available to a batch operator or the batch administrator.

Operands

The qdel command accepts one or more job_identifier operands of the form:

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

```
  all
```

Examples

```
> qdel 1324
> qdel 1324-3    To delete one job of a job array
> qdel all       To delete all jobs (Version 2.3.0 and later)
```

Standard Error

The qdel command will write a diagnostic messages to standard error for each error occurrence.

Exit Status

Upon successful processing of all the operands presented to the qdel command, the exit status will be a value of zero.

If the qdel command fails to process any operand, the command exits with a value greater than zero.

See Also

qsub(1B), qsig(1B), and pbs_deljob(3B)

qhold (hold job)

Synopsis

```
qhold [{-h <HOLD LIST>}] <JOBID>[ <JOBID>] ...
```

Description

The qhold command requests that a server place one or more holds on a job. A job that has a hold is not eligible for execution. There are three supported holds: USER, OTHER (also known as operator), and SYSTEM.

A user may place a USER hold upon any job the user owns. An "operator", who is a user with "operator privilege," may place ether an USER or an OTHER hold on any job. The batch administrator may place any hold on any job.

If no -h option is given, the USER hold will be applied to the jobs described by the job_identifier operand list.

If the job identified by job_identifier is in the queued, held, or waiting states, then all that occurs is that the hold type is added to the job. The job is then placed into held state if it resides in an execution queue.

If the job is in running state, then the following additional action is taken to interrupt the execution of the job. If checkpoint / restart is supported by the host system, requesting a hold on a running job will (1) cause the job to be checkpointed, (2) the resources assigned to the job will be released, and (3) the job is placed in the held state in the execution queue.

If checkpoint / restart is not supported, qhold will only set the the requested hold attribute. This will have no effect unless the job is rerun with the qrerun command.

The qhold command sends a Hold Job batch request to the server as described in the general section.

Options

-h *hold_list*
The hold_list argument is a string consisting of one or more of the letters "u", "o", or "s" in any combination. The hold type associated with each letter is:

u - USER
o - OTHER
s - SYSTEM

Operands

The qhold command accepts one or more job_identifier operands of the form:

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

Examples

```
> qhold -h u 3233     place user hold on job 3233
```

Standard Error

The qhold command will write a diagnostic message to standard error for each error occurrence.

Exit Status

Upon successful processing of all the operands presented to the qhold command, the exit status will be a value of zero.

If the qhold command fails to process any operand, the command exits with a value greater than zero.

See Also

```
qrls(1B), qalter(1B), qsub(1B), pbs_alterjob(3B), pbs_holdjob(3B),
pbs_rlsjob(3B), pbs_job_attributes(7B), pbs_resources_unicos8(7B)
```

qmgr (PBS Queue Manager)

NAME

qmgr - pbs batch system manager

SYNOPSIS

qmgr [-a] [-c command] [-e] [-n] [-z] [server...]

DESCRIPTION

The **qmgr** command provides an administrator interface to query and configure batch system parameters.

The command reads directives from standard input. The syntax of each directive is checked and the appropriate request is sent to the batch server or servers.

The list or print subcommands of qmgr can be executed by general users. Creating or deleting a queue requries PBS Manager privilege. Setting or unsetting server or queue attributes requires PBS Operator or Manager privilege. **NOTE**: by default the user **root** is the only PBS Operator and Manager. To allow other users to be privileged, the `server` attributes **operators** and **managers** will need to be set (i.e., as root, issue '`qmgr -c 'set server managers += <USER1>@<HOST>`'). See PBS Access Config for more information.

OPTIONS

-a
    Abort **qmgr** on any syntax errors or any requests rejected by a server.

-c command
    Execute a single *command* and exit **qmgr .**

-e
    Echo all commands to standard output.

-n
    No commands are executed, syntax checking only is performed.

-z
    No errors are written to standard error.

OPERANDS

The *server* operands identify the name of the batch server to which the administrator requests are sent. Each *server* conforms to the following syntax:
host_name[:port]

where **host_name** is the network name of the host on which the server is running and **port** is the port number to which to connect. If **port** is not specified, the default port number is used.

If *server* is not specified, the administrator requests are sent to the local server.

STANDARD INPUT

The **qmgr** command reads standard input for directives until end of file is reached, or the *exit* or *quit* directive is read.

STANDARD OUTPUT

If Standard Output is connected to a terminal, a command prompt will be written to standard output when qmgr is ready to read a directive.

If the *-e* option is specified, **qmgr** will echo the directives read from standard input to standard output.

STANDARD ERROR

If the *-z* option is not specified, the qmgr command will write a diagnostic message to standard error for each error occurrence.

EXTENDED DESCRIPTION

If **qmgr** is invoked without the *-c* option and standard output is connected to a terminal, qmgr will write a prompt to standard output and read a directive from standard input.

Commands can be abbreviated to their minimum unambiguous form. A command is terminated by a new line character or a semicolon, ";", character. Multiple commands may be entered on a single line. A command may extend across lines by escaping the new line character with a back-slash "\".

Comments begin with the # character and continue to end of the line. Comments and blank lines are ignored by qmgr.

DIRECTIVE SYNTAX

A qmgr directive is one of the following forms:

```
command server [names] [attr OP value[,attr OP value,...]]
command queue [names] [attr OP value[,attr OP value,...]]
command node [names] [attr OP value[,attr OP value,...]]
```

Where, command is the command to perform on a object. Commands are:

active
: sets the active objects. If the active objects are specified, and the name is not given in a qmgr cmd the active object names will be used.

create
: is to create a new object, applies to queues and nodes.

delete
: is to destroy an existing object, applies to queues and nodes.

set
: is to define or alter attribute values of the object.

unset
: is to clear the value of attributes of the object. Note, this form does not accept an OP and value, only the attribute name.

list
: is to list the current attributes and associated values of the object.

print
: is to print all the queue and server attributes in a format that will be usable as input to the qmgr command.

names
: is a list of one or more names of specific objects The name list is in the form:
[name][@server][,queue_name[@server]...]
with no intervening white space. The name of an object is declared when the object is first created. If the name is @server, then all the objects of specified type at the server will be effected.

attr
: specifies the name of an attribute of the object which is to be set or modified. If the attribute is one which consist of a set of resources, then the attribute is specified in the form:
attribute_name**.**resource_name

OP
: operation to be performed with the attribute and its value:
=
set the value of the attribute. If the attribute has a existing value, the current value is replaced with the new value.
+=
increase the current value of the attribute by the amount in the new value.
-=
decrease the current value of the attribute by the amount in the new value.

value
: the value to assign to an attribute. If the value includes white space, commas or other special characters, such as the # character, the value string must be inclosed in quote marks (").

The following are examples of qmgr directives:

```
create queue fast priority=10,queue_type=e,enabled = true,max_running=0
set queue fast max_running +=2
create queue little
set queue little resources_max.mem=8mw,resources_max.cput=10
unset queue fast max_running
set node state = "down,offline"
active server s1,s2,s3
list queue @server1
```

```
set queue max_running = 10      - uses active queues
```

EXIT STATUS

Upon successful processing of all the operands presented to the qmgr command, the exit status will be a value of zero.

If the qmgr command fails to process any operand, the command exits with a value greater than zero.

SEE ALSO

pbs_server (8B), pbs_queue_attributes (7B), pbs_server_attributes (7B), qstart (8B), qstop (8B), qenable (8B), qdisable (8), and the PBS External Reference Specification

qrerun (rerun a batch job)

Synopsis

```
qrerun [{-f}] <JOBID>[ <JOBID>] ...
```

Overview

The **qrerun** command requests that a job be rerun.

Format

Name	Format	Default	Description	Example
-f	---	---	force a rerun on job.	qrerun -f 15406

Command Details

The **qrerun** command directs that the specified jobs are to be rerun if possible. To rerun a job is to terminate the session leader of the job and return the job to the queued state in the execution queue in which the job currently resides.

If a job is marked as not rerunable then the rerun request will fail for that job. If the mini-server running the job is down, or it rejects the request, the Rerun Job batch request will return a failure unless **-f** is used.

Using **-f** violates IEEE Batch Processing Services Std and should be handled with great care. It should only be used under exceptional circumstances. Best practice is to fix the problem mini-server host and let qrerun run normally. The nodes may need manual cleaning. See the **-r** option on the **qsub** and **qalter** commands.

Operands

The qrerun command accepts one or more job_identifier operands of the form:

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

Standard Error

The qrerun command will write a diagnostic message to standard error for each error occurrence.

Exit Status

Upon successful processing of all the operands presented to the qrerun command, the exit status will be a value of zero.

If the qrerun command fails to process any operand, the command exits with a value greater than zero.

Examples

```
> qrerun 3233        rerun job 3233
```

See Also

qsub(1B), qalter(1B), pbs_alterjob(3B), pbs_rerunjob(3B)

qrls (release hold on pbs batch jobs)

Synopsis

```
qrls [{-h <HOLD LIST>}] <JOBID>[ <JOBID>] ...
```

Description

The **qrls** command removes or releases holds which exist on batch jobs.

A job may have one or more types of holds which make the job ineligible for execution. The types of holds are USER, OTHER, and SYSTEM. The different types of holds may require that the user issuing the qrls command have special privilege. Typically, the owner of the job will be able to remove a USER hold, but not an OTHER or SYSTEM hold. An attempt to release a hold for which the user does not have the correct privilege is an error and no holds will be released for that job.

If no -h option is specified, the USER hold will be released.

If the job has no execution_time pending, the job will change to the queued state. If an execution_time is still pending, the job will change to the waiting state.

Options

-h *hold_list*

Defines the types of hold to be released from the jobs. The hold_list option argument is a string consisting of one or more of the letters "u", "o", an "s" in any combination. The hold type associated with each letter is:

u - USER
o - OTHER
s - SYSTEM

Operands

The qrls command accepts one or more job_identifier operands of the form:

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

Examples

```
> qrls -h u 3233     release user hold on job 3233
```

Standard Error

The qrls command will write a diagnostic message to standard error for each error occurrence.

Exit Status

Upon successful processing of all the operands presented to the qrls command, the exit status will be a value of zero.

If the qrls command fails to process any operand, the command exits with a value greater than zero.

See Also

```
qsub(1B), qalter(1B), qhold(1B), pbs_alterjob(3B), pbs_holdjob(3B), and
pbs_rlsjob(3B).
```

qrun (run a batch job)

Synopsis

```
qrun [{-H <HOST>|-a}] <JOBID>[ <JOBID>] ...
```

Overview

The **qrun** command runs a job.

Format

Name	Format	Default	Description	Example
-H	<STRING> Host Identifier	---	specifies the host within the cluster on which the job(s) are to be run. The host argument is the name of a host that is a member of the cluster of hosts managed by the server. If the option is not specified, the server will select the "worst possible" host on which to execute the job.	`qrun -H hostname 15406`
-a	---	---	run the job(s) asynchronously.	`qrun -a 15406`

Command Details

The **qrun** command is used to force a batch server to initiate the execution of a batch job. The job is run regardless of scheduling position or resource requirements.

In order to execute qrun, the user must have PBS Operation or Manager privilege.

Examples

```
> qrun 3233          run job 3233
```

qsig (signal a job)

Synopsis

```
qsig [{-s <SIGNAL>}] <JOBID>[ <JOBID>] ...
```

Description

The qsig command requests that a signal be sent to executing batch jobs. The signal is sent to the session leader of the job. If the -s option is not specified, SIGTERM is sent. The request to signal a batch job will be rejected if:

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

The qsig command sends a Signal Job batch request to the server which owns the job.

Options

-s *signal*
Declares which signal is sent to the job.

The signal argument is either a signal name, e.g. SIGKILL, the signal name without the SIG prefix, e.g. KILL, or a unsigned signal number, e.g. 9. The signal name SIGNULL is allowed; the server will send the signal 0 to the job which will have no effect on the job, but will cause an obituary to be sent if the job is no longer executing. Not all signal names will be recognized by qsig. If it doesnt recognize the signal name, try issuing the signal number instead.

Two special signal names, "suspend" and "resume", are used to suspend and resume jobs. Cray systems use the Cray-specific suspend()/resume() calls.

On non-Cray system, suspend causes a SIGTSTP to be sent to all processes in jobs top task, wait 5 seconds, and then send a SIGSTOP to all processes in all tasks on all nodes in the job. This differs from TORQUE 2.0.0 which did not have the ability to propogate signals to sister nodes. Resume sends a SIGCONT to all processes in all tasks on all nodes.

When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. The job will be listed in the "S" state. Manager or operator privilege is required to suspend or resume a job.

Note that interactive jobs may not resume properly because the top-level shell will background the suspended child process.

Operands

The qsig command accepts one or more job_identifier operands of the form:

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

Examples

```
> qsig -s SIGKILL 3233     send a SIGKILL to job 3233
> qsig -s KILL 3233        send a SIGKILL to job 3233
> qsig -s 9 3233           send a SIGKILL to job 3233
```

Standard Error

The qsig command will write a diagnostic messages to standard error for each error occurrence.

Exit Status

Upon successful processing of all the operands presented to the qsig command, the exit status will be a value of zero.

If the qsig command fails to process any operand, the command exits with a value greater than zero.

See Also

```
  qsub(1B), pbs_sigjob(3B), pbs_resources_*(7B) where * is system type,
  and the PBS ERS.
```

qstat

show status of pbs batch jobs

SYNOPSIS

```
qstat [-f [-1]][-W site_specific] [job_identifier... | destination...]

qstat   [-a|-i|-r|-e]  [-n  [-1]]  [-s] [-G|-M]  [-R]  [-u  user_list]
        [job_identifier... |  destination...]

qstat -Q [-f [-1]][-W site_specific] [destination...]

qstat -q [-G|-M] [destination...]

qstat -B [-f [-1]][-W site_specific] [server_name...]
```

DESCRIPTION

The qstat command is used to request the status of jobs, queues, or a batch server. The requested status is written to standard out.

When requesting job status, synopsis format 1 or 2, qstat will output information about each job_identifier or all jobs at each destination. Jobs for which the user does not have status privilege are not displayed.

When requesting queue or server status, synopsis format 3 through 5, qstat will output information about each destination.

OPTIONS

-f
: Specifies that a full status display be written to standard out.

-a
: "All" jobs are displayed in the alternative format, see the Standard Output section. If the operand is a destination id, all jobs at that destination are displayed. If the operand is a job id, information about that job is displayed.

-e
: If the operand is a job id or not specified, only jobs in executable queues are displayed. Setting the PBS_QSTAT_EXECONLY environment variable will also enable this option.

-i

Job status is displayed in the alternative format. For a destination id operand, status for jobs at that destination which are not running are displayed. This includes jobs which are queued, held or waiting. If an operand is a job id, status for that job is displayed regardless of its state.

-r

If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs at that destination which are running are displayed, this includes jobs which are suspended.

-n

In addition to the basic information, nodes allocated to a job are listed.

-1

In combination with -n, the -1 option puts all of the nodes on the same line as the job ID. In combination with -f, attributes are not folded to fit in a terminal window. This is intended to ease the parsing of the qstat output.

-s

In addition to the basic information, any comment provided by the batch administrator or scheduler is shown.

-G

Show size information in giga-bytes.

-M

Show size information, disk or memory in mega-words. A word is considered to be 8 bytes.

-R

In addition to other information, disk reservation information is shown. Not applicable to all systems.

-u

Job status is displayed in the alternative format. If an operand is a job id, status for that job is displayed. For a destination id operand, status for jobs at that destination which are owned by the user(s) listed in user_list are displayed. 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. "*.nasa.gov". User_name without a "@host" is equivalent to "user_name@*", that is at any host.

-Q

Specifies that the request is for queue status and that the operands are destination identifiers.

-q

Specifies that the request is for queue status which should be shown in the alternative format.

-B

Specifies that the request is for batch server status and that the operands are the names of servers.

OPERANDS

If neither the -Q nor the -B option is given, the operands on the qstat command must be either job identifiers or destinations identifiers.

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 is 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.

If the -Q option is given, the operands are destination identifiers as specified above. If queue is specified, the status of that queue at the default server will be given. If queue@server is specified, the status of the named queue at the named server will be given. If @server is specified, the status of all queues at the named server will be given. If no destination is specified, the status of all queues at the default server will be given.

If the -B option is given, the operand is the name of a server.

STANDARD OUTPUT

Displaying Job Status

If job status is being displayed in the default format and the -f option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job name given by the submitter.
- the job owner
- he CPU time used
- the job state:
    - C

      Job is completed after having run

E	Job is exiting after having run.
H	Job is held.
Q	job is queued, eligible to run or routed.
R	job is running.
T	job is being moved to new location.
W	job is waiting for its execution time (-a option) to be reached.
S	(Unicos only) job is suspend.

- the queue in which the job resides

If job status is being displayed and the -f option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See the configuration section for details. If Tcl is not being used, full display for each job consists of the header line:

```
Job Id: job identifier
```

Followed by one line per job attribute of the form:

```
attribute_name = value
```

If any of the options -a, -i, -r, -u, -n, -s, -G or -M are provided, the alternative display format for jobs is used. The following items are displayed on a single line, in the specified order, separated by white space:

- the job identifier assigned by PBS.
- the job owner.
- The queue in which the job currently resides.
- The job name given by the submitter.
- The session id (if the job is running).
- The number of nodes requested by the job.

- The number of cpus or tasks requested by the job.
- The amount of memory requested by the job.
- Either the cpu time, if specified, or wall time requested by the job, (hh:mm).
- The jobs current state.
- The amount of cpu time or wall time used by the job (hh:mm).

If the -R option is provided, the line contains:

- the job identifier assigned by PBS.
- the job owner.
- The queue in which the job currently resides.
- The number of nodes requested by the job.
- The number of cpus or tasks requested by the job.
- The amount of memory requested by the job.
- Either the cpu time or wall time requested by the job.
- The jobs current state.
- The amount of cpu time or wall time used by the job.
- The amount of SRFS space requested on the big file system.
- The amount of SRFS space requested on the fast file system.
- The amount of space requested on the parallel I/O file system.

The last three fields may not contain useful information at all sites or on all systems.

Displaying Queue Status

If queue status is being displayed and the -f option was not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the queue name
- the maximum number of jobs that may be run in the queue concurrently
- the total number of jobs in the queue
- the enable or disabled status of the queue
- the started or stopped status of the queue
- for each job state, the name of the state and the number of jobs in the queue in that state.
- the type of queue, execution or routing.

If queue status is being displayed and the -f option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See the configuration section for details. If Tcl is not being used, the full display for each queue consists of the header line:

```
Queue: queue_name
```

Followed by one line per queue attribute of the form:

```
attribute_name = value
```

If the -q option is specified, queue information is displayed in the alternative format: The following information is displayed on a single line:

- the queue name
- the maximum amount of memory a job in the queue may request
- the maximum amount of cpu time a job in the queue may request
- the maximum amount of wall time a job in the queue may request
- the maximum amount of nodes a job in the queue may request
- the number of jobs in the queue in the running state
- the number of jobs in the queue in the queued state
- the maximum number (limit) of jobs that may be run in the queue concurrently
- the state of the queue given by a pair of letters:
  - either the letter E if the queue is Enabled or D if Disabled
  - and
  - either the letter R if the queue is Running (started) or S if Stopped.

Displaying Server Status

If batch server status is being displayed and the -f option is not specified, the following items are displayed on a single line, in the specified order, separated by white space:

- the server name
- the maximum number of jobs that the server may run concurrently
- the total number of jobs currently managed by the server
- the status of the server
- for each job state, the name of the state and the number of jobs in the server in that state

If server status is being displayed and the -f option is specified, the output will depend on whether qstat was compiled to use a Tcl interpreter. See the configuration section for details. If Tcl is not being used, the full display for the server consist of the header line:

```
Server: server name
```

Followed by one line per server attribute of the form:

```
attribute_name = value
```

STANDARD ERROR

The qstat command will write a diagnostic message to standard error for each error occurrence.

CONFIGURATION

If qstat is compiled with an option to include a Tcl interpreter, using the -f flag to get a full display causes a check to be made for a script file to use to output the requested information. The first location checked is $HOME/.qstatrc. If this does not exist, the next location checked is administrator configured. If one of these is found, a Tcl interpreter is started and the script file is passed to it along with three global variables. The command line arguments are split into two variable named flags and operands . The status information is passed in a variable named objects . All of these variables are Tcl lists. The flags list contains the name of the command (usually "qstat") as its first element. Any other elements are command line option flags with any options they use, presented in the order given on the command line. They are broken up individually so that if two flags are given together on the command line, they are separated in the list. For example, if the user typed:

```
qstat -QfWbigdisplay
```

the flags list would contain

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

The operands list contains all other command line arguments following the flags. There will always be at least one element in operands because if no operands are typed by the user, the default destination or server name is used. The objects list contains all the information retrieved from the server(s) so the Tcl interpreter can run once to format the entire output. This list has the same number of elements as the operands list. Each element is another list with two elements.

The first element is a string giving the type of objects to be found in the second. The string can take the values "server", "queue", "job" or "error".

The second element will be a list in which each element is a single batch status object of the type given by the string discussed above. In the case of "error", the list will be empty. Each object is again a list. The first element is the name of the object. The second is a list of attributes.

The third element will be the object text.

All three of these object elements correspond with fields in the structure batch_status which is described in detail for each type of object by the man pages for pbs_statjob(3), pbs_statque(3), and pbs_statserver(3). Each attribute in the second element list whose elements correspond with the attrl structure. Each will be a list with two elements. The first will be the attribute name and the second will be the attribute value.

EXIT STATUS

Upon successful processing of all the operands presented to the qstat command, the exit status will be a value of zero.

If the qstat command fails to process any operand, the command exits with a value greater than zero.

See Also:

- qalter(1B)
- qsub(1B)
- **pbs_alterjob(3B)**
- **pbs_statjob(3B)**
- **pbs_statque(3B)**
- **pbs_statserver(3B)**
- **pbs_submit(3B)**
- **pbs_job_attributes(7B)**
- **pbs_queue_attributes(7B)**
- **pbs_server_attributes(7B)**
- **qmgr** query_other_jobs parameter (allow non-admin users to see other users' jobs
- **pbs_resources_*(7B)** where * is system type
- **PBS ERS**

qsub

submit pbs job

SYNOPSIS

```
qsub [-a date_time] [-A account_string] [-b secs] [-c checkpoint_options]
       [-C directive_prefix] [-d path] [-D path] [-e path] [-h]
       [-I ] [-j  join ] [-k   keep ] [-l resource_list ]
       [-m  mail_options] [-M  user_list] [-N  name] [-o path]
       [-p  priority] [-q  destination] [-r c] [-S  path_list]
       [-t  array_request] [-u  user_list]
       [-v  variable_list] [-V ] [-W additional_attributes] [-X] [-z]
[script]
```

DESCRIPTION

To create a job is to submit an executable script to a batch server. The batch server will be the default server unless the -q option is specified. See discussion of PBS_DEFAULT under Environment Variables below. Typically, the script is a shell script which will be executed by a command shell such as sh or csh.

Options on the qsub command allow the specification of attributes which affect the behavior of the job.

The qsub command will pass certain environment variables in the Variable_List attribute of the job. These variables will be available to the job. The value for the following variables will be taken from the environment of the qsub command: HOME, LANG, LOGNAME, PATH, MAIL, SHELL, and TZ. These values will be assigned to a new name which is the current name prefixed with the string "PBS_O_". For example, the job will have access to an environment variable named PBS_O_HOME which have the value of the variable HOME in the qsub command environment.

In addition to the above, the following environment variables will be available to the batch job.

PBS_O_HOST
: the name of the host upon which the qsub command is running.

PBS_SERVER
: the hostname of the pbs_server which qsub submits the job to.

PBS_O_QUEUE
: the name of the original queue to which the job was submitted.

PBS_O_WORKDIR
: the absolute path of the current working directory of the qsub command.

PBS_ARRAYID
: each member of a job array is assigned a unique identifier (see -t).

PBS_ENVIRONMENT
: set to PBS_BATCH to indicate the job is a batch job, or to PBS_INTERACTIVE to indicate the job is a PBS interactive job, see -I option.

PBS_JOBID
: the job identifier assigned to the job by the batch system.

PBS_JOBNAME
: the job name supplied by the user.

PBS_NODEFILE
: the name of the file contain the list of nodes assigned to the job (for parallel and cluster systems).

PBS_QUEUE
: the name of the queue from which the job is executed.

OPTIONS

-a *date_time*

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 -a 1110, the job will be eligible to run at 11:10am tomorrow.

-A *account_string*

Defines the account string associated with the job. The account_string is an undefined string of characters and is interpreted by the server which executes the job. See section 2.7.1 of the PBS ERS.

-b *seconds*

Defines the maximum number of seconds qsub will block attempting to contact pbs_server. If pbs_server is down, or for a variety of communication failures, qsub will continually retry connecting to pbs_server for job submission. This value overrides the CLIENTRETRY parameter in torque.cfg. This is a non-portable TORQUE extension. Portability-minded users can use the PBS_CLIENTRETRY environmental variable. A negative value is interpreted as infinity. The default is 0.

-c *checkpoint_options*

Defines the options that will apply to the job. If the job executes upon a host which does not support checkpoint, these options will be ignored.

Valid checkpoint options are:

none

No checkpointing is to be performed.

enabled

Specify that checkpointing is allowed but must be explicitly invoked by either the qhold or qchkpt commands.

shutdown

Specify that checkpointing is to be done on a job at pbs_mom shutdown.

periodic

Specify that periodic checkpointing is enabled. The default interval is 10 minutes and can be changed by the $checkpoint_interval option in the mom config file or by specifying an interval when the job is submitted

interval=minutes

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

depth=number

Specify a number (depth) of checkpoint images to be kept in the checkpoint directory.

dir=path

Specify a checkpoint directory (default is /var/spool/torque/checkpoint).

-C *directive_prefix*

Defines the prefix that declares a directive to the qsub command within the script file. See the paragraph on script directives in the Extended Description section.

If the -C option is presented with a directive_prefix argument that is the null string, qsub will not scan the script file for directives.

-d *path*

Defines the working directory path to be used for the job. If the -d option is not specified, the default working directory is the home directory. This option sets the environment variable PBS_O_INITDIR.

-D *path*

Defines the root directory to be used for the job. This option sets the environment variable PBS_O_ROOTDIR.

-e *path*

Defines the path to be used for the standard error stream of the batch job. 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 in the syntax recognized by POSIX. The argument will be interpreted as follows:

path_name

Where path_name is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component.

hostname:path_name

Where path_name is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard error, the path name will be expanded relative to the users home directory on the hostname system.

path_name

Where path_name specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname.

hostname:path_name

Where path_name specifies an absolute path name, the path will be used as specified.

If the -e option is not specified, the default file name for the standard error stream will be used. The default name has the following form:

```
job_name.esequence_number
```

where job_name is the name of the job, see -N option, and sequence_number is the job number assigned when the job is submitted.

-h

Specifies that a user hold be applied to the job at submission time.

-I

Declares that the job is to be run "interactively". The job will be queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected through qsub to the terminal session in which qsub is running. Interactive jobs are forced to not rerunable. See the "Extended Description" paragraph for addition information of interactive jobs.

-j *join*

Declares if the standard error stream of the job will be merged with the standard output stream of the job.

An option argument value of oe directs that the two streams will be merged, intermixed, as standard output. An option 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.

-k *keep*

Defines which (if either) of standard output or standard error will be retained on the execution host. If set for a stream, this option overrides the path name for that stream. If not set, neither stream is retained on the execution host.

The argument is either the single letter "e" or "o", or the letters "e" and "o" combined in either order. Or the argument is the letter "n".

e

The standard error stream is to retained on the execution host. The stream will be placed in the home directory of the user under whose user id the job executed. The file name will be the default file name given by:

job_name.esequence

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

o

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

job_name.osequence

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

eo

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

oe

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

n

Neither stream is retained.

-l *resource_list*

Defines the resources that are required by the job and establishes a limit to the amount of resource that can be consumed. If not set for a generally available resource, such as CPU time, the limit is infinite. The resource_list argument is of the form:

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

-m *mail_options*

Defines the set of conditions under which the execution server will send a mail message about the job. The mail_options argument is a string which consists of either the single character "n", or one or more of the characters "a", "b", and "e".

If the character "n" is specified, no mail will be sent.

For the letters "a", "b", and "e":

a

mail is sent when the job is aborted by the batch system.

b

mail is sent when the job begins execution.

e

mail is sent when the job terminates.

If the -m option is not specified, mail will be sent if the job is aborted.

-M *user_list*

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.

-N *name*

Declares a name for the job. The name specified may be up to and including 15 characters in length. It must consist of printable, non white space characters with the first character alphabetic.

If the -N option is not specified, the job name will be the base name of the job script file specified on the command line. If no script file name was specified and the script was read from the standard input, then the job name will be set to STDIN.

-o *path*

Defines the path to be used for the standard output stream of the batch job. 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 in the syntax recognized by POSIX. The argument will be interpreted as follows:

path_name

Where path_name is not an absolute path name, then the qsub command will expand the path name relative to the current working directory of the command. The command will supply the name of the host upon which it is executing for the hostname component.

hostname:path_name

Where path_name is not an absolute path name, then the qsub command will not expand the path name relative to the current working directory of the command. On delivery of the standard output, the path name will be expanded relative to the users home directory on the hostname system.

path_name

Where path_name specifies an absolute path name, then the qsub will supply the name of the host on which it is executing for the hostname.

hostname:path_name

Where path_name specifies an absolute path name, the path will be used as specified.

If the -o option is not specified, the default file name for the standard output stream will be used. The default name has the following form:

```
job_name.osequence_number
```

where job_name is the name of the job, see -N option, and sequence_number is the job number assigned when the job is submitted.

-p *priority*

Defines the priority of the job. The priority argument must be a integer between -1024 and +1023 inclusive. The default is no priority which is equivalent to a priority of zero.

-q *destination*

Defines the destination of the job. The destination names a queue, a server, or a queue at a server.

The qsub command will submit the script to the server defined by the destination argument. If the destination is a routing queue, the job may be routed by the server to a new destination.

If the -q option is not specified, the qsub command will submit the script to the default server. See PBS_DEFAULT under the Environment Variables section on this man page and the PBS ERS section 2.7.4, "Default Server".

If the -q option is specified, it is in one of the following three forms:

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

If the destination argument names a queue and does not name a server, the job will be submitted to the named queue at the default server.

If the destination argument names a server and does not name a queue, the job will be submitted to the default queue at the named server.

If the destination argument names both a queue and a server, the job will be submitted to the named queue at the named server.

-r *y|n*
Declares whether the job is rerunable. See the qrerun command. The option argument is a single character, either y or n.dd

If the argument is "y", the job is rerunable. If the argument is "n", the job is not rerunable. The default value is y, rerunable.

-S *path_list*
Declares the shell that interprets the job script.

The option argument path_list is in the form:

```
path[@host][,path[@host],...]
```

Only one path may be specified for any host named. 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, the execution will use the users login shell on the execution host.

-t *array_request*
Specifies the task ids of a job array. Single task arrays are allowed.

The array_request argument is an integer id or a range of integers. Multiple ids or id ranges can be combined in a comma delimted list. Examples : -t 1-100 or -t 1,10,50-100

-u *user_list*

Defines the user name under which the job is to run on the execution system.

The user_list argument is of the form:

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

Only one user name may be given per specified host. Only one of the user specifications may be supplied without the corresponding host specification. That user name will used for execution on any host not named in the argument list. If unset, the user list defaults to the user who is running qsub.

-v *variable_list*

Expands the list of environment variables that are exported to the job.

In addition to the variables described in the "Description" section above, 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.

-V

Declares that all environment variables in the qsub commands environment are to be exported to the batch job.

-W *additional_attributes*

The -W option allows for the specification of additional job attributes. The general syntax of the -W is in the form:

```
-W attr_name=attr_value[,attr_name=attr_value...]
```

Note if white space occurs anywhere within the option argument string or the equal sign, "=", occurs within an attribute_value string, then the string must be enclosed with either single or double quote marks.

PBS currently supports the following attributes within the -W option.

depend=dependency_list

Defines the dependency between this and other jobs. The dependency_list is in the form:

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

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

synccount:count

This job is the first in a set of jobs to be executed at the same time. Count is the number of additional jobs in the set.

syncwith:jobid

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

after:jobid[:jobid...]

This job may be scheduled for execution at any point after jobs jobid have started execution.

afterok:jobid[:jobid...]

This job may be scheduled for execution only after jobs jobid have terminated with no errors. See the csh warning under "Extended Description".

afternotok:jobid[:jobid...]

This job may be scheduled for execution only after jobs jobid have terminated with errors. See the csh warning under "Extended Description".

afterany:jobid[:jobid...]

This job may be scheduled for execution after jobs jobid have terminated, with or without errors.

on:count

This job may be scheduled for execution after count dependencies on other jobs have been satisfied. This form is used in conjunction with one of the before forms, see below.

before:jobid[:jobid...]

When this job has begun execution, then jobs jobid... may begin.

beforeok:jobid[:jobid...]

If this job terminates execution without errors, then jobs jobid... may begin. See the csh warning under "Extended Description".

beforenotok:jobid[:jobid...]

If this job terminates execution with errors, then jobs jobid... may begin. See the csh warning under "Extended Description".

beforeany:jobid[:jobid...]

When this job terminates execution, jobs jobid... may begin.

If any of the before forms are used, the jobs referenced by jobid must have been submitted with a dependency type of on.

If any of the before forms are used, the jobs referenced by jobid must have the same owner as the job being submitted. Otherwise, the dependency is ignored.

Error processing of the existence, state, or condition of he 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.

Dependency examples:

- qsub -W depend=afterok:123.big.iron.com /tmp/script
- qsub -W depend=before:234.hunk1.com:235.hunk1.com
- /tmp/script

group_list=g_list

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.

interactive=true

If the interactive attribute is specified, the job is an interactive job. The -I option is a alternative method of specifying this attribute.

stagein=file_list
stageout=file_list

Specifies which files are staged (copied) in before job start or staged out after the job completes execution. On completion of the job, all staged-in and staged-out files are removed from the execution system. The file_list is in the form:

local_file@hostname:remote_file[,...]

regardless of the direction of the copy. The name local_file is the name of the file on the system where the job executed. 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 users home directory on the destination host. The use of wildcards in the file name is not recommended. The file names map to a remote copy program (rcp) call on the execution system in the follow manner:

- For stagein: rcp hostname:remote_file local_file
- For stageout: rcp local_file hostname:remote_file

Data staging examples:

```
-W stagein=/tmp/input.txt@headnode:/home/user/input.txt
-W stageout=/tmp/output.txt@headnode:/home/user/output.txt
```

If TORQUE has been compiled with wordexp support, then variables can be used in the specified paths. Currently only $PBS_JOBID, $HOME, and $TMPDIR are supported for stagein.

umask=XXX

Sets umask used to create stdout and stderr spool files in pbs_mom spool directory. Values starting with 0 are treated as octal values, otherwise the value is treated as a decimal umask value.

-X

Enables X11 forwarding. The DISPLAY environment variable must be set.

-z

Directs that the qsub command is not to write the job identifier assigned to the job to the commands standard output.

OPERANDS

The qsub command accepts a script operand that is the path to the script of the job. If the path is relative, it will be expanded relative to the working directory of the qsub command.

If the script operand is not provided or the operand is the single character "-", the qsub command reads the script from standard input. When the script is being read from Standard Input, qsub will copy the file to a temporary file. This temporary file is passed to the library interface routine pbs_submit. The temporary file is removed by qsub after pbs_submit returns or upon the receipt of a signal which would cause qsub to terminate.

STANDARD INPUT

The qsub command reads the script for the job from standard input if the script operand is missing or is the single character "-".

INPUT FILES

The script file is read by the qsub command. Qsub acts upon any directives found in the script.

When the job is created, a copy of the script file is made and that copy cannot be modified.

STANDARD OUTPUT

Unless the -z option is set, the job identifier assigned to the job will be written to standard output if the job is successfully created.

STANDARD ERROR

The qsub command will write a diagnostic message to standard error for each error occurrence.

ENVIRONMENT VARIABLES

The values of some or all of the variables in the qsub commands environment are exported with the job, see the -v and -V options.

The environment variable PBS_DEFAULT defines the name of the default server. Typically, it corresponds to the system name of the host on which the server is running. If PBS_DEFAULT is not set, the default is defined by an administrator established file.

The environment variable PBS_DPREFIX determines the prefix string which identifies directives in the script.

The environment variable PBS_CLIENTRETRY defines the maximum number of seconds qsub will block. See the -b option above. Despite the name, currently qsub is the only client that supports this option.

TORQUE.CFG

The torque.cfg file, located in PBS_SERVER_HOME (/var/spool/torque by default) controls the behavior of the qsub command. This file contains a list of parameters and values separated by whitespace

QSUBSLEEP takes an integer operand which specifies time to sleep when running qsub command. Used to prevent users from overwhelming the scheduler.

SUBMITFILTER specifies the path to the submit filter used to pre-process job submission. The default path is $(libexecdir)/qsub_filter, which falls back to /usr/local/sbin/torque_submitfilter for backwards compatibility. This torque.cfg parameter overrides this default.

SERVERHOST

QSUBHOST

QSUBSENDUID

XAUTHPATH

CLIENTRETRY

VALIDATEGROUP

DEFAULTCKPT

VALIDATEPATH

RERUNNABLEBYDEFAULT this parameter specifies if a job is rerunnable by default. The default is true, setting this to false causes the rerunnable attribute value to be false unless the users specifies otherwise with the -r option.

For example:

- QSUBSLEEP 2
- RERUNNABLEBYDEFAULT false

EXTENDED DESCRIPTION

Script Processing:

A job script may consist of PBS directives, comments and executable statements. A PBS directive provides a way of specifying job attributes in addition to the command line options. For example:

```
:
#PBS -N Job_name
#PBS -l walltime=10:30,mem=320kb
#PBS -m be
#
step1 arg1 arg2
step2 arg3 arg4
```

The qsub command scans the lines of the script file for directives. An initial line in the script that begins with the characters "#!" or the character ":" will be ignored and scanning will start with the next line. Scanning will continue until the first executable line, that is a line that is not blank, not a directive line, nor a line whose first non white space character is "#". If directives occur on subsequent lines, they will be ignored.

A line in the script file will be processed as a directive to qsub if and only if the string of characters starting with the first non white space character on the line and of the same length as the directive prefix matches the directive prefix.

The remainder of the directive line consists of the options to qsub in the same syntax as they appear on the command line. The option character is to be preceded with the "-" character.

If an option is present in both a directive and on the command line, that option and its argument, if any, will be ignored in the directive. The command line takes precedence.

If an option is present in a directive and not on the command line, that option and its argument, if any, will be processed as if it had occurred on the command line.

The directive prefix string will be determined in order of preference from:

The value of the -C option argument if the option is specified on the command line.

The value of the environment variable PBS_DPREFIX if it is defined.

The four character string #PBS.

If the -C option is found in a directive in the script file, it will be ignored.

User Authorization:

When the user submits a job from a system other than the one on which the PBS Server is running, the name under which the job is to be executed is selected according to the rules listed under the -u option. The user submitting the job must be authorized to run the job under the execution user name. This authorization is provided if:

- The host on which qsub is run is trusted by the execution host (see /etc/hosts.equiv)
- The execution user has an .rhosts file naming the submitting user on the submitting host.

C-Shell .logout File:

The following warning applies for users of the c-shell, csh. If the job is executed under the csh and a .logout file exists in the home directory in which the job executes, the exit status of the job is that of the .logout script, not the job script. This may impact any inter-job dependencies. To preserve the job exit status, either remove the .logout file or place the following line as the first line in the .logout file

```
set EXITVAL = $status
```

and the following line as the last executable line in .logout

```
exit $EXITVAL
```

Interactive Jobs:

If the -I option is specified on the command line or in a script directive, or if the "interactive" job attribute declared true via the -W option, -W interactive=true, either on the command line or in a script directive, the job is an interactive job. The script will be processed for directives, but will not be included with the job. When the job begins execution, all input to the job is from the terminal session in which qsub is running.

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

One 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 escaped by qsub. The recognized escape sequences are:

~.

Qsub terminates execution. The batch job is also terminated.

~susp

Suspend the qsub program if running under the C shell. "susp" is the suspend character, usually CNTL-Z.

~asusp

Suspend the input half of qsub (terminal to job), but allow output to continue to be displayed. Only works under the C shell. "asusp" is the auxiliary suspend character, usually CNTL-Y.

EXIT STATUS

Upon successful processing, the qsub exit status will be a value of zero.

If the qsub command fails, the command exits with a value greater than zero.

SEE ALSO

```
qalter(1B), qdel(1B), qhold(1B), qmove(1B), qmsg(1B), qrerun(1B),
qrls(1B), qselect(1B), qsig(1B), qstat(1B), pbs_connect(3B),
pbs_job_attributes(7B),  pbs_queue_attributes(7B),
pbs_resources_irix5(7B), pbs_resources_sp2(7B),
pbs_resources_sunos4(7B), pbs_resources_unicos8(7B),
pbs_server_attributes(7B), and pbs_server(8B)

qterm(8B)                            PBS
qterm(8B)



NAME
       qterm - terminate processing by a pbs batch server

SYNOPSIS
       qterm [-t type] [server...]

DESCRIPTION
       The  qterm command terminates a batch server.  When a server receives
a
       terminate command, The server will go into Terminating state.   No
new
       jobs will be allowed to be started into execution nor enqueued into
the
       server.  The impact on jobs currently being run by the  server
depends
       on  the type of shut down requested as described below.  The qterm
com-
       mand will not exit until the server has completed  it  shutdown
proce-
       dure.

       In  order to execute qterm, the user must have PBS Operation or
Manager
       privilege.

OPTIONS
       -t type   Specifies the type of shut down.  The types are:

                 immediate
```

All running jobs are to immediately stop execution. If checkpoint is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If checkpoint is not supported or the job cannot be checkpointed, running jobs are requeued if the rerunable attribute is true. Otherwise, jobs are killed.

Normally the server will not shutdown until there are no jobs in the running state. If the server is unable to contact the MOM of running job, the job is still listed as running. The server may be forced down by a second qterm -t immediate command.

delay

If checkpoint is supported, running jobs that can be checkpointed are checkpointed, terminated, and requeued. If a job cannot be checkpointed, but can be rerun, the job is terminated and requeued. Otherwise, running jobs are allowed to continue to run. Note, the operator or administrator may use the qrerun and qdel commands to remove running jobs.

quick

This option is used when you wish that running jobs be left running when the server shuts down. The server will cleanly shutdown and can be restarted when desired. Upon restart of the server, jobs that continue to run are shown as running; jobs that terminated during the server’s absence will be placed into the exiting state.

This is the default action if the -t option is not specified.

OPERANDS

```
       The  server  operand  specifies  which  servers are to shutdown.  If
no
       servers are given, then the default server will be terminated.

STANDARD ERROR
       The qterm command will write a diagnostic message to standard error
for
       each error occurrence.

EXIT STATUS
       Upon  successful  processing of all the operands presented to the
qterm
       command, the exit status will be a value of zero.

       If the qterm command fails to process any operand,  the  command
exits
       with a value greater than zero.

SEE ALSO
       pbs_server(8B),            qmgr(8B),
pbs_resources_aix4(7B),
       pbs_resources_irix5(7B),
pbs_resources_sp2(7B),
       pbs_resources_sunos4(7B), and  pbs_resources_unicos8(7B)



Local
qterm(8B)
```

TORQUE Resource Manager qtracejob sample output

qtracejob

```
root@headnode# qtracejob.pbs-server 100015

headnode    05/03/2006 10:13:32 A queue=batch
headnode    05/03/2006 10:13:32 A queue=parallel
headnode    05/03/2006 10:13:32 S enqueuing into batch, state 1 hop 1
headnode    05/03/2006 10:13:32 S dequeuing from batch, state 1
headnode    05/03/2006 10:13:32 S enqueuing into parallel, state 1 hop 1
headnode    05/03/2006 10:13:32 S Job Queued at request of
user13@login1.univ.edu, owner = user13@login1.univ.edu, job name = test,
queue = parallel
node245  05/03/2006 10:59:38 M Job Modified at request of
PBS_Server@headnode-node.ten.univ.edu
headnode    05/03/2006 10:59:38 A user=user13 group=group13 jobname=test
queue=parallel ctime=1146665612 qtime=1146665612 etime=1146665612
start=1146668378 exec_host=node245/0+node241/0+node240/0+node239/0+node238/0
+node237/0+node236/0+node235/0+node234/0+node231/0+node229/0+node226/0+node22
1/0
+node220/0+node217/0+node216/0
Resource_List.neednodes=node245+node241+node240
```

```
+node239+node238+node237+node236+node235+node234+node231+node229+node226+node
221
+node220+node217+node216 Resource_List.nodect=16
Resource_List.nodes=16:ppn=1 Resource_List.walltime=28:35:00
headnode    05/03/2006 10:59:38 S Job Modified at request of
root@headnode.univ.edu
headnode    05/03/2006 10:59:38 S Job Run at request of
root@headnode.univ.edu
headnode    05/03/2006 10:59:38 S Job Modified at request of
root@headnode.univ.edu
node216  05/03/2006 10:59:46 M JOIN JOB as node 15
node220  05/03/2006 10:59:46 M JOIN JOB as node 13
node226  05/03/2006 10:59:46 M JOIN JOB as node 11
node229  05/03/2006 10:59:46 M JOIN JOB as node 10
node231  05/03/2006 10:59:46 M JOIN JOB as node 9
node236  05/03/2006 10:59:46 M JOIN JOB as node 6
node238  05/03/2006 10:59:46 M JOIN JOB as node 4
node240  05/03/2006 10:59:46 M JOIN JOB as node 2
node221  05/03/2006 10:59:47 M JOIN JOB as node 12
node234  05/03/2006 10:59:47 M JOIN JOB as node 8
node235  05/03/2006 10:59:47 M JOIN JOB as node 7
node237  05/03/2006 10:59:47 M JOIN JOB as node 5
node239  05/03/2006 10:59:47 M JOIN JOB as node 3
node241  05/03/2006 10:59:47 M JOIN JOB as node 1
node217  05/03/2006 10:59:48 M JOIN JOB as node 14
node245  05/03/2006 10:59:50 M Started, pid = 19582
node245  05/03/2006 11:37:15 M start_process: task started, tid 2, sid
20048, cmd /bin/sh
node220  05/03/2006 11:37:21 M start_process: task started, tid 15, sid
27145, cmd /bin/sh
node240  05/03/2006 11:37:22 M start_process: task started, tid 4, sid
26719, cmd /bin/sh
node216  05/03/2006 11:37:23 M start_process: task started, tid 17, sid
20811, cmd /bin/sh
node221  05/03/2006 11:37:23 M start_process: task started, tid 14, sid
20231, cmd /bin/sh
node226  05/03/2006 11:37:23 M start_process: task started, tid 13, sid
25681, cmd /bin/sh
node234  05/03/2006 11:37:23 M start_process: task started, tid 10, sid
23538, cmd /bin/sh
node236  05/03/2006 11:37:23 M start_process: task started, tid 8, sid
23733, cmd /bin/sh
node238  05/03/2006 11:37:23 M start_process: task started, tid 6, sid
2550, cmd /bin/sh
node217  05/03/2006 11:37:24 M start_process: task started, tid 16, sid
22618, cmd /bin/sh
node229  05/03/2006 11:37:24 M start_process: task started, tid 12, sid
21030, cmd /bin/sh
node231  05/03/2006 11:37:24 M start_process: task started, tid 11, sid
23731, cmd /bin/sh
node235  05/03/2006 11:37:24 M start_process: task started, tid 9, sid
19893, cmd /bin/sh
node237  05/03/2006 11:37:24 M start_process: task started, tid 7, sid
23218, cmd /bin/sh
node239  05/03/2006 11:37:24 M start_process: task started, tid 5, sid
32213, cmd /bin/sh
node241  05/03/2006 11:37:24 M start_process: task started, tid 3, sid
```

```
28998, cmd /bin/sh
```

pbs_mom

start a pbs batch execution mini-server

Synopsis

```
pbs_mom [-C chkdirectory] [-c config] [-d directory] [-h hostname]
        [-L logfile] [-M MOMport] [-R RPPport] [-p|-r] [-x]
```

Description

The pbs_mom command starts the operation of a batch Machine Oriented Mini-server, MOM, on the local host. Typically, this command will be in a local boot file such as /etc/rc.local. To insure that the pbs_mom command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

One function of pbs_mom is to place jobs into execution as directed by the server, establish resource usage limits, monitor the jobs usage, and notify the server when the job completes. If they exist, pbs_mom will execute a prologue script before executing a job and an epilogue script after executing the job. The next function of pbs_mom is to respond to resource monitor requests. This was done by a separate pro cess in previous versions of PBS but has now been combined into one process. The resource monitor function is provided mainly for the PBS scheduler. It provides information about the status of running jobs, memory available etc. The next function of pbs_mom is to respond to task manager requests. This involves communicating with running tasks over a tcp socket as well as communicating with other MOMs within a job (a.k.a. a "sisterhood").

pbs_mom will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the mom_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

Options

Flag	Name	Description
-C	chkdirectory	Specifies The path of the directory used to hold checkpoint files. [Currently this is only valid on Cray systems.] The default directory is PBS_HOME/spool/checkpoint, see the -d option. The directory specified with the -C option must be owned by root and accessible (rwx) only by root to protect the security of the checkpoint files.
-c	config	Specify a alternative configuration file, see description below. If this is a relative file name it will be relative to PBS_HOME/mom_priv, see the -d option. If the specified file cannot be opened, pbs_mom will abort. If the -c option is not supplied, pbs_mom will attempt to open the default configuration file "config" in PBS_HOME/mom_priv. If this file is not present, pbs_mom will log the fact and continue.

-h	hostnames	Set MOMs hostname. This can be useful on multi-homed networks.
-d	directory	Specifies the path of the directory which is the home of the servers working files, PBS_HOME. This option is typically used along with -M when debugging MOM. The default directory is given by $PBS_SERVER_HOME which is typically /usr/spool/PBS.
-L	logfile	Specify an absolute path name for use as the log file. If not specified, MOM will open a file named for the current date in the PBS_HOME/mom_logs directory, see the -d option.
-M	port	Specifies the port number on which the mini-server (MOM) will listen for batch requests.
-R	port	Specifies the port number on which the mini-server (MOM) will listen for resource monitor requests, task manager requests and inter-MOM messages. Both a UDP and a TCP port of this number will be used.
-p	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. On any restart of MOM, the new mini-server will not be the parent of any running jobs, MOM has lost control of her offspring (not a new situation for a mother). With the -p option, MOM will allow the jobs to continue to run and monitor them indirectly via polling. This flag is redundant in that this is the default behavior when starting the server. The -p option is mutually exclusive with the -r and -q options.
-q	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -q option, MOM will allow the processes belonging to jobs to continue to run, but will not attempt to monitor them. The -q option is mutually exclusive with the -p and -r options.
-r	n/a	Specifies the impact on jobs which were in execution when the mini-server shut down. With the -r option, MOM will kill any processes belonging to jobs, mark the jobs as terminated, and notify the batch server which owns the job. The -r option is mutually exclusive with the -p and -q options. Normally the mini-server is started from the system boot file without the -p or the -r option. The mini-server will make no attempt to signal the former session of any job which may have been running when the mini-server terminated. It is assumed that on reboot, all processes have been killed. If the -r option is used following a reboot, process IDs (pids) may be reused and MOM may kill a process that is not a batch session.
-a	alarm	Used to specify the alarm timeout in seconds for computing a resource. Every time a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before the given time, an alarm signal is generated. The default is 5 seconds.
-x	n/a	Disables the check for privileged port resource monitor connections. This is used mainly for testing since the privileged port is the only mechanism used to prevent any ordinary user from connecting.

Configuration File

The configuration file may be specified on the command line at program start with the -c flag. The use of this file is to provide several types of run time information to pbs_mom: static resource names and values, external resources provided by a program to be run on request via a shell escape, and values to pass to internal set up functions at initialization (and re-initialization).

Each item type is on a single line with the component parts separated by white space. If the line starts with a hash mark (pound sign, #), the line is considered to be a comment and is skipped.

Static Resources

For static resource names and values, the configuration file contains a list of resource names/values pairs, one pair per line and separated by white space. An Example of static resource names and values could be the number of tape drives of different types and could be specified by:

- `tape3480 4`
- `tape3420 2`
- `tapedat 1`
- `tape8mm 1`

Shell Commands

If the first character of the value is an exclamation mark (!), the entire rest of the line is saved to be executed through the services of the system(3) standard library routine.

The shell escape provides a means for the resource monitor to yield arbitrary information to the scheduler. Parameter substitution is done such that the value of any qualifier sent with the query, as explained below, replaces a token with a percent sign (%) followed by the name of the qualifier. For example, here is a configuration file line which gives a resource name of "escape":

```
escape !echo %xxx %yyy
```

If a query for "escape" is sent with no qualifiers, the command executed would be `echo %xxx %yyy`. If one qualifier is sent, `escape[xxx=hi there]`, the command executed would be `echo hi there %yyy`. If two qualifiers are sent, `escape[xxx=hi][yyy=there]`, the command executed would be `echo hi there`. If a qualifier is sent with no matching token in the command line, `escape[zzz=snafu]`, an error is reported.

size[fs=<FS>]

Specifies that the available and configured disk space in the <FS> filesystem is to be reported to the pbs_server and scheduler. To request disk space on a per job basis, specify the file resource as in, `qsub -l nodes=1,file=1000kb`. For example, the available and configured disk space in the /localscratch filesystem will be reported:

```
size[fs=/localscratch]
```

Initialization Value

An initialization value directive has a name which starts with a dollar sign ($) and must be known to MOM via an internal table. The entries in this table now are:

- pbsserver
  - Defines hostnames running pbs_server that will be allowed to submit jobs, issue Resource Monitor (RM) requests, and get status updates. MOM will continually attempt to contact all server hosts for node status and state updates. Like $PBS_SERVER_HOME/server_name, the hostname may be followed by a colon and a port number. This parameter replaces the oft-confused $clienthost parameter from TORQUE 2.0.0p0 and earlier. Note that the hostname in $PBS_SERVER_HOME/server_name is used if no $pbsserver parameters are found.

- pbsclient
  - Causes a host name to be added to the list of hosts which will be allowed to connect to MOM as long as they are using a privilaged port for the purposes of resource monitor requests. For example, here are two configuration file lines which will allow the hosts "fred" and "wilma" to connect:

    ```
    $pbsclient fred
    $pbsclient wilma
    ```

    Two host name are always allowed to connection to pbs_mom, "localhost" and the name returned to pbs_mom by the system call gethostname(). These names need not be specified in the configuration file. The hosts listed as "clients" can issue Resource Monitor (RM) requests. Other MOM nodes and servers do not need to be listed as clients.

- restricted
  - Causes a host name to be added to the list of hosts which will be allowed to connect to MOM without needing to use a privilaged port. These names allow for wildcard matching. For example, here is a configuration file line which will allow queries from any host from the domain "ibm.com".

    ```
    $restricted *.ibm.com
    ```

    The restriction which applies to these connections is that only internal queries may be made. No resources from a config file will be found. This is to prevent any shell commands from being run by a non-root process. This parameter is generally not required except for some versions of OSX.

- logevent
  - Sets the mask that determines which event types are logged by pbs_mom. For example:

    ```
    $logevent 0x1fff $logevent 255
    ```

    The first example would set the log event mask to 0x1ff (511) which enables logging of all events including debug events. The second example would set the mask to 0x0ff (255) which enables all events except debug events.

- cputmult

  - Sets a factor used to adjust cpu time used by a job. This is provided to allow adjustment of time charged and limits enforced where the job might run on systems with different cpu performance. If Moms system is faster than the reference system, set cputmult to a decimal value greater than 1.0. If Moms system is slower, set cputmult to a value between 1.0 and 0.0. For example:

    ```
    $cputmult 1.5 $cputmult 0.75
    ```

- usecp
  - Specifies which directories should be staged with cp instead of rcp/scp. If a shared filesystem is available on all hosts in a cluster, this directive is used to make these filesystems known to MOM. For example, if /home is NFS mounted on all nodes in a cluster:

    ```
    $usecp *:/home /home
    ```

- wallmult
  - Sets a factor used to adjust wall time usage by to job to a common reference system. The factor is used for walltime calculations and limits the same as cputmult is used for cpu time.
- configversion
  - Specifies the version of the config file data, a string.
- check_poll_time
  - Specifies the MOM interval in seconds. MOM checks each job for updated resource usages, exited processes, over-limit conditions, etc. once per interval. This value should be equal or lower to pbs_servers job_stat_rate. High values result in stale information reported to pbs_server. Low values result in increased system usage by MOM. Default is 45 seconds.
- down_on_error
  - Causes MOM to report itself as state "down" to pbs_server in the event of a failed health check. This feature is EXPERIMENTAL and likely to be removed in the future. See HEALTH CHECK below.
- ideal_load
  - Ideal processor load. Represents a low water mark for the load average. Nodes that are currently busy will consider itself free after falling below ideal_load.
- auto_ideal_load
  - If jobs are running, sets idea_load based on a simple expression. The expressions start with the variable t (total assigned CPUs) or c (existing CPUs), an operator (+ - / *), and followed by a float constant.

    ```
    $auto_ideal_load t-0.2
    ```

- loglevel
  - Specifies the verbosity of logging with higher numbers specifying more verbose logging. Values may range between 0 and 7.

- log_file_max_size
    - If this is set to a value > 0 then pbs_mom will roll the current log file to log-file-name.1 when its size is greater than or equal to the value of log_file_max_size. This value is interpreted as kilo bytes.
- log_file_roll_depth
    - If this is set to a value >=1 and log_file_max_size is set then pbs_mom will continue rolling the log files to log-file-name.log_file_roll_depth.
- max_load
    - Maximum processor load. Nodes over this load average are considered busy (see ideal_load above).
- auto_max_load
    - If jobs are running, sets max_load based on a simple expression. The expressions start with the variable t (total assigned CPUs) or c (existing CPUs), an operator (+ - / *), and followed by a float constant.
- enablemomrestart
    - Enable automatic restarts of MOM. If enabled, MOM will check if its binary has been updated and restart itself at a safe point when no jobs are running; thus making upgrades easier. The check is made by comparing the mtime of the pbs_mom executable. Command-line args, the process name, and the PATH env variable are preserved across restarts. It is recommended that this not be enabled in the config file, but enabled when desired with momctl (see RESOURCES for more information.)
- node_check_script
    - Specifies the fully qualified pathname of the health check script to run (see HEALTH CHECK for more information).
- node_check_interval
    - Specifies when to run the MOM health check. The check can be either periodic, event-driver, or both. The value starts with an integer specifying the number of MOM intervals between subsequent executions of the specified health check. After the integer is an optional comma-separated list of event names. Currently supported are "jobstart" and "jobend". This value defaults to 1 with no events indicating the check is run every mom interval. (see HEALTH CHECK for more information)

```
$node_check_interval 0Disabled
$node_check_interval 0,jobstartOnly
$node_check_interval 10,jobstart,jobend
```

- prologalarm
    - Specifies maximum duration (in seconds) which the mom will wait for the job prolog or job job epilog to complete. This parameter defaults to 300 seconds (5 minutes).
- rcpcmd
    - Specify the the full path and argument to be used for remote file copies. This overrides the compile-time default found in configure. This must contain 2 words: the full path to the command and the switches. The copy command must be able to recursively copy

files to the remote host and accept arguments of the form "user@host:files" For example:

```
$rcpcmd /usr/bin/rcp -rp
$rcpcmd /usr/bin/scp -rpB
```

- remote_reconfig
  - Enables the ability to remotely reconfigure pbs_mom with a new config file. Default is disabled. This parameter accepts various forms of true, yes, and 1.
- timeout
  - Specifies the number of seconds before TCP messages will time out. TCP messages include job obituaries, and TM requests if RPP is disabled. Default is 60 seconds.
- tmpdir
  - Sets the directory basename for a per-job temporary directory. Before job launch, MOM will append the jobid to the tmpdir basename and create the directory. After the job exit, MOM will recursively delete it. The env variable TMPDIR will be set for all pro/epilog scripts, the job script, and TM tasks.

    Directory creation and removal is done as the job owner and group, so the owner must have write permission to create the directory. If the directory already exists and is owned by the job owner, it will not be deleted after the job. If the directory already exists and is NOT owned by the job owner, the job start will be rejected.
- status_update_time
  - Specifies (in seconds) how often MOM updates its status information to pbs_server. This value should correlate with the servers scheduling interval. High values increase the load of pbs_server and the network. Low values cause pbs_server to report stale information. Default is 45 seconds.
- varattr
  - This is similar to a shell escape above, but includes a TTL. The command will only be run every TTL seconds. A TTL of -1 will cause the command to be executed only once. A TTL of 0 will cause the command to be run every time varattr is requested. This parameter may be used multiple times, but all output will be grouped into a single "varattr" attribute in the request and status out put. If the command has no output, the name will be skipped in the output.

    ```
    $varattrseta
    $varattrsetb
    ```
- xauthpath
  - Specifies the path to the xauth binary to enable X11 fowarding.
- ignvmem
  - If set to true, then pbs_mom will ignore vmem/pvmem limit enforcement.
- ignwalltime

    - If set to true, then pbs_mom will ignore walltime limit enforcement.

- mom_host
    - Sets the local hostname as used by pbs_mom.

Resources

Resource Monitor queries can be made with momctls -q option to retrieve and set pbs_mom options. Any configured static resource may be retrieved with a request of the same name. These are resource requests not otherwise documented in the PBS ERS.

- cycle
    - Forces an immediate MOM cycle.
- status_update_time
    - Retrieve or set the $status_update_time parameter.
- check_poll_time
    - Retrieve or set the $check_poll_time parameter.
- configversion
    - Retrieve the config version.
- jobstartblocktime
    - Retrieve or set the $jobstartblocktime parameter.
- enablemomrestart
    - Retrieve or set the $enablemomrestart parameter.
- loglevel
    - Retrieve or set the $loglevel parameter.
- down_on_error
    - Retrieve or set the EXPERIMENTAL $down_on_error parameter.
- diag0 - diag4
    - Retrieves various diagnostic information.
- rcpcmd
    - Retrieve or set the $rcpcmd parameter.
- version
    - Retrieves the pbs_mom version.

Health Check

The health check script is executed directly by the pbs_mom daemon under the root user id. It must be accessible from the compute node and may be a script or compiled executable program. It may make any needed system calls and execute any combination of system utilities but should not execute resource manager client commands. Also, as of TORQUE 1.0.1, the pbs_mom daemon blocks until the health check is completed and does not possess a built-in timeout. Consequently, it is advisable to keep the launch script execution time short and verify that the script will not block even under failure conditions.

If the script detects a failure, it should return the keyword ERROR to stdout followed by an error message. The message (up to 256 characters) immediately following the ERROR string will be assigned to the node attribute message of the associated node.

If the script detects a failure when run from "jobstart", then the job will be rejected. This should probably only be used with advanced schedulers like Moab so that the job can be routed to another node.

TORQUE currently ignores ERROR messages by default, but advanced schedulers like moab can be configured to react appropriately.

If the experimental $down_on_error MOM setting is enabled, MOM will set itself to state down and report to pbs_server; and pbs_server will report the node as "down". Additionally, the experimental "down_on_error" server attribute can be enabled which has the same effect but moves the decision to pbs_server. It is redundant to have MOMs $down_on_error and pbs_servers down_on_error features enabled. See "down_on_error" in pbs_server_attributes(7B).

Files

- $PBS_SERVER_HOME/server_name
  - Contains the hostname running pbs_server.
- $PBS_SERVER_HOME/mom_priv
  - The default directory for configuration files, typically (/usr/spool/pbs)/mom_priv.
- $PBS_SERVER_HOME/mom_logs
  - Directory for log files recorded by the server.
- $PBS_SERVER_HOME/mom_priv/prologue
  - The administrative script to be run before job execution.
- $PBS_SERVER_HOME/mom_priv/epilogue
  - The administrative script to be run after job execution.

Signal Handling

pbs_mom handles the following signals:

- SIGHUP
    - Causes pbs_mom to re-read its configuration file, close and reopen the log file, and reinitialize resource structures.
- SIGALRM
    - Results in a log file entry. The signal is used to limit the time taken by certain children processes, such as the prologue and epilogue.
- SIGINT and SIGTERM
    - Results in pbs_mom exiting without terminating any running jobs. This is the action for the following signals as well: SIGXCPU, SIGXFSZ, SIGCPULIM, and SIGSHUTDN.
- SIGUSR1, SIGUSR2
    - Causes mom to increase and decrease logging levels, respectively.
- SIGPIPE, SIGINFO
    - Are ignored.
- SIGBUS, SIGFPE, SIGILL, SIGTRAP, and SIGSYS
    - Cause a core dump if the PBSCOREDUMP environmental variable is defined.

All other signals have their default behavior installed.

Exit Status

If the mini-server command fails to begin operation, the server exits with a value greater than zero.

See Also

pbs_server(8B), pbs_scheduler_basl(8B), pbs_scheduler_tcl(8B), the PBS External Reference Specification, and the PBS Administrators Guide.

pbs_server (PBS Server)

pbs_server - pbs batch system manager

SYNOPSIS

```
pbs_server [-a active] [-d config_path] [-p port] [-A acctfile]
           [-L logfile] [-M mom_port] [-R momRPP_port] [-S scheduler_port]
           [-h hostname] [-t type]
```

DESCRIPTION

The pbs_server command starts the operation of a batch server on the local host. Typically, this command will be in a local boot file such as /etc/rc.local . If the batch server is already in execution, pbs_server will exit with an error. To insure that the pbs_server command is not runnable by the general user community, the server will only execute if its real and effective uid is zero.

The server will record a diagnostic message in a log file for any error occurrence. The log files are maintained in the server_logs directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console.

OPTIONS

-a *active*

Specifies if scheduling is active or not. This sets the server attribute scheduling. If the option argument is "true" ("True", "t", "T", or "1"), the server is active and the PBS job scheduler will be called. If the argument is "false" ("False", "f", "F", or "0), the server is idle, and the scheduler will not be called and no jobs will be run. If this option is not specified, the server will retain the prior value of the scheduling attribute.

-d *config_path*

Specifies the path of the directory which is home to the servers configuration files, PBS_HOME. A host may have multiple servers. Each server must have a different configuration directory. The default configuration directory is given by the symbol $PBS_SERVER_HOME which is typically /usr/spool/PBS.

-p *port*

Specifies the port number on which the server will listen for batch requests. If multiple servers are running on a single host, each must have its own unique port number. This option is for use in testing with multiple batch systems on a single host.

-A *acctfile*

Specifies an absolute path name of the file to use as the accounting file. If not specified, the file is named for the current date in the PBS_HOME/server_priv/accounting directory.

-L *logfile*

Specifies an absolute path name of the file to use as the log file. If not specified, the file is one named for the current date in the PBS_HOME/server_logs directory, see the -d option.

-M *mom_port*

Specifies the host name and/or port number on which the server should connect the job executor, MOM. The option argument, mom_conn, is one of the forms: host_name, [:]port_number, or host_name:port_number. If host_name not specified, the local host is assumed. If port_number is not specified, the default port is assumed. See the -M option for pbs_mom(8).

-R *mom_RPPport*

Specifies the port number on which the the server should query the up/down status of Mom. See the -R option for pbs_mom(8).

-S *scheduler_port*

Specifies the port number to which the server should connect when contacting the Scheduler. The option argument, scheduler_conn, is of the same syntax as under the -M option.

-h *hostname*

Causes the server to start under a different hostname as obtained from gethostname(2). Useful for servers with multiple network interfaces to support connections from clients over an interface that has a hostname assigned that differs from the one that is returned by gethost name(2).

-t *type*

Specifies the impact on jobs which were in execution, running, when the server shut down. If the running job is not rerunnable or restartable from a checkpoint image, the job is aborted. If the job is rerunnable or restartable, then the actions described below are taken. When the type argument is:

hot

All jobs are requeued except non-rerunnable jobs that were executing. Any rerunnable job which was executing when the server went down will be run immediately. This returns the server to the same state as when it went down. After those jobs are restarted, then normal scheduling takes place for all remaining queued jobs.

If a job cannot be restarted immediately because of a missing resource, such as a node being down, the server will attempt to restart it periodically for upto 5 minutes. After that period, the server will revert to a normal state, as if warm started, and will no longer attempt to restart any remaining jobs which were running prior to the shutdown.

warm

All rerunnable jobs which were running when the server went down are requeued. All other jobs are maintained. New selections are made for which jobs are placed into execution. Warm is the default if -t is not specified.

cold

All jobs are deleted. Positive confirmation is required before this direction is accepted.

create

The server will discard any existing configuration files, queues and jobs, and initialize configuration files to the default values. The server is idled.

FILES

$PBS_SERVER_HOME/server_priv

default directory for configuration files, typically /usr/spool/pbs/server_priv

$PBS_SERVER_HOME/server_logs

directory for log files recorded by the server

SIGNAL HANDLING

On receipt of the following signals, the server performs the defined action:

SIGHUP

The current server log and accounting log are closed and reopened. This allows for the prior log to be renamed and a new log started from the time of the signal.

SIGINT

Causes an orderly shutdown of pbs_server.

SIGUSR1, SIGUSR2

Causes server to increase and decrease logging levels, respectively.

SIGTERM

Causes an orderly shutdown of pbs_server.

SIGSHUTDN

On systems (Unicos) where SIGSHUTDN is defined, it also causes an orderly shutdown of the server.

SIGPIPE

This signal is ignored.

All other signals have their default behavior installed.

EXIT STATUS

If the server command fails to begin batch operation, the server exits with a value greater than zero.

SEE ALSO

```
  qsub (1B), pbs_connect(3B), pbs_mom(8B), pbs_sched_basl(8B),
  pbs_sched_tcl(8B), pbsnodes(8B), qdisable(8B), qenable(8B), qmgr(1B),
  qrun(8B), qstart(8B), qstop(8B), qterm(8B), and the PBS External
Reference Specification.
```

pbs_track

starts a new process and informs pbs_mom to start tracking it

Synopsis

```
pbs_track -j <JOBID> [-b] a.out [args]
```

Description

The pbs_track command tells a pbs_mom daemon to monitor the lifecycle and resource usage of the process that it launches using exec(). The pbs_mom is told about this new process via the Task Manager API, using tm_adopt(). The process must also be associated with a job that already exists on the pbs_mom.

By default, pbs_track will send its PID to TORQUE via tm_adopt(). It will then perform an exec(), causing a.out to run with the supplied arguments. pbs_track will not return until the launched process has completed because it becomes the launched process.

This command can be considered related to the pbsdsh command which uses the tm_spawn() API call. The pbsdsh command asks a pbs_mom to launch and track a new process on behalf of a job. When it is not desirable or possible for the pbs_mom to spawn processes for a job, pbs_track can be used to allow an external entity to launch a process and include it as part of a job.

The original motivation behind creating this command was to improve integration with TORQUE and SGI's MPT MPI implementation.

Options

- -j <JOBID>
  - Job ID the new process should be associated with.
- -b
  - Instead of having pbs_track send its PID to TORQUE, it will fork() first, send the child PID to TORQUE, and then execute from the forked child. This essentially "backgrounds" pbs_track so that it will return after the new process is launched.

Operands

The pbs_track command accepts a path to a program/executable (a.out) and, optionally, one or more arguments to pass to that program.

Exit Status

Because the pbs_track command becomes a new process (if used without -b), its exit status will match that of the new process. If the -b option is used, the exit status will be zero if no errors occurred before launching the new process.

If pbs_track fails, whether due to a bad argument or other error, the exit status will be set to a non-zero value.

See Also

pbsdsh(1B), tm_spawn(3B)

Appendix B: Server Parameters

TORQUE server parameters are specified using the qmgr command. Specifically, inside the qmgr command, the **set** subcommand should be used to modify the **server** object as in

qmgr

```
> qmgr -c 'set server default_queue=batch'
```

Parameter	**Format**	**Default**	**Description**

acl_hosts	<HOST>[,<HOST>]...	(all hosts allowed to submit jobs)	specifies the list of hosts allowed to submit jobs. **NOTE:** can also be used to specify 'queue-to-host mapping'. (See TORQUE/OpenPBS Queue to Node Mapping in the Moab Workload Administrator's Guide for more information.) **Example:** acl_hosts
			
			`Qmgr: set queue batch acl_hosts = "hostA,hostB"`
			`Qmgr: set queue batch acl_hosts += "hostE,hostF,hostG"`

acl_host_enable	<BOOLEAN>	**FALSE**	specifies if the **acl_hosts** value is enabled
acl_logic_or	<BOOLEAN>	**FALSE**	specifies if user and group queue ACL's should be logically AND'd or logically OR'd
acl_roots	<username>@<domain>	**---**	specifies which root users are allowed to submit and run jobs.
allow_node_submit	<BOOLEAN>	**FALSE**	specifies if users can submit jobs directly from any trusted compute host directly

			or from within batch jobs (See Configuring Job Submit Hosts)
allow_proxy_user	<BOOLEAN>	**FALSE**	specifies users can proxy from one user to another. Proxy requests will be either validated by *ruserok()* or by the scheduler. (See Job Submission Configuration.)
auto_node_np	<BOOLEAN>	**DISABLED**	Automatically configure a node's np value based on the ncpus value from the status update. Requires full manager privilege to set or alter.
default_queue	<STRING>	---	indicates the queue to assign to a job if no queue is explicitly specified by the submitter
job_nanny	<BOOLEAN>	**FALSE**	Enables the experimental "job deletion nanny" feature. All job cancels will create a repeating task that will resend KILL signals if the initial job cancel failed. Further job cancels will be rejected with the message "job cancel in progress." This is useful for temporary failures with a job's execution node during a job delete request.
job_stat_rate	<INT>	45 (set to 30 in TORQUE 1.2.0p5 and earlier)	specifies the maximum age of mom level job data which is allowed when servicing a qstat request. If data is older than this value, the pbs_server daemon will contact mom's with stale data to request an update. **NOTE**: For large systems, this value should be increased

			to 5 minutes or higher.
log_level	<INT>	0	specifies the pbs_server logging verbosity. Maximum value is 7.
log_file_max_size	<INT>	0	specifies a soft limit, in kilobytes, for the server's log file. The filesize is checked every 5 minutes, and if the **current day** filesize is greater than or equal to this value then it will be rolled from X to X.1 and a new empty log will be opened. Any value ⇐ 0 will be ignored by pbs_server (the log will not be rolled).
log_file_roll_depth	<INT>	1	This parameter controlls how deep the **current day** log files will be rolled, if log_file_max_size is set, before they are deleted.
mail_domain	<STRING>	---	Override the default domain for outgoing mail messages. If set, emails will be addressed to "euser@mail_domain". If unset, the job's Job_Owner attribute will be used.
managers	`user@host.sub.domain[,user@host.sub.domain...]`	root on the local host	List of users granted batch administrator privileges. The host, sub-domain, or domain name may be wildcarded by the use of an asterisk character (*). Requires full manager privilege to set or alter.
mom_job_sync	<BOOLEAN>	**TRUE**	specifies that the pbs_server will synchronize its view of the job queue and resource allocation with compute nodes as they come online. If a job exists on a

			compute node in a pre-execution or corrupt state, it will be automatically cleaned up and purged. (enabled by default in TORQUE 2.2.0 and higher)
operators	`user@host.sub.domain[,user@host.sub.domain...]`	root on the local host	List of users granted batch operator privileges. Requires full manager privilege to set or alter.
node_check_rate	<INT>	600	specifies the minimum duration (in seconds) that a node can be unresponsive to server queries before being marked down by the pbs_server daemon
node_pack	<BOOLEAN>	---	Controls how multiple processor nodes are allocated to jobs. If this attribute is set to true, jobs will be assigned to the multiple processor nodes with the fewest free processors. This packs jobs into the fewest possible nodes leaving multiple processor nodes free for jobs which need many processors on a node. If set to false, jobs will be scattered across nodes reducing conflicts over memory between jobs. If unset, the jobs are packed on nodes in the order that the nodes are declared to the server (in the nodes file). Default value: unset - assigned to nodes as nodes in order that were declared.
node_ping_rate	<INT>	300	specifies the maximum interval (in seconds) between successive *pings* sent from the **pbs_server** daemon to the

			pbs_mom daemon to determine node/daemon health.
poll_jobs	<BOOLEAN>	**TRUE** (FALSE in TORQUE 1.2.0p5 and earlier)	if set to **TRUE**, pbs_server will poll job info from mom's over time and will not block on handling requests which require this job information. If not set, no polling will occur and if job information is requested which is stale, pbs_server may block while it attempts to update this information. **NOTE**: For large systems, this value should be set to **TRUE**.
query_other_jobs	<BOOLEAN>	**FALSE**	specifies whether or not non-admin users may view jobs they do not own
resources_available	<STRING>	---	allows overriding of detected resource quantity limits (see queue resources_available) **NOTE:** pbs_server must be restarted for changes to take effect. Also, resources_available is constrained by the smallest of queue.resources_available and the server.resources_available.
submit_hosts	"<HOSTNAME>[,<HOSTNAME>]..."	---	specifies the list of hosts beyonds the host running **pbs_server** which can submit batch or interactive jobs. (see Configuring Job Submit Hosts)
tcp_timeout	<INT>	8	specifies the **pbs_server** to **pbs_mom** TCP socket timeout in seconds. (see

			Considerations for Large Clusters)

These parameters are set using the qmgr command. For example,

```
> qmgr -c set server tcp_timeout=8
```

Appendix C: Node Manager (MOM) Configuration

- C.1 Parameters
- C.2 Node Features and Generic Consumable Resource Specification
- C.3 Command Line Arguments

Under TORQUE, MOM configuration is accomplished using the `mom_priv/config` file located in the PBS directory on each execution server.

C.1 Parameters

parameter	format	description	example
arch	<STRING>	specifies the architecture of the local machine. This information is used by the scheduler only.	`arch ia64`
$clienthost	<STRING>	specifies the machine running **pbs_server** (**NOTE:** This parameter is deprecated, use pbsserver)	`$clienthost node01.teracluster.org`
$configversion	<STRING>	specifies the version of the config file data	`$configversion 113`
$cputmult	<FLOAT>	cpu time multiplier. **NOTE:** if set to 0.0, MOM level cputime enforcement is disabled.	`$cputmult 2.2`
$ideal_load	<FLOAT>	ideal processor load	`$ideal_load 4.0`
$ignwalltime	<BOOLEAN>	ignore walltime (do not enable mom based walltime limit enforcement)	`$ignwalltime true`
$job_output_file_umask	<STRING>	uses the specified umask when creating job output and error files. Values can be specified in base 8, 10,	`$job_output_file_umask 027`

		or 16; leading 0 implies octal and leading 0x or 0X hexadecimal. A value of "userdefault" will use the user's default umask. This parameter is in version 2.3.0 and later.	
$logevent	<STRING>	specifies a bitmap for event types to log	`$logevent 255`
$loglevel	<INTEGER>	specifies the verbosity of logging with higher numbers specifying more verbose logging. Values may range between 0 and 7.	`$loglevel 4`
$log_file_max_size	<INTEGER>	Soft limit for log file size in kilobytes. Checked every 5 minutes. If the log file is found to be greater than or equal to log_file_max_size the current log file will be moved from X to X.1 and a new empty file will be opened.	`$log_file_max_size = 100`
$log_file_roll_depth	<INTEGER>	specifies how many times a log fill will be rolled before it is deleted.	`$log_file_roll_depth = 7`
$max_load	<FLOAT>	maximum processor load	`$max_load 4.0`
$node_check_script	<STRING>	specifies the fully qualified pathname of the health check script to run. (see Health Check for more information)	`$node_check_script /opt/batch_tools/nodecheck.pl`
$node_check_interval	<INTEGER>	specifies the number of MOM intervals between subsequent executions of the specified health check. This value default to 1 indicating the check is run every mom interval. (see Health Check for more information)	`$node_check_interval 5`
$nodefile_suffix	<STRING>	Specifies the suffix to append to a host names to denote the data channel network	`$nodefile_suffix i` With the suffix of 'i' and the

		adapter in a multihomed compute node.	control channel adapter with the name `node01`, the data channel would have a hostname of `node01i`.
opsys	<STRING>	specifies the operating system of the local machine. This information is used by the scheduler only.	`opsys RHEL3`
$pbsclient	<STRING>	specifies machines which the mom daemon will trust to run resource manager commands via momctl. This may include machines where monitors, schedulers, or admins require the use of this command.)	`$pbsclient node01.teracluster.org`
$pbsserver	<STRING>	specifies the machine running **pbs_server** (**NOTE:** This parameter replaces the deprecated parameter clienthost)	`$pbsserver node01.teracluster.org`
$prologalarm	<INTEGER>	Specifies maximum duration (in seconds) which the mom will wait for the job prolog or job job pilog to complete. This parameter default to 300 seconds (5 minutes)	`$prologalarm 60`
$rcpcmd	<STRING>	specifies the full path and optional additional command line args to use to perform remote copies	mom_priv/config `$rcpcmd /usr/local/bin/scp -i /etc/sshauth.dat`
$remote_reconfig	<STRING>	Enables the ability to remotely reconfigure pbs_mom with a new config file. Default is disabled. This parameter accepts various forms of true, yes, and 1.	`$remote_reconfig true`
$restricted	<STRING>	Specifies hosts which can be trusted to access mom services as non-root. By default, no hosts are trusted to access mom	`$restricted *.teracluster.org`

		services as non-root.	
size[fs=<FS>]	N/A	Specifies that the available and configured disk space in the <FS> filesystem is to be reported to the pbs_server and scheduler. **NOTE**: To request disk space on a per job basis, specify the `file` resource as in '`qsub -l nodes=1,file=1000kb`' **NOTE**: unlike most mom config options, the **size** parameter is not preceded by a '**$**' character.	`size[fs=/localscratch]` the available and configured disk space in the `/localscratch` filesystem will be reported.
$source_login_batch	<STRING>	Specifies whether or not mom will source the /etc/profile, etc. type files for **batch** jobs. Parameter accepts various forms of true, false, yes, no, 1 and 0. Default is True. This parameter is in version 2.3.1 and later.	`$source_login_batch False` mom will bypass the sourcing of /etc/profile, etc. type files.
$source_login_interactive	<STRING>	Specifies whether or not mom will source the /etc/profile, etc. type files for **interactive** jobs. Parameter accepts various forms of true, false, yes, no, 1 and 0. Default is True. This parameter is in version 2.3.1 and later.	`$source_login_interactive False` mom will bypass the sourcing of /etc/profile, etc. type files.
$status_update_time	<INTEGER>	Specifies the number of seconds between subsequent mom-to-server update reports. Default is 45 seconds	status_update_time `$status_update_time 120` mom will send server update reports every 120 seconds.
$timeout	<INTEGER>	Specifies the number of seconds before mom-to-mom messages will timeout if RPP is disabled. Default is 60 seconds	`timeout 120` mom-to-mom communication will allow up to 120 seconds before timing out.
$tmpdir	<STRING>	Specifies a directory to create job-specific scratch space (see	`$tmpdir /localscratch`

		Creating Per-Job Temporary Directories	
$usecp	<HOST>:<SRCDIR> <DSTDIR>	Specifies which directories should be staged (see TORQUE Data Management)	`$usecp *.fte.com:/data /usr/local/data`
$wallmult	<FLOAT>	wall time multiplier. NOTE: if set to 0.0, MOM level walltime enforcement is disabled.	`$wallmult 2.2`

C.2 Node Features and Generic Consumable Resource Specification

Node features (a.k.a. **node properties**) are *opaque* labels which can be applied to a node. They are not consumable and cannot be associated with a value. (use generic resources described below for these purposes). Node features are configured within the global nodes file on the pbs_server head node and are not specified on a per node basis. This file can be used to specify an arbitrary number of node features.

Additionally, per node *consumable* generic resources may be specified using the format '<ATTR> <VAL>' with no leading dollar ('$') character. When specified, this information is routed to the scheduler and can be used in scheduling decisions. For example, to indicate that a given host has two tape drives and one node-locked matlab license available for batch jobs, the following could be specified:

mom_priv/config

```
$clienthost 241.13.153.7

tape    2
matlab  1
```

Dynamic Consumable Generic Resources

Dynamic consumable resource information can be routed in by specifying a value preceded by a *exclamation point* (i.e., '!') as in the example below. If the resource value is configured in this manner, the specified file will be periodically executed to load the effective resource value. (See section 2.5.3 of the 'PBS Administrator Guide' for more information)

mom_priv/config

```
$clienthost 241.13.153.7

tape    !/opt/rm/gettapecount.pl
```

```
matlab  !/opt/tools/getlicensecount.pl
```

C.3 Command Line Arguments

a <integer>	alarm time in seconds
c <file>	configfile
C <directory>	checkpoint path
d <directory>	home directory
L <file>	logfile
M <integer>	MOM port to listen on
p	perform 'poll' based job recovery on restart (jobs persist until associated processes terminate)
r	perform level 1 job recovery on restart
R <integer>	MOM 'RM' port to listen on
S <integer>	pbs_server port to connect to
v	display version information and exit
x	disable use of privileged port
?	show usage information and exit

See Also

- Server Commands
- Setting up Prolog and Epilog Scripts

TORQUE Resource Manager

Appendix D: Error Codes and Diagnostics

D.1 TORQUE Error Codes

Error Code Number	Error Code Name	Description
PBSE_NONE	15000	no error
PBSE_UNKJOBID	15001	Unknown Job Identifier
PBSE_NOATTR	15002	Undefined Attribute
PBSE_ATTRRO	15003	attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	No permission
PBSE_BADHOST	15008	access from host not allowed
PBSE_JOBEXIST	15009	job already exists
PBSE_SYSTEM	15010	system error occurred

PBSE_INTERNAL	15011	internal server error occurred
PBSE_REGROUTE	15012	parent job of dependent in rte queue
PBSE_UNKSIG	15013	unknown signal name
PBSE_BADATVAL	15014	bad attribute value
PBSE_MODATRRUN	15015	Cannot modify attribute in run state
PBSE_BADSTATE	15016	request invalid for job state
PBSE_UNKQUE	15018	Unknown queue name
PBSE_BADCRED	15019	Invalid Credential in request
PBSE_EXPIRED	15020	Expired Credential in request
PBSE_QUNOENB	15021	Queue not enabled
PBSE_QACESS	15022	No access permission for queue
PBSE_BADUSER	15023	Bad user - no password entry
PBSE_HOPCOUNT	15024	Max hop count exceeded
PBSE_QUEEXIST	15025	Queue already exists
PBSE_ATTRTYPE	15026	incompatible queue attribute type
PBSE_QUEBUSY	15027	Queue Busy (not empty)
PBSE_QUENBIG	15028	Queue name too long
PBSE_NOSUP	15029	Feature/function not supported
PBSE_QUENOEN	15030	Cannot enable queue,needs add def
PBSE_PROTOCOL	15031	Protocol (ASN.1) error
PBSE_BADATLST	15032	Bad attribute list structure
PBSE_NOCONNECTS	15033	No free connections
PBSE_NOSERVER	15034	No server to connect to
PBSE_UNKRESC	15035	Unknown resource
PBSE_EXCQRESC	15036	Job exceeds Queue resource limits
PBSE_QUENODFLT	15037	No Default Queue Defined
PBSE_NORERUN	15038	Job Not Rerunnable
PBSE_ROUTEREJ	15039	Route rejected by all destinations
PBSE_ROUTEEXPD	15040	Time in Route Queue Expired
PBSE_MOMREJECT	15041	Request to MOM failed
PBSE_BADSCRIPT	15042	(qsub) cannot access script file
PBSE_STAGEIN	15043	Stage In of files failed
PBSE_RESCUNAV	15044	Resources temporarily unavailable
PBSE_BADGRP	15045	Bad Group specified
PBSE_MAXQUED	15046	Max number of jobs in queue
PBSE_CKPBSY	15047	Checkpoint Busy, may be retries
PBSE_EXLIMIT	15048	Limit exceeds allowable
PBSE_BADACCT	15049	Bad Account attribute value
PBSE_ALRDYEXIT	15050	Job already in exit state
PBSE_NOCOPYFILE	15051	Job files not copied
PBSE_CLEANEDOUT	15052	unknown job id after clean init

PBSE_NOSYNCMSTR	15053	No Master in Sync Set
PBSE_BADDEPEND	15054	Invalid dependency
PBSE_DUPLIST	15055	Duplicate entry in List
PBSE_DISPROTO	15056	Bad DIS based Request Protocol
PBSE_EXECTHERE	15057	cannot execute there
PBSE_SISREJECT	15058	sister rejected
PBSE_SISCOMM	15059	sister could not communicate
PBSE_SVRDOWN	15060	requirement rejected -server shutting down
PBSE_CKPSHORT	15061	not all tasks could checkpoint
PBSE_UNKNODE	15062	Named node is not in the list
PBSE_UNKNODEATR	15063	node-attribute not recognized
PBSE_NONODES	15064	Server has no node list
PBSE_NODENBIG	15065	Node name is too big
PBSE_NODEEXIST	15066	Node name already exists
PBSE_BADNDATVAL	15067	Bad node-attribute value
PBSE_MUTUALEX	15068	State values are mutually exclusive
PBSE_GMODERR	15069	Error(s) during global modification of nodes
PBSE_NORELYMOM	15070	could not contact Mom
PBSE_NOTSNODE	15071	no time-shared nodes

Appendix E: Considerations Before Upgrading

TORQUE is flexible in regards to how it can be upgraded. In most cases, a TORQUE *shutdown* followed by a **configure**, **make**, **make install** procedure as documented in the TORQUE QuickStart Guide is all that is required. This process will preserve existing configuration and in most cases, existing workload.

A few considerations are included below:

- If upgrading from OpenPBS, PBSPro, or TORQUE 1.0.3 or earlier, queued jobs whether active or idle will be lost. In such situations, job queues should be completely drained of all jobs.
- If not using the **pbs_mom** -r or -p flag, running jobs may be lost. In such cases, running jobs should be allowed to completed or should be requeued before upgrading TORQUE.
- **pbs_mom** and **pbs_server** daemons of differing versions may be run together. However, not all combinations have been tested and unexpected failures may occur.

Upgrade Steps

1. build new release (do not install) - See TORQUE Quick Start Guide
2. stop all TORQUE daemons - See qterm and momctl -s
3. install new TORQUE - use **make install**
4. start all TORQUE daemons - See sections 7 and 8 of the TORQUE Quick Start Guide

Rolling Upgrade

Using the **enablemomrestart** option, allows automatic restarts of the MOM. When enabled, the MOM will check if its binary has been updated and will restart itself at a safe point when no jobs are running; thus making upgrades easier. This can be enabled in the MOM config file, but it is recommended to enable it with momctl.

1. Prepare the new version MOM package - See TORQUE Quick Start Guide
2. Install the MOM package on the compute nodes - See TORQUE Quick Start Guide
3. Run **momctl -q enablemomrestart=1 -h :ALL**

Appendix F: Large Clusters Considerations

F.1 Communication Overview

TORQUE has enhanced much of the communication found in the original OpenPBS project. This has resulted in a number of key advantages:

- Support for larger clusters
- Support for more jobs
- Support for larger jobs
- Support for larger messages

In most cases, enhancements made apply to all systems and no tuning is required. However, some changes have been made configurable to allow site specific modification. These parameters are documented in the table below.

parameter	format	default	description
tcp_timeout	<INTEGER>	6	This parameter specifies the pbs_server to pbs_mom TCP socket timeout in seconds. It is specified as a *server* attribute and is set using the **qmgr** command.

Appendix F: Large Clusters Considerations

F.1 Communication Overview

TORQUE has enhanced much of the communication found in the original OpenPBS project. This has resulted in a number of key advantages:

- Support for larger clusters
- Support for more jobs
- Support for larger jobs
- Support for larger messages

In most cases, enhancements made apply to all systems and no tuning is required. However, some changes have been made configurable to allow site specific modification. These parameters are documented in the table below.

parameter	format	default	description
tcp_timeout	<INTEGER>	6	This parameter specifies the pbs_server to pbs_mom TCP socket timeout in seconds. It is specified as a *server* attribute and is set using the **qmgr** command.

F.2 Scalability Guidelines

In very large clusters (in excess of 1,000 nodes), it may be advisable to additionally tune a number of communication layer timeouts. By default, PBS MOM daemons will timeout on inter-mom messages after 60 seconds. In TORQUE 1.1.0p5 and higher, this can be adjusted by setting the **timeout** parameter in the mom_priv/config file. If 15059 errors (cannot receive message from sisters) are seen in the mom logs, it may be necessary to increase this value. **NOTE**: the **timeout** option is only effective if **RPP** is disabled.

Client-to-PBS server and mom-to-PBS server communication timeouts are specified via the tcp_timeout server option using the qmgr command.

On some systems, **ulimit** values may prevent large jobs from running. In particular, the open file descriptor limit (i.e., `ulimit -n`) should be set to at least the maximum job size in procs + 20. Further, there may be value in setting the `fs.file-max` in `sysctl.conf` to a high value, i.e.,

/etc/sysctl.conf

```
fs.file-max = 65536
```

F.3 End User Command Caching

F.3.1 qstat

In a large system, users may tend to place excessive load on the system by manual or automated use of resource manager end user client commands. A simple way of reducing this load is through the use of client command wrappers which cache data. The example script below will cache the output of the command '`qstat -f`' for 60 seconds and report this info to end users.

```
#!/bin/sh

# USAGE: qstat $@
```

```
CMDPATH=/usr/local/bin/qstat
CACHETIME=60
TMPFILE=/tmp/qstat.f.tmp

if [ "$1" != "-f" ] ; then
  #echo "direct check (arg1=$1) "
  $CMDPATH $1 $2 $3 $4
  exit $?
fi

if [ -n "$2" ] ; then
   #echo "direct check (arg2=$2)"
   $CMDPATH $1 $2 $3 $4
   exit $?
fi

if [ -f $TMPFILE ] ; then
  TMPFILEMTIME=`stat -c %Z $TMPFILE`
else
  TMPFILEMTIME=0
fi

NOW=`date +%s`

AGE=$(($NOW - $TMPFILEMTIME))

#echo AGE=$AGE

for i in 1 2 3;do
  if [ "$AGE" -gt $CACHETIME ] ; then
    #echo "cache is stale "

    if [ -f $TMPFILE.1 ] ; then
      #echo someone else is updating cache

      sleep 5

      NOW=`date +%s`

      TMPFILEMTIME=`stat -c %Z $TMPFILE`

      AGE=$(($NOW - $TMPFILEMTIME))
    else
      break;
    fi
  fi
done

if [ -f $TMPFILE.1 ] ; then
  #echo someone else is hung

  rm $TMPFILE.1
fi

if [ "$AGE" -gt $CACHETIME ] ; then
  #echo updating cache
  $CMDPATH -f > $TMPFILE.1

  mv $TMPFILE.1 $TMPFILE

fi

#echo "using cache"

cat $TMPFILE

exit 0
```

The above script can easily be modified to cache any command and any combination of args by changing one or more of the following attributes:

- script name
- value of $CMDPATH
- value of $CACHETIME
- value of $TMPFILE

For example, to cache the command `pbsnodes -a`, make the following changes:

- move original `pbsnodes` command to `pbsnodes.orig`
- save the script as '`pbsnodes`'
- change $CMDPATH to `pbsnodes.orig`
- change $TMPFILE to `/tmp/pbsnodes.a.tmp`

F.5 Other Considerations

F.5.1 job_stat_rate

In a large system, there may be many users, many jobs, and many requests for information. To speed up response time for users and for programs using the API the job_stat_rate can be used to tweak when the pbs_server daemon will query moms for job information. By increasing this number, a system will not be constantly querying job information and causing other commands to block.

F.5.2 poll_jobs

The poll_jobs parameter allows a site to configure how the pbs_server daemon will poll for job information. When set to **TRUE**, the pbs_server will poll job information in the background and not block on user requests. When set to **FALSE**, the pbs_server may block on user requests when it has stale job information data. Large clusters should set this parameter to **TRUE**.

F.5.3 Internal Settings

On large, slow, and/or heavily loaded systems, it may be desirable to increase the **pbs_tcp_timeout** setting used by the **pbs_mom** daemon in mom-to-mom communication. (**NOTE:** A system may be heavily loaded if it reports multiple 'End of File from addr' or 'Premature end of message' failures in the **pbs_mom** or **pbs_server** logs) This setting defaults to 20 seconds and requires rebuilding code to adjust. For client-server based communication, this attribute can be set using the qmgr command. For mom-to-mom communication, a source code modification is required. To make this change, edit the `$TORQUEBUILDDIR/src/lib/Libifl/tcp_dis.c` file and set **pbs_tcp_timeout** to the desired maximum number of seconds allowed for a mom-to-mom request to be serviced.

F.5.4 Scheduler Settings

If using Moab, there are a number of parameters which can be set which may improve TORQUE performance. In an environment containing a large number of short-running jobs, the JOBAGGREGATIONTIME parameter (see Appendix F of the Moab Workload Manager Administrator's Guide) can be set to reduce the number of workload and resource queries performed by the scheduler when an event based interface is enabled. If the **pbs_server** daemon is heavily loaded and PBS API timeout errors (ie. 'Premature end of message') are reported within the scheduler, the **TIMEOUT** attribute of the RMCFG parameter (see Appendix F of the Moab Workload Manager Administrator's Guide) may be set with a value of between 30 and 90 seconds.

F.5.5 File System

TORQUE can be configured to disable file system blocking until data is physically written to the disk by using the `--disable-filesync` argument with **configure**. While having filesync enabled is more reliable, it may lead to server delays for sites with either a larger number of nodes, or a large number of jobs. Filesync is enabled by default.

F.5.6 Network ARP cache

For networks with more than 512 nodes it is mandatory to increase the kernel’s internal ARP cache size. For a network of ~1000 nodes, we use these values in **/etc/sysctl.conf** on all nodes and servers:

/etc/sysctl.conf

```
# Don't allow the arp table to become bigger than this
net.ipv4.neigh.default.gc_thresh3 = 4096
# Tell the gc when to become aggressive with arp table cleaning.
# Adjust this based on size of the LAN.
net.ipv4.neigh.default.gc_thresh2 = 2048
# Adjust where the gc will leave arp table alone
net.ipv4.neigh.default.gc_thresh1 = 1024
# Adjust to arp table gc to clean-up more often
net.ipv4.neigh.default.gc_interval = 3600
# ARP cache entry timeout
```

```
net.ipv4.neigh.default.gc_stale_time = 3600
```

Use sysctl -p to reload this file.

The ARP cache size on other UNIXes can presumably be modified in a similar way;

An alternative approach is to have a static **/etc/ethers** file with all hostnames and MAC addresses and load this by **arp -f /etc/ethers**. However, maintaining this approach is quite cumbersome when nodes get new MAC addresses (due to repairs, for example).

Appendix G: Prologue & Epilogue Scripts

TORQUE provides to ability to run scripts before and/or after each job executes. With such a script, a site can prepare systems, perform node health checks, prepend and append text to output and error log files, cleanup systems, etc.

The table below shows which scripts will be run by which mom. All scripts must be in the $PBS_HOME/mom_priv/ directory and be available on *every* compute node. $PBS_HOME by default is `/usr/spool/PBS/`. *Mother Superior* is the pbs_mom on the first node allocated. *Sisters* refer to all other pbs_moms. $USER_HOME means the $HOME directory of the user running the job.

Script	**Execution Location**	**Privileges**	**Execution Directory**	**File Permissions**
prologue	Mother Superior	root	$PBS_HOME/mom_priv/	readable and executable by root and NOT writable by anyone besides root (e.g., -r-x------)
epilogue	Mother Superior	root	$USER_HOME	readable and executable by root and NOT writable by anyone besides root (e.g., -r-x------)
prologue.<name>	Mother Superior	root	$PBS_HOME/mom_priv/	readable and executable by root and NOT writable by anyone besides root (e.g., -r-x------)
epilogue.<name>	Mother Superior	root	$PBS_HOME/mom_priv/	readable and executable by root and NOT writable by anyone besides root (e.g., -r-x------)
prologue.user	Mother Superior	user	$PBS_HOME/mom_priv/	readable and executable by root and other (e.g., -r-x---r-x)

epilogue.user	Mother Superior	user	$USER_HOME	readable and executable by root and other (e.g., `-r-x---r-x`)
prologue.parallel	Sisters	root	$PBS_HOME/mom_priv/	readable and executable by root and NOT writable by anyone besides root (e.g., `-r-x------`)
epilogue.parallel*	Sisters	root		readable and executable by root and NOT writable by anyone besides root (e.g., `-r-x------`)
epilogue.precancel	Mother Superior (**NOTE**: this script is run after a job cancel request is received from pbs_server and before a kill signal is sent to the job process)	root	$USER_HOME	readable and executable by root and NOT writable by anyone besides root (e.g., `-r-x------`)

* available in Version 2.1

G.1 Script Environment

The prolog and epilog scripts can be very simple. On most systems, the script must declare the execution shell using the **#!<SHELL>** syntax (i.e., '#!/bin/sh'). In addition, the script may want to process context sensitive arguments passed by TORQUE to the script.

Prolog Environment

The following arguments are passed to the prologue, prologue.user and prologue.parallel scripts:

argv[1]	job id
argv[2]	job execution user name
argv[3]	job execution group name
argv[4]	job name (TORQUE 1.2.0p4 and higher only)
argv[5]	list of requested resource limits (TORQUE 1.2.0p4 and higher only)
argv[6]	job execution queue (TORQUE 1.2.0p4 and higher only)
argv[7]	job account (TORQUE 1.2.0p4 and higher only)

Epilog Environment

TORQUE supplies the following arguments to the epilogue, epilogue.user, epilogue.precancel, and epilogue.parallel scripts:

argv[1]	job id
argv[2]	job execution user name
argv[3]	job execution group name
argv[4]	job name
argv[5]	session id
argv[6]	list of requested resource limits
argv[7]	list of resources used by job
argv[8]	job execution queue
argv[9]	job account

The **epilogue.precancel** script is run after a job cancel request is received by the MOM and before any signals are sent to job processes. If this script exists, it is run whether the canceled job was active or idle.

For all scripts, the environment passed to the script is empty. Also, standard input for both scripts is connected to a system dependent file. Currently, for all systems this is **/dev/null**. Except for the epilogue scripts of an interactive job, the standard output and error, are connected to input and error files associated with the job. For an interactive job, since the pseudo terminal connection is released after the job completes, the standard input and error point to **/dev/null**.

G.2 Per Job Prologue and Epilogue Scripts

TORQUE now supports per job prologue and epilogue scripts when using the **qsub -T** option. The scripts are `prologue.<name>` and `epilogue.<name>` where <name> is an arbitrary name. When submitting a job, the syntax is **qsub -T <name>**.

Example

```
$PBS_HOME/mom_priv/

drwxr-x--x 2 root root 28672 Sep  8 15:36 jobs
-r-x------ 1 root root   107 Sep  8 16:31 prologue.prescript

$ qsub -T prescript jobscript.sh
```

G.3 Prologue and Epilogue Scripts Time Out

TORQUE takes preventative measures against prologue and epilogue scripts by placing an alarm around the scripts execution. By default, TORQUE sets the alarm to go off after 5 minutes of execution. If the script exceeds this time, it will be terminated and the node will be marked down. This timeout can be adjusted by setting the prologalarm parameter in the `mom_priv/config` file.

While TORQUE is executing the `epilog`, `epilog.user`, or `epilog.precancel` scripts, the job will be in the **E** (exiting) state.

G.4 Prolog Error Processing

If the prolog script executes successfully, it should exit with a zero status. Otherwise, the script should return the appropriate error code as defined in the table below. The **pbs_mom** will report the script's exit status to pbs_server which will in turn take the associated action. The following table describes each exit code for the prologue scripts and the action taken.

Error	Description	Action
-4	The script timed out	Job will be requeued
-3	The wait(2) call returned an error	Job will be requeued
-2	Input file could not be opened	Job will be requeued
-1	Permission error (script is not owned by root, or is writable by others)	Job will be requeued
0	Successful completion	Job will run
1	Abort exit code	Job will be aborted
>1	other	Job will be requeued

Example 1

Below are example prologue and epilogue scripts that write the arguments passed to them in the job's standard out file:

Script		stdout

prologue	`#!/bin/sh`
	
	`echo "Prologue Args:"`
	`echo "Job ID: $1"`
	`echo "User ID: $2"`
	`echo "Group ID: $3"`
	`echo ""`
	
	`exit 0`
		`Prologue Args:`
		`Job ID: 13724.node01`
		`User ID: user1`
		`Group ID: user1`

epilogue	`#!/bin/sh`
	
	`echo "Epilogue Args:"`
	`echo "Job ID: $1"`
	`echo "User ID: $2"`
	`echo "Group ID: $3"`
	`echo "Job Name: $4"`
	`echo "Session ID: $5"`
		`Epilogue Args:`
		`Job ID: 13724.node01`
		`User ID: user1`
		`Group ID: user1`
		`Job Name: script.sh`
		`Session ID: 28244`
		`Resource List:`
		`neednodes=node01,nodes=1,walltime=00:01:00`
		`Resources Used:`
		`cput=00:00:00,mem=0kb,vmem=0kb,walltime=00:00:07`
		`Queue Name: batch`
		`Account String:`

	`echo "Resource`
	`List: $6"`
	`echo "Resources`
	`Used: $7"`
	`echo "Queue`
	`Name: $8"`
	`echo "Account`
	`String: $9"`
	`echo ""`
	
	`exit 0`	

Example 2

The Ohio Supercomputer Center contributed the following scripts:
"prologue creates a unique temporary directory on each node assigned to a job before the job begins to run, and epilogue deletes that directory after the job completes. (Note that having a separate temporary directory on each node is probably not as good as having a good, high performance parallel filesystem.)"

prologue

```
#!/bin/sh
# Create TMPDIR on all the nodes
# Copyright 1999, 2000, 2001 Ohio Supercomputer Center
# prologue gets 3 arguments:
# 1 -- jobid
# 2 -- userid
# 3 -- grpid
#
jobid=$1
user=$2
group=$3
nodefile=/var/spool/pbs/aux/$jobid
if [ -r $nodefile ] ; then
    nodes=$(sort $nodefile | uniq)
else
    nodes=localhost
fi
tmp=/tmp/pbstmp.$jobid
for i in $nodes ; do
    ssh $i mkdir -m 700 $tmp \&\& chown $user.$group $tmp
done
exit 0
```

epilogue

```
#!/bin/sh
# Clear out TMPDIR
# Copyright 1999, 2000, 2001 Ohio Supercomputer Center
# epilogue gets 9 arguments:
# 1 -- jobid
# 2 -- userid
# 3 -- grpid
# 4 -- job name
# 5 -- sessionid
# 6 -- resource limits
# 7 -- resources used
# 8 -- queue
# 9 -- account
#
jobid=$1
nodefile=/var/spool/pbs/aux/$jobid
if [ -r $nodefile ] ; then
    nodes=$(sort $nodefile | uniq)
else
    nodes=localhost
fi
tmp=/tmp/pbstmp.$jobid
for i in $nodes ; do
    ssh $i rm -rf $tmp
done
exit 0
```

Prologue, prologue.user and prologue.parallel scripts can have dramatic effects on job scheduling if written improperly.

Appendix H: Running Multiple TORQUE Servers and Moms On the Same Node

By configuring, making, and installing TORQUE with different options, multiple servers and moms can be configured to run on the same node. This example will show how to configure, compile and install two different TORQUE servers and moms on the same node.

Configuring the First TORQUE

```
./configure --with-server-home=/usr/spool/PBS1 --bindir=/usr/spool/PBS1/bin -
-sbindir=/usr/spool/PBS1/sbin
```

Then *make* and *make install* will place the first TORQUE into "/usr/spool/PBS1" with the executables in their corresponding directories.

Configuring the Second TORQUE

```
./configure --with-server-home=/usr/spool/PBS2 --bindir=/usr/spool/PBS2/bin -
-sbindir=/usr/spool/PBS2/sbin
```

Then *make* and *make install* will place the second TORQUE into "/usr/spool/PBS2" with the executables in their corresponding directories.

Bringing the First TORQUE server online

Each command (including *pbs_server* and *pbs_mom* takes parameters indicating which servers and ports to connect to or listen on (when appropriate). Each of these is documented in their corresponding **man** pages (configure with --enable-docs).

In this example the first TORQUE server will accept batch requests on port 35000, communicate with the MOMS on port 35001, and communicate via RPP on port 35002. The first TORQUE mom will try to connect to the server on port 35000, it will listen for requests from the server on port 35001 and will communicate via RPP on port 35002. (Each of these command arguments is discussed in further details on the corresponding **man** page. In particular, `-t create` is only used the first time a server is run)

```
> pbs_server -p 35000 -M 35001 -R 35002 -t create
> pbs_mom -S 35000 -M 35001 -R 35002
```

Afterwords, when using a client command to make a batch request it is necessary to specify the servername and serverport (35000):

```
> pbsnodes -a -s node01:35000
```

Submitting jobs can be accomplished using the -q option ([queue][@host[:port]]):

```
> qsub -q @node01:35000 /tmp/script.pbs
```

Bringing the Second TORQUE Server Online

In this example the second TORQUE server will accept batch requests on port 36000, communicate with the MOMS on port 36002, and communicate via RPP on port 36002. The second TORQUE mom will try to connect to the server on port 36000, it will listen for requests from the server on port 36001 and will communicate via RPP on port 36002.

```
> pbs_server -p 36000 -M 36001 -R 36002 -t create
> pbs_mom -S 36000 -M 36001 -R 36002
```

Afterwords, when using a client command to make a batch request it is necessary to specify the servername and serverport (36002):

```
> pbsnodes -a -s node01:36000
> qsub -q @node01:36000 /tmp/script.pbs
```

Appendix I: Security Overview

- I.1 SUID Usage
- I.2 /etc/hosts Usage

I.1 SUID Usage

TORQUE uses setuid (SUID) permissions in a single location so as to validate the identity of a user request. This is accomplished using the **pbs_iff** tool which is SUID root and performs the following actions:

- parse specified server hostname and port
- connect to specified server port using reserved/privileged port
- determine UID of executing user
- report UID and socket port info of caller to server
- verify response from server

I.2 /etc/hosts Usage

In systems where security is a major concern, please be aware that some security experts consider adding the compute nodes to the /etc/hosts file to be more secure than using ACL lists.

Appendix J: Job Submission Filter (aka 'qsub Wrapper')*

When a *submit filter* exists, TORQUE will send the command file (or contents of STDIN if piped to qsub) to that script/executable and allow it to modify the submitted request based on specific site policies. The resulting file is then handed back to qsub and processing continues. This script should read the original submission request from STDIN and report the resulting request to STDOUT as in the following example:

```
#!/bin/sh

# add default memory constraints to all requests
# that did not specify it in user's script or on command line

echo "#PBS -l mem=16MB"

while read i
  do
    echo $i
  done
```

Command line arguments passed to qsub are passed as arguments to the submit filter (filter won't see them in STDIN) in the same order and may be used as needed. It should be noted that as of TORQUE 2.2.0 extended attributes are not passed to the filter. Exit status of -1 will cause qsub to reject the submission with a message stating that it failed due to administrative policies.

The *submit filter* must be executable, must be available on each of the nodes where users may submit jobs, and by default, must be located at `/usr/local/sbin/torque_submitfilter`. As of TORQUE 2.2.0, the build process will set the path to "${libexecdir}/qsub_filter". At run time, if the file does not exist at this new preferred path then qsub will fall back to the old hard-coded path. If needed, the submit filter location can be customized by setting the **SUBMITFILTER** parameter inside the torque.cfg file, as in the following example:

torque.cfg

```
SUBMITFILTER /opt/torque/submit.pl
...
```

*Initial development courtesy of Oak Ridge National Laboratories

Appendix K: torque.cfg File

parameter	format	default	description
ALLOWCOMPUTEHOSTSUBMIT	<BOOLEAN>	FALSE	If true, the server will trust job submit requests coming from compute nodes even if these nodes are not listed in `hosts.equiv`
QSUBHOST	<HOSTNAME>	none	Specify the hostname where pbs_mom will find qsub for interactive jobs.
QSUBSLEEP	<INT>	0	Specifies time to sleep when running qsub command, used to prevent users from overwhelming the scheduler.
SERVERHOST	<STRING>	localhost	If set, the server will open socket connections and

			communicate with client commands and other services using the specified network interface. (useful with multi-homed hosts, i.e., nodes with multiple network adapters)
SUBMITFILTER	<STRING>	/usr/local/sbin/torque_submitfilter	Specifies the location of the submit filter used to pre-process job submission.
SUBMITHOSTS	<HOSTNAME>[{,:}<HOSTNAME>]...	login1.ucc.edu,login2.ucc.edu	Specifies hosts which should be allowed to submit jobs without host validation using the ruserok() routine. This parameter can be used in conjunction with the **ALLOWCOMPUTEHOSTSUBMIT** parameter so that only non-compute hosts need to be explicitly specified.

The `torque.cfg` file should be placed in the TORQUE home directory (i.e., `/usr/spool/PBS`).

Example

torque.cfg

```
QSUBSLEEP              2
SERVERHOST             orion15
ALLOWCOMPUTEHOSTSUBMIT true
```

Appendix L: TORQUE Quick Start Guide

L.1 Initial Installation

- Download the TORQUE distribution file from http://clusterresources.com/downloads/torque
- Extract and build the distribution on the machine that will act as the "TORQUE server" - the machine that will monitor and control all compute nodes by running the **pbs_server** daemon. See the example below:

```
> tar -xzvf torque.tar.gz
> cd torque
> ./configure
> make
> make install
```

OSX 10.4 users need to change the *#define __TDARWIN* in `src/include/pbs_config.h` to *#define __TDARWIN_8*.

- **(OPTIONAL)** Set the **PATH** environment variable. The default installation directories for the binaries are either `/usr/local/bin` and `/usr/local/sbin`

In this document $(TORQUECFG) corresponds to where TORQUE stores its configuration files. This defaults to `/var/spool/torque`.

L.2 Initialize/Configure TORQUE on the Server (pbs_server)

- Once installation on the TORQUE server is complete, configure the pbs_server daemon by executing the command `torque.setup` <USER> found packaged with the distribution source code, where <USER> is a username that will act as the TORQUE admin. This script will set up a basic batch queue to get you started. If you experience problems, make sure that the most recent TORQUE executables are being executed, or that the executables are in your current PATH.
- If doing this step manually, be certain to run the command '**pbs_server -t create**' to create the new batch database. If this step is not taken, the **pbs_server** daemon will be unable to start.
- Proper server configuration can be verified by following the steps listed in Section 1.4 Testing

L.3 Install TORQUE on the Compute Nodes

To configure a compute node do the following on each machine (see page 19, Section 3.2.1 of PBS Administrator's Manual for full details):

- Create the self-extracting, distributable packages with `make packages` (See the `INSTALL` file for additional options and features of the distributable packages) and use the parallel shell command from your cluster management suite to copy and execute the package on all nodes (ie: xCAT users might do `prcp torque-package-linux-i686.sh main:/tmp/; psh main /tmp/torque-package-linux-i686.sh --install`. Optionally, distribute and install the `clients` package.

L.4 Configure TORQUE on the Compute Nodes

- For each compute host, the MOM daemon must be configured to trust the **pbs_server** daemon. In TORQUE 2.0.0p4 and earlier, this is done by creating the `$(TORQUECFG)/mom_priv/config` file and setting the $pbsserver parameter. In TORQUE 2.0.0p5 and later, this can also be done by creating the `$(TORQUECFG)/server_name` file and placing the server hostname inside.
- Additional config parameters may be added to `$(TORQUECFG)/mom_priv/config` (See the MOM Config page for details.)

L.5 Configure Data Management on the Compute Nodes

Data management allows jobs' data to be staged in/out or to and from the server and compute nodes.

- For shared filesystems (i.e., NFS, DFS, AFS, etc.) use the **$usecp** parameter in the `mom_priv/config` files to specify how to map a user's home directory.
  (Example: `$usecp gridmaster.tmx.com:/home /home`)
- For local, non-shared filesystems, rcp or scp must be configured to allow direct copy without prompting for passwords (key authentication, etc.)

L.6 Update TORQUE Server Configuration

- On the TORQUE server, append the list of newly configured compute nodes to the `$(TORQUECFG)/server_priv/nodes` file:

server_priv/nodes

```
computenode001.cluster.org
computenode002.cluster.org
computenode003.cluster.org
```

L.7 Start the pbs_mom Daemons on Compute Nodes

- Next start the **pbs_mom** daemon on each compute node by running the `pbs_mom` executable.

L.8 Verifying Correct TORQUE Installation

The **pbs_server** daemon was started on the TORQUE server when the `torque.setup` file was executed or when it was manually configured. It must now be restarted so it can reload the updated configuration changes.

```
# shutdown server
> qterm # shutdown server

# start server
> pbs_server

# verify all queues are properly configured
> qstat -q

# view additional server configuration
>  qmgr -c 'p s'

# verify all nodes are correctly reporting
> pbsnodes -a

# submit a basic job
```

```
> echo "sleep 30" | qsub

# verify jobs display
> qstat
```

At this point, the job will not start because there is no scheduler running. The scheduler is enabled in the next step below.

L.9 Enabling the Scheduler

Selecting the cluster scheduler is an important decision and significantly affects cluster utilization, responsiveness, availability, and intelligence. The default TORQUE scheduler, **pbs_sched**, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager are highly recommended. If using Maui/Moab, refer to the Moab-PBS Integration Guide. If using **pbs_sched**, start this daemon now.

If you are installing ClusterSuite, TORQUE and Moab were configured at installation for interoperability and no further action is required.

L.10 Startup/Shutdown Service Script for TORQUE/Moab (OPTIONAL)

An optional startup/shutdown service script is provided as an example of how to run TORQUE as an OS service that starts at bootup.

- Download the script here. (**NOTE:** this script was written specifically for Redhat variants, and may require modification to work with other Linux/UNIX distributions.)
- Place the file in `/etc/init.d/` directory
- Make symbolic links (S99moab and K15moab, for example) in desired runtimes (e.g. `/etc/rc.d/rc3.d/` on Redhat, etc.)

See Also:

- Advanced Server Configuration

Appendix M: User's Guide

1.0 Overview

1.1 Installation

- Download the TORQUE distribution file from http://clusterresources.com/downloads/torque (CHANGELOG)
- Extract and build the distribution on the machine that will act as the "TORQUE server" - the machine that will monitor and control all compute nodes by running the **pbs_server** daemon. See the example below (where XXX stands for the latest distribution (e.g., "-1.2.0p4"):

```
> tar -xzvf torqueXXX.tar.gz
> cd torqueXXX
> ./configure
> make
> make install
```

- **(OPTIONAL)** Set the **PATH** environment variable. The default installation directories for the binaries are either `/usr/local/bin` and `/usr/local/sbin`

See configure options for information on customizing the build at configure time.

In this document $(TORQUECFG) corresponds to where TORQUE stores its configuration files. This defaults to `/usr/spool/PBS`.

TORQUE 2.0p2 and higher includes a standard *spec* file for building your own rpms. It is also possible to use the checkinstall program to create your own RPM, tgz, or deb package.

1.1.1 Architecture

TORQUE has two main executable files, one for the head node **pbs_server** and one for each of the compute nodes **pbs_mom**. Therefore, TORQUE needs to be installed on a head node, every machine that jobs will run on and any machines that commands will be called from. When a job is run, the first node that the job is run on is designated as the Mother Superior. All other nodes for that job are designated as sister moms.

1.1.1 Compute Nodes

To install TORQUE on a compute node do the following on each machine (see page 19, Section 3.2.1 of PBS Administrator's Manual for full details):

- Create the self-extracting, distributable packages with `make packages` (See the `INSTALL` file for additional options and features of the distributable packages) and use the parallel shell command from your cluster management suite to copy and execute the package on all nodes (ie: xCAT users might do `prcp torque-package-linux-i686.sh main:/tmp/; psh main /tmp/torque-package-linux-i686.sh --install`. Optionally, distribute and install the `clients` package.
- Although *optional*, it is also possible to use the TORQUE server as a compute node and install a **pbs_mom** alongside the **pbs_server** daemon.

Example: Compute Node Installation

installing TORQUE compute nodes from the source directory

```
> make packages
Building ./torque-package-clients-linux-i686.sh ...
Building ./torque-package-mom-linux-i686.sh ...
Building ./torque-package-server-linux-i686.sh ...
Building ./torque-package-gui-linux-i686.sh ...
Building ./torque-package-devel-linux-i686.sh ...
Done.

The package files are self-extracting packages that can be copied
and executed on your production machines.  Use --help for options.

> cp torque-package-mom-linux-i686.sh /shared/storage
> cp torque-package-clients-linux-i686.sh /shared/storage
> dsh /shared/storage/torque-package-mom-linux-i686.sh --install
> dsh /shared/storage/torque-package-clients-linux-i686.sh --install
```

both **pbs_iff** and **pbs_rcp** will be installed *suid* root.

1.1.2 Enabling TORQUE/Moab as a Service (OPTIONAL)

An optional startup/shutdown service script is provided as an example of how to run TORQUE as an OS service that starts at bootup.

- Download the script here. (**NOTE:** This script was written specifically for Redhat variants, and may require modification to work with other Linux/UNIX distributions.)
- Place the file in `/etc/init.d/` directory
- Make symbolic links (`S99moab` and `K15moab`, for example) in desired runtimes (e.g. `/etc/rc.d/rc3.d/` on Redhat, etc.)
- This can be added to the self-extracting packages (See `INSTALL` for details.)

See Also

- TORQUE Installation TroubleShooting
- TORQUE Quick Start Guide

1.2 Basic Configuration

TORQUE only requires a few steps to get the system initially configured and running. First, the server needs to know which compute nodes it can trust. Second, the server should be initialized and configured. Finally, each of the compute nodes need to know where the **pbs_server** is located.

1.2.1 Specify Compute Nodes

In order for the **pbs_server** to communicate with each of the compute nodes, it needs to know which machines to contact. Each node which is to be a part of the cluster must be specified on a line in the server **nodes** file. This file is located at `$TORQUECFG/server_priv/nodes`. In most cases, it is sufficient to specify just the node name on a line as in the following example:

server_priv/nodes

```
node001
node002
node003
node004
```

If the compute node has multiple processors, specify the number of processors with `np=<#CPUs>`. For example, if `node001` has `2` processors and `node002` has `4`, use the following:

server_priv/nodes

```
node001 np=2
node002 np=4
...
```

For more information on this file, please see the **PBS Administrator Guide** or Server Node File Configuration.

1.2.2 Initialize/Configure TORQUE on the Server (pbs_server)

Once installation on the TORQUE server is complete, configure the **pbs_server** daemon by executing the command `torque.setup <USER>` found packaged with the distribution source code, where <USER> is a username that will act as the TORQUE admin. This script will setup a basic batch queue to get you started.

See section 1.4 Manual Setup of Initial Server Configuration if you would prefer to configure manually.

If you experience problems, make sure that the most recent TORQUE executables are being executed, or that the executables are in your current PATH.

Proper server configuration can be verified by following the steps listed in 1.5 Testing.

1.2.3 Initialize/Configure TORQUE on Each Compute Node

If using TORQUE's self extracting packages, and using default compute node configuration, no additional steps are required and the remainder of this section can be skipped.

If manual installation is occuring or more advanced compute node configuration is required, begin mom configuration by editing the `$(TORQUECFG)/mom_priv/config` file on each node. Recommended settings are as follows:

mom_priv/config

```
$pbsserver      10.10.10.100      # note: IP address of host running
pbs_server
```

Since this file is identical for all compute nodes, it can be created on the head node and distributed in parallel to all systems.

See Also

- MOM Configuration Overview
- Advanced Configuration (Customizing the Install)

1.3 Advanced Configuration

- 1.3.1 Customizing the Install
- 1.3.2 Server Configuration
    - o 1.3.2.1 Server Config Overview
    - o 1.3.2.2 Name Service Config
    - o 1.3.2.3 Configuring Job Submission Hosts
    - o 1.3.2.4 Configuring TORQUE on a Multi-Homed Server
    - o 1.3.2.5 Architecture Specific Notes

1.3.1 Customizing the Install

The TORQUE configure command takes a number of options. A few key options:

- By default, TORQUE does not install the admin manuals. To enable this, use '--enable-docs'
- By default, TORQUE uses **rcp** to copy data files. Using **scp** is highly recommended, use '--with-scp'
- By default, TORQUE does not enable **syslog** usage. To enable this, use '--enable-syslog'

Full Configure Option List

Option	Description
--enable-docs	build PBS with documentation. This is off by default since it takes a while to build. Man pages will still be installed, however.
--enable-server	build/not build the server. Can be used to disable the building of the server. Normally all components (docs, server, mom, clients) are built.
--enable-mom	build/not build the mom.
--enable-clients	build/not build the clients.
--with-tcl=DIR_PREFIX	Use this if you want Tcl support. If you are compiling with tclx then you need only use the --with-tcl flag if the Tcl libraries are in a different place than the tclx libraries. The prefix given here should be something like "/usr/local".
--with-tclx=DIR_PREFIX	Use this if you want TclX support. This implies --with-tcl. The only time it is useful to give both flags is if TclX and Tcl are in two different places. If no DIR_PREFIX is given then the libraries and includes must be installed in places

	automatically searched by system tools.
--enable-gui	build the xpbs GUI. This is on by default but can be turned off with --disable-gui.
--set-cc=ccprog	set the compiler to use. This will be used to set the CC variable and will override any CC setting in the environment. If only "--set-cc" is given then CC will be set to "cc".
--set-cflags=flags	set the compiler flags. This will be used to set the CFLAGS variable. If only "--set-flags" is given then CFLAGS will be set to "". Normally the flag settings are guessed by configure to be those appropriate for the architecture, compiler, and PBS machine type.
--enable-debug	turn on the DEBUG flag. Off by default.
--set-tmpdir=DIR	set the tmp directory that **pbs_mom** will use. This will default to "/tmp".
--set-server-home=DIR	set the server home/spool directory for PBS use this will default to /usr/spool/PBS if unspecified
--set-server-name-file=FILE	set the file that will contain the name of the default server for clients to use. If this is not an absolute pathname, it will be evaluated relative to the server home directory that either defaults to /usr/spool/PBS or is set using the --set-server-home option to configure. If this option is not specified, the default name for this file will be set to "server_name".
--set-default-server=HOSTNAME	set the name of the computer that clients will access when no machine name is specified as part of the queue name. It defaults to the hostname of the machine on which PBS is being compiled. **NOTE:** Do not set this to the string '**localhost**' as it will prevent you from configuring pbs_server using **qmgr**.
--set-environ=PATH	set the path containing the environment variables for the daemons. For SP2 and AIX systems, suggest setting to /etc/environment. Defaults to the file pbs_environment in the server-home. Relative paths are interpreted within the context of the server-home.
--enable-plock-daemons	enable daemons to lock themselves into memory logical-or of 1 for pbs_server, 2 for pbs_scheduler, 4 for **pbs_mom**. (e.g. set to 7 for all three)
--enable-syslog	enable the use of syslog for error reporting
--set-sched=TYPE	sets the scheduler type. If TYPE is "c" the scheduler will be written in C "tcl" the server will use a Tcl based scheduler "basl" will use the rule based scheduler "no" then their will be no scheduling done (the "c" scheduler is the default)
--set-sched-code=PATH	sets the name of the file or directory where the code for the scheduler is to be found. This only applies to BASL schedulers and those written in the C language. For C schedulers this should be a directory name and for BASL schedulers a filename ending in ".basl". If the path given is not absolute, it will be interpreted relative to srctree/src/schedulers.SCHD_TYPE/samples. As an example, an appropriate BASL scheduler relative path would be "nas.basl". The default scheduler code for "C" schedulers is "fifo".
--set-mansuffix=CHAR	set the man page suffix letter this will default to "B" if unspecified
--set-tclatrsep=CHAR	set the Tcl attribute separator character this will default to "." if unspecified
--set-qstatrc-file=FILE	set the name of the file that qstat will use if there is no ".qstatrc" file in the directory where it is being invoked. Relative path names will be evaluated relative to the server home directory that either defaults to /usr/spool/PBS or is set using the --set-server-home option to configure. If this option is not

	specified, the default name for this file will be set to "qstatrc" (no dot) in the server home directory.
--with-scp	use scp program to perform file delivery
--enable-shell-pipe	enable (default) this if you want to have the name of the job script passed to the shell via a pipe. If disabled, the behavior is to give the script file as standard input
--enable-rpp	use RPP/UDP for resource queries to mom. This is enabled by default. If disabled TCP is used.
--enable-sp2	setting this informs PBS that it is being built for an SP2.
--enable-tcl-qstat	setting this builds qstat with Tcl interpreter features. This is normally disabled and is only valid if --with-tcl(x) is already present.
--enable-array	setting this under IRIX enables the SGI Origin 2000 parallel support. Normally autodetected from the /etc/config/array file.
--enable-nodemask	setting this nodemask-based scheduling on the Origin 2000.
--enable-pemask	setting this enables pemask-based scheduling on the Cray T3e.
--enable-srfs	on the Crays you can enable support of SRFS. This is disabled by default
--enable-depend-cache	this turns on the ability to cache makedepend information across runs of configure. This can be bad if the user makes certain configuration changes in rerunning configure but it can save significant amounts of time in the hands of experienced developers. Changing the contents of Makefile.in will flush the cache in that directory. This is disabled by default
--disable-filesync	disable file system blocking until data is physically written to the disk. Filesync enabled is more reliable but may lead to server delays in larger systems. filesync is enabled by default

Recommended Configure Options Most recommended configure options have been selected as default. The few exceptions include '**--with-scp**' and possibly '**--enable-syslog**'.

1.3.2 Server Configuration

1.3.2.1 Server Config Overview

There are several steps to ensure that the server and the nodes are completely aware of each other and able to communicate directly. Some of this configuration takes place within TORQUE directly using the "*qmgr*" command. Other configuration settings are managed using the pbs_server nodes file, DNS files such as "/etc/hosts" and the "/etc/hosts.equiv" file.

1.3.2.2 Name Service Config

Each node, as well as the server, must be able to resolve the name of every node with which it will interact. This can be accomplished using "/etc/hosts", **DNS**, **NIS**, or other mechanisms. In the case of "/etc/hosts", the file can be shared across systems in most cases.

A simple method of checking proper name service configuration is to verify that the server and the nodes can "ping" each other.

1.3.2.3 Configuring Job Submission Hosts

Using RCmd Authentication

When jobs can be submitted from several different hosts, these hosts should be trusted via the R* commands (i.e., rsh, rcp, etc.). This can be enabled by adding the hosts to the "`/etc/hosts.equiv`" file of the machine executing the **pbs_server** daemon or using other R* command authorization methods. The exact specification can vary from OS to OS (see the man page for ***ruserok*** to find out how your OS validates remote users). In most cases, configuring this file is as simple as adding a line to your "`/etc/hosts.equiv`" as in the following:

hosts.equiv

```
#[+ | -] [hostname] [username]
mynode.myorganization.com
.....
```

Please note that when a hostname is specified, it must be the fully qualified domain name (FQDN) of the host. Job submission can be further secured using the server or queue acl_hosts and acl_host_enabled parameters.

Using the 'submit_hosts' Server Parameter

Trusted submit host access may be directly specified without using RCmd authentication by setting the server submit_hosts parameter via qmgr as in the example below:

qmgr

```
> qmgr -c 'set server submit_hosts = login1'
> qmgr -c 'set server submit_hosts += login2'
> qmgr -c 'set server submit_hosts += login3'
```

NOTE: use of **submit_hosts** is potentially subject to DNS spoofing and should not be used outside of controlled and trusted environments.

Allowing Job Submission from Compute Hosts

If desired, all compute nodes can be enabled as job submit hosts without setting `.rhosts` or `hosts.equiv` by setting the allow_node_submit parameter to **true**.

1.3.2.4 Configuring TORQUE on a Multi-Homed Server

If the **pbs_server** daemon is to be run on a multi-homed host (a host possessing multiple network interfaces), the interface to be used can be explicitly set using the SERVERHOST parameter.

1.3.2.5 Architecture Specific Notes

1.3.2.5.1 Mac OS/X Specific Notes

With some versions of Mac OS/X, it is required to add the line `$restricted *.<DOMAIN>` to the **pbs_mom** config file. This is required to work around some socket bind bugs in the OS.

1.3.2.6 Specifying Non-Root Administrators

By default, only root is allowed to start, configure and manage the **pbs_server** daemon. Additional trusted users can be authorized using the parameters **managers** and **operators**. To configure these parameters use the qmgr command as in the example below:

moab.cfg

```
> qmgr

Qmgr: set server managers += josh@*.fsc.com
Qmgr: set server operators += josh@*.fsc.com
```

All manager and operator specifications must include a user name and either a fully qualified domain name or a host expression.

NOTE: To enable all users to be trusted as both operators and admins, place the '+' character (plus) on its own line in the file `server_priv/acl_svr/operators` and `server_priv/acl_svr/managers`.

See Also

- Appendix B: Server Parameters

2.0 Submitting and Managing Jobs

2.1 Job Submission

Job submission is accomplished using the qsub command. This command takes a number of command line arguments and integrates this into the specified *PBS command file*. The PBS command file may be specified as a filename on the **qsub** command line or may be entered via STDIN.

- The PBS command file does not need to be executable.
- The PBS command file may be *piped* into qsub (i.e., '`cat pbs.cmd | qsub`')
- In the case of parallel jobs, the PBS command file is staged to, and executed on the first allocated compute node only. (use pbsdsh to run actions on multiple nodes)
- The command script is executed from the user's home directory in all cases (the script may determine the submission directory by using the $PBS_O_WORKDIR environment variable)

- The command script will be executed using the default set of user environment variables unless the '**-V**' or **-v** flags are specified to include aspects of the job submission environment.

By default, job submission is allowed only on the TORQUE server host (host on which **pbs_server** is running). Enablement of job submission from other hosts is documented in Configuring Job Submit Hosts.

2.1.1 Requesting Resources

Various resources can be requested at the time of job submission. A job can request a particular node, a particular node attribute, or even a number of nodes with particular attributes. Either native TORQUE resources, or external scheduler resource extensions may be specified. The native TORQUE resources are listed in the table below :

Resource	Format	Description
arch	string	Specifies the administrator defined system architecture required. This defaults to whatever the **PBS_MACH** string is set to in "local.mk".
cput	seconds, or [[HH:]MM:]SS	Maximum amount of CPU time used by all processes in the job
file	size*	The amount of total disk requested for the job
host	string	Name of the host on which the job should be run. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent.
mem	size*	Maximum amount of physical memory used by the job
nice	integer between -20 (highest priority) and 19 (lowest priority)	Adjust the process' execution priority
nodes	{<node_count> | <hostname>} [:ppn=<ppn>][:<property>[:<property>]...] [+ ...]	Number and/or type of nodes to be reserved for exclusive use by the job. The value is one or more node_specs joined with the '+' character, "node_spec[+node_spec...]". Each node_spec is an number of nodes required of the type declared in the node_spec and a name or one or more property or properties desired for the nodes. The number, the name, and each property in the node_spec are separated by a colon ':'. If no number is specified, one (1) is assumed. The name of a node is its hostname. The properties of nodes are:

		<ul><li>**ppn=#** - specify the number of processors per node requested. Defaults to 1.</li><li>property - a string assigned by the system administrator specify a node's features. Check with your administrator as to the node names and properties available to you.</li></ul>See Example 1 (-l nodes) for examples. **NOTE**: By default, the **node** resource is mapped to a virtual node or in other words, maps directly to a processor, not a full physical compute node. This behavior can be changed within Maui or Moab by setting the JOBNODEMATCHPOLICY parameter.
opsys	string	Specifies the administrator defined system operating system as defined in the mom **config** file.
other	string	Allows a user to specify site specific information. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent.
pcput	seconds, or [[HH:]MM:]SS	Maximum amount of CPU time used by any single process in the job
pmem	size*	Maximum amount of physical memory used by any single process of the job
pvmem	size*	Maximum amount of virtual memory used by any single process in the job
software	string	Allows a user to specify software required by the job. This is useful if certain software packages are only available on certain systems in the site. This resource is provided for use by the site's scheduling policy. The allowable values and effect on job placement is site dependent. (see Scheduler License Management)
vmem	size*	Maximum amount of virtual memory used by all concurrent processes in the job
walltime	seconds, or [[HH:]MM:]SS	Maximum amount of real time during which the job can be in the running state

***size** format:
The size format specifies the maximum amount in terms of bytes or words. It is expressed in the form *integer[suffix]*. The suffix is a multiplier defined in the following table ('b' means bytes (the

default) and 'w' means words). The size of a word is calculated on the execution server as its word size.

Suffix		Multiplier
b	w	1
kb	kw	1024
mb	mw	1,048,576
gb	gw	1,073,741,824
tb	tw	1,099,511,627,776

Example 1 (-l nodes)

Usage	Description
qsub -l `> qsub -l nodes=12`	request 12 nodes of any type
qsub -l `> qsub -l nodes=2:server+14`	request 2 "server" nodes and 14 other nodes (a total of 16) - this specifies two node_specs, "2:server" and "14"
qsub -l `> qsub -l nodes=server:hippi+10:noserver+3:bigmem:hippi`	request (a) 1 node that is a "server" and has a "hippi" interface, (b) 10 nodes that are not servers, and (c) 3 nodes that have a large amount of memory an have hippi
qsub -l `> qsub -l nodes=b2005+b1803+b1813`	request 3 specific nodes by hostname
qsub -l `> qsub -l nodes=4:ppn=2`	request 2 processors on each of four nodes
qsub -l `> qsub -l nodes=1:ppn=4`	request 4 processors on one node
qsub -l `> qsub -l nodes=2:blue:ppn=2+red:ppn=3+b1014`	request 2 processors on each of two blue nodes, three processors on one red node, and the compute node "b1014"

Example 2

```
> qsub -l mem=200mb /home/user/script.sh
```
This job requests a node with 200 MB of available memory.

Example 3

```
> qsub -l nodes=node01,mem=200mb /home/user/script.sh
```
This job will wait until node01 is free with 200 MB of available memory.

2.1.2 Requesting Generic Resources

When **generic** resources have been assigned to nodes using the server's nodes file, these resources can be requested at the time of job submission using the ***other*** field. (See the Consumable Generic Resources page for details on configuration within Moab).

Example 1

```
> qsub -l other=matlab /home/user/script.sh
```
This job will run on any node that has the generic resource ***matlab***.

NOTE: This can also be requested at the time of job submission using the ***-W x=GRES:matlab*** flag.

2.1.3 Requesting Floating Resources

When **floating** resources have been set up inside Moab, they can be requested in the same way as **generic** resources. Moab will automatically understand that these resources are **floating** and will schedule the job accordingly. (See the Floating Generic Resources page for details on configuration within Moab).

Example 2

```
> qsub -l other=matlab /home/user/script.sh
```
This job will run on any node when there are enough floating resources available.

NOTE: This can also be requested at the time of job submission using the ***-W x=GRES:matlab*** flag.

2.1.4 Requesting Other Resources

Many other resources can be requested at the time of job submission using the Moab Workload Manger. See the Resource Manager Extensions page for a list of these supported requests and correct syntax.

2.1.5 Exported Batch Environment Variables

When a batch job is started, a number of variables are introduced into the job's environment which can be used by the batch script in making decisions, creating output files, etc. These variables are listed in the table below:

Variable	**Description**
PBS_JOBNAME	user specified jobname
PBS_O_WORKDIR	user specified jobname
PBS_ENVIRONMENT	N/A
PBS_TASKNUM	number of tasks requested
PBS_O_HOME	home directory of submitting user
PBS_MOMPORT	active port for mom daemon
PBS_O_LOGNAME	name of submitting user
PBS_O_LANG	language variable for job
PBS_JOBCOOKIE	job cookie
PBS_NODENUM	node offset number
PBS_O_SHELL	script shell
PBS_O_JOBID	unique pbs job id
PBS_O_HOST	host on which job script is currently running
PBS_QUEUE	job queue
PBS_NODEFILE	file containing line delimited list on nodes allocated to the job
PBS_O_PATH	path variable used to locate executables within job script

See Also

- Many of these commands are also supported in Maui. See the Maui Documentation for details.
- qsub wrapper - Allow local checking and modification of submitted jobs

2.2 Specifying Files and Directories

- 2.2.1 Default Behavior
- 2.2.2 Job Environment Variables
- 2.2.3 Redirecting Output/Error Data
- 2.2.4 Staging Input and Output Files
- 2.2.5 Interactive Jobs
- 2.2.6 Specifying Working Directory
- 2.2.7 TORQUE Configuration Options

2.2.1 Default Behavior

The batch job submitted by a user will execute from within a specific directory. It will create both output and error files containing info written to stdout and stderr respectively. The directory in which the job executes will be the same as the directory from which the qsub command was executed.

2.2.2 Job Environment Variables

When a job is submitted, aspects of the calling environment are saved. When the job is actually executed, this environment may be restored (see the v and V qsub options)

Variable	Description
PBS_O_HOST	the name of the host where the qsub command was run
PBS_JOBID	the job identifier assigned to the job by the batch system
PBS_JOBNAME	the job name supplied by the user.
PBS_O_WORKDIR	the absolute path of the current working directory of the qsub command

See the qsub manpage for full list of available variables.

Example

torquejob.cmd

```
#!/bin/bash

echo "starting $PBS_JOBNAME" > /tmp/output.$PBS_JOBID
```

2.2.3 Redirecting Output/Error Data

By default, job output data will be placed in the file <SUBMITDIR>/<JOBNAME>.o<SEQUENCENUMBER> and job error data will be placed in the file <SUBMITDIR>/<JOBNAME>.e<SEQUENCENUMBER> These files will be populated and available to the user once the job completes.

This behavior can be changed by specifying the '-o' or '-e' arguments on the command line or within the command script as in the following example:

torquejob.cmd

```
#!/bin/bash

#PBS -e /tmp/error.$PBS_O_JOBID
#PBS -o /tmp/output.$PBS_O_JOBID
...
```

NOTE: If desired, the output and error streams can be *joined* by specifying the '-j' option.

NOTE: By default, output and error file information will be removed from the compute hosts when the job is complete. If desired, these files can be retained by specifying the '-k' option

See the qsub manpage for more information on the '-o', '-e', '-j' and '-k' options.

2.2.4 Staging Input and Output Files

If an input file is required by the job which is not by default available on the compute nodes, then the file can be staged in within the script or the script can request that the batch system copy the file on behalf of the user. If the latter approach is desired, the stagein option should be specified.

If one or more output files are produced by the job and must be staged to an accessible location, then the script can explicitly copy this data or can request the the batch system copy the file on behalf of the user. If the latter approach is desired, the stageout option should be specified.

FIXME from here down

torquejob.cmd

```
#!/bin/bash

#PBS -W stagein=dataset13442
#PBS -W stageout=/localscratch/data1.$PBS_O_JOBID
#PBS -W stageout=/localscratch/data2.$PBS_O_JOBID
...
```

See the qsub manpage for more details.

2.2.5 Interactive Jobs

A job run with the interactive option will run normally, but stdout and stderr will be connected directly to the users terminal. This also allows stdin from the user to be sent directly to the application.

qsub -I

```
> qsub -I /tmp/script
```

See the interactive option in the qsub manpage for more details.

2.2.6 Specifying Working Directory

The **-d** qsub option is used to define the working directory path to be used for the job. If the **-d** option is not specified, the default working directory is the home directory. This option sets the environment variable **PBS_O_INITDIR** and launches the job from the specified directory.

torquejob.cmd #!/bin/bash

```
#PBS -d /tmp/working_dir
...
```

2.2.7 TORQUE Configuration Options

2.2.7.1 Disabling Spooling - Allowing Users to Watch Output/Error Data in Real-time

Spooling can be disabled by using the qsub '**-k**' option. With this option, job output and error streams can be sent directly to a file in the job's working directory bypassing the intermediate spooling step. This is useful if the job is submitted from within a parallel filesystem or if the output is small and the user would like to view it in real-time. If the output is large and the remote working directory is not available via a high performance network, excessive use of this option may result in reduced cluster performance.

Example

qsub

```
> qsub -l nodes=1,walltime=1:00:00 -k oe file.cmd
```

Disabling Spooling at a Global Level

When the **--disable-spool configure** option is used, TORQUE will create the output and error files directly in `$HOME/.pbs_spool` if it exists, or in `$HOME` if it does not exist. By default, TORQUE will spool files in `$TORQUEHOME/spool` and copy them to the users home directory when the job completes.

NOTE: If desired for disk space or performance reasons, the default TORQUE spool directory can be changed by making `$TORQUEHOME/spool` a symbolic link to the desired target directory.

2.2.7.2 Creating Per-Job Temporary Directories

The `tmpdir` mom config option sets the directory basename for a per-job temporary directory. Before the job is launched, MOM will append the jobid to the tmpdir base-name and create the directory. After the job completes and exits, MOM will recursively delete that temporary directory. The env variable TMPDIR will be set for all pro/epilog scripts, the job script, and TM tasks.

Directory creation and removal is done as the job owner and group, so the owner must have write permission to create the directory. If the directory already exists and is owned by the job owner, it will not be deleted after the job. If the directory already exists and is NOT owned by the job owner, the job start will be rejected.

mom_priv/config

```
$tmpdir /localscratch
```

2.3 Canceling Jobs

TORQUE allows users and administrators to cancel submitted jobs with the qdel command. If the command is run by just a user, it will output just that user's jobs. For example,

```
Job id           Name             User             Time Use S Queue
---------------- ---------------- ---------------- -------- - -----
4807             scatter          user01           12:56:34 R batch
...
```

2.4 Job Preemption

TORQUE supports job preemption by allowing authorized users to suspend and resume jobs. This is supported using one of two methods. If the node supports OS-level preemption, TORQUE will recognize that during the `configure` process and enable it. Otherwise, the MOM may be configured to launch a custom *checkpoint script* in order to support preempting a job. Using a custom checkpoint script requires that the job understand how to resume itself from a checkpoint after the preemption occurs.

Configuring a Checkpoint Script on a MOM

To configure the MOM to support a checkpoint script, the `$checkpoint_script` parameter must be set in the MOM's configuration file found in `$TORQUEHOME/mom_priv/config`. The checkpoint script should have execute permissions set. A typical config file might look like:

mom_priv/config

```
$pbsserver          node06
$logevent           255
$restricted         *.mycluster.org
$checkpoint_script  /opt/moab/tools/mom-checkpoint.sh
```

The second thing that must be done to enable the checkpoint script is to change the value of `MOM_CHECKPOINT` to `1` in `.../src/include/pbs_config.h`. In some instances, `MOM_CHECKPOINT` may already be defined as `1`. The new line should be:

.../src/include/pbs_config.h

```
#define MOM_CHECKPOINT 1
```

2.5 Completed Jobs

TORQUE provides the ability to report on the status of completed jobs for a duration of five minutes after the job has completed. This can be enabled by setting the environment variable TORQUEKEEPCOMPLETED. By maintaining status information about completed (or canceled, failed...etc) jobs, administrators can better track failures and improve system performance. This also allows TORQUE to better communicate with Moab Workload Manager and track the status of jobs. This also gives Moab the ability to track specific failures schedule the cluster around possible hazards (see NODEFAILURERESERVETIME for more information).